



Dremio Software

# Reading Dremio Job Profiles

## Introduction

Dremio allows users to query and analyze data from various sources. A critical feature of Dremio is its ability to generate query profiles, which store metadata and data about the queries executed in the platform. In this document, we will delve into the functionality and purpose of Dremio profiles, exploring how they can be used to gain insights into performance, resource utilization and errors related to queries.

Regarding reviewing the details held within the profile, Dremio provides two approaches for accessing profiles: users can view the raw profile and a visual representation of the profile. Both methods are covered in this document.

This document presents various ways to view query execution details in Dremio, starting with the data on the Job Details page and drilling down into the more detailed profiles mentioned above. It also gives you an overview of Dremio's query processing architecture since understanding that architecture is vital to understanding what you see in a profile.

This document aims to provide you with a better understanding of the detailed information about queries in Dremio and where to find the key data points related to the queries you execute so that you start feeling empowered to investigate query issues for yourself.

## Job Details

Before discussing profiles, let's consider the information available in the Dremio User Interface for every executed job. When you click a job on the Jobs List page, you are shown a page like the following with a lot of in-depth information for that particular job.

There are several tabs along the top (Overview, SQL, Visual Profile, Raw Profile) and the contents of each tab are described in more depth in the following sections.

The screenshot shows the Dremio Job Details page for job ID 1a975b8e-a6a4-5090-707c-50b9a2208900. The Overview tab is active, showing a summary of the job's execution. The job is completed, with a total memory usage of 39.38 MB and a duration of less than 1 second. The submitted SQL is a query that lists privileges for users. The queried datasets are sys.privileges and sys.users. The scans section shows that the query scanned both datasets.

**Jobs** » 1a975b8e-a6a4-5090-707c-50b9a2208900 Overview SQL Profile Raw Profile

**Summary**

Status: COMPLETED  
 Total Memory: 39.38 MB  
 CPU Used: <1s  
 Query Type: UI (run)  
 Start Time: 30/11/2023 15:04:16  
 Duration: <1s  
 Wait on Client: <1s  
 User: dremio  
 Queue: Low Cost User Queries  
 Input: 0 B / 72 Rows  
 Output: 1.58 KB / 14 Rows

**Total Execution Time** <1s (100%)

Phase	Duration	Percentage
Metadata Retrieval	36ms	14.94%
Planning	80ms	33.20%
Queued	15ms	6.22%
Execution Planning	2ms	0.83%
Starting	3ms	1.24%
Running	105ms	43.57%

**Submitted SQL**

```
1 SELECT u.user_id, p.grantee_id, p.grantee_type, p.object_id, listagg(p.privilege, ',') AS privileges
2 FROM sys.privileges p INNER JOIN sys.users u
3   ON p.grantee_id = u.user_name
4 WHERE p.object_type = 'SCRIPT'
5 GROUP BY u.user_id, p.grantee_id, p.grantee_type, p.object_id
```

**Queried Datasets**

- privileges (sys.privileges)
- users (sys.users)

**Scans**

- > privileges
- > users

**Need a hand?**  
 Click "Download Profile" to download the query profile.

Download Profile

## Overview Tab

The default tab initially displayed is the Overview tab, shown above. There are various components or sections of information arranged on this tab. Each section is described below.

### Summary

In the summary section, you can see several high-level details related to the query. These are at-a-glance metrics to give you some initial characteristics of the query, such as what type of query it is, when it started, how long it took to run, who executed it, its peak memory consumption, as well as details of the Input and Output of the query, both in terms of bytes and rows; this is useful because it gives you an understanding of how many records were read

from the data sources and how many were ultimately exposed to clients, which provides a good idea of how much work Dremio needed to do. It's not uncommon, for example, to see queries that had an input of, say, 250M records and an output of, say, 14 records; this would be an indication that the query might not have had much of a filter or maybe the filter could not be pushed to the source and that Dremio was highly aggregating the data before being returned to the client.

## Total Execution Time

In the Total Execution Time section, you can get a very accurate picture of how long this query spent in each of the key execution phases as it planned, processed and ran the query. So, at a glance, you can see what the most significant time-consuming phases were, which can often be a first indicator as to where bottlenecks in your query could be.

## Submitted SQL

This tab section simply shows the exact SQL submitted to Dremio.

## Queried Datasets

This section summarizes which views were queried directly in the SQL statement shown in the Submitted SQL section. Under the Scans sub-heading, it also indicates which tables needed to be scanned for data.

If a reflection was used to accelerate this query, it reports which reflection(s) accelerated it in the Acceleration > Reflections Used sub-section.

If the optimizer considered a bunch of reflections but chose not to use them, then you will see all considered reflections in the Acceleration > Reflections Not Used sub-section.


## Profile Download:

If a query is not performing well or giving errors, you may want to bring this to Dremio's attention. Suppose you were discussing this with Dremio Support or Dremio Professional Services, for example. In that case, they are likely to request a copy of the profile associated with the issue so that they can look at the profile details offline from your environment. Clicking Download will automatically download a zip file containing the profile information, which can then be shared per any instructions you are given.

## SQL Tab

The SQL tab is relatively simple but provides a different window into the structure of the submitted query.

Jobs » 1a975b8e-a6a4-5090-707c-50b9a2208900 Overview SQL Profile Raw Profile

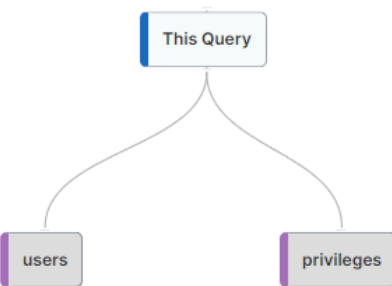
Submitted SQL 

```

1 SELECT u.user_id, p.grantee_id, p.grantee_type, p.object_id, listagg(p.privilege, ',') AS privileges
2 FROM sys.privileges p INNER JOIN sys.users u
3   ON p.grantee_id = u.user_name
4 WHERE p.object_type = 'SCRIPT'
5 GROUP BY u.user_id, p.grantee_id, p.grantee_type, p.object_id

```

Dataset Graph




```

graph TD
    ThisQuery[This Query] --- users[users]
    ThisQuery --- privileges[privileges]

```

## Submitted SQL

This is the same information that was shown in the Submitted SQL section of the Overview tab. If you want to copy your SQL text, use the Copy icon to the right of the section name (  ).

## Dataset Graph

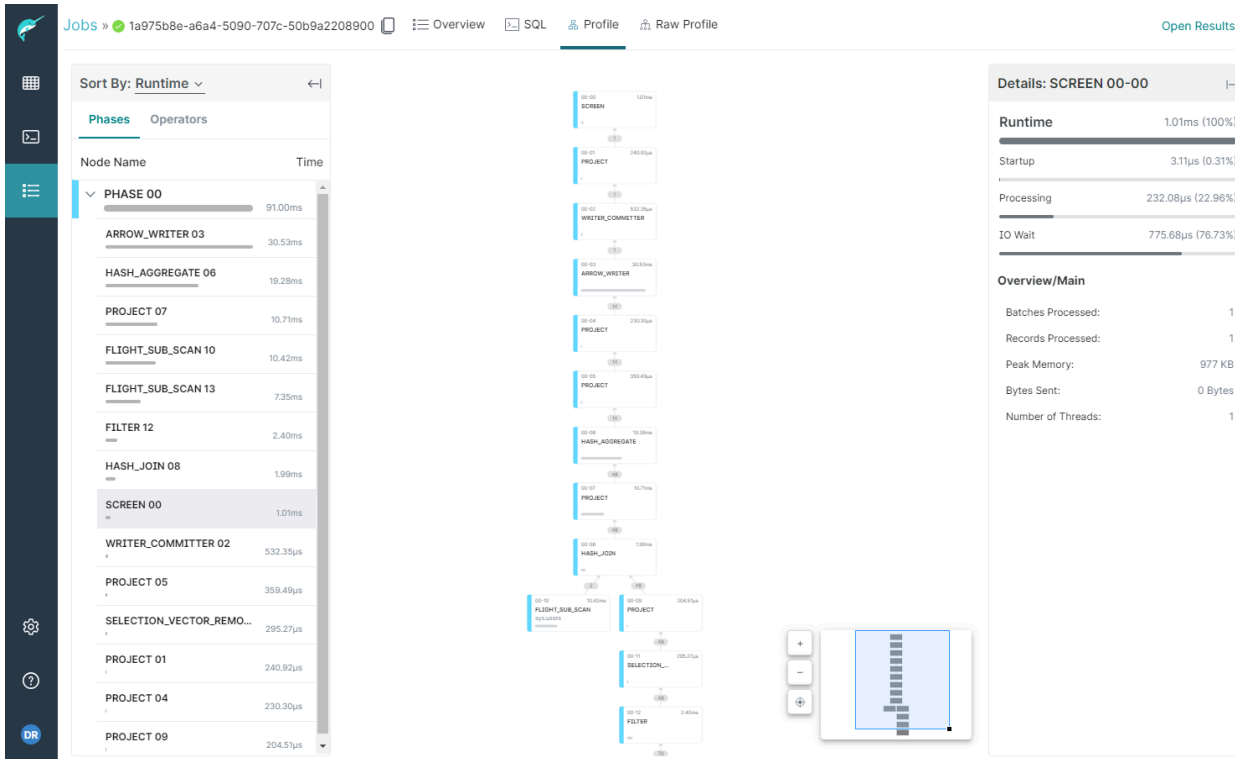
This is an interactive\draggable visual representation of the lineage of the submitted query. It shows all tables immediately accessed by the query, and if the query accessed a view it will show the lineage of that view all the way back to the views and physical tables that constituted the view in the lineage.

A dataset graph only appears if the query contains queried datasets (i.e. if the query was a SELECT statement against views or tables).

## Visual Profile Tab

The Visual Profile tab includes a visual representation of the [phases and operators](#) the planner has deduced are required to execute this query, along with some key data points related to

those phases and operators. More details on the information presented on this tab are available in the [Visual Profiles](#) section of this document.



## Raw Profile Tab

This tab contains the original form of the profile of a single specific query. Various further tabs on the raw profile let you drill into different aspects of it. To better understand what information is shared in the raw profile, it is important first to understand Dremio's query processing architecture. After understanding the architecture, more details are available in the [Raw Profiles](#) section of this document.

**Raw Profile** ✕

---

**Query and Planning**

Query   Visualized Plan   Planning   Acceleration

```

SELECT u.user_id, p.grantee_id, p.grantee_type, p.object_id, listagg(p.privilege, ',') AS privileges
FROM sys.privileges p INNER JOIN sys.users u
  ON p.grantee_id = u.user_name
WHERE p.object_type = 'SCRIPT'
GROUP BY u.user_id, p.grantee_id, p.grantee_type, p.object_id

```

**Job Summary**

State: COMPLETED  
Coordinator: ip-172-31-60-122.ec2.internal  
Threads: 1  
Command Pool Wait: 0ms  
Total Query Time: 241ms  
Considered Reflections: 0  
Matched Reflections: 0  
Chosed Reflections: 0

**State Durations**

Pending: 0ms  
Metadata Retrieval: 36ms  
Planning: 80ms  
Engine Start: 0ms  
Queued: 15ms  
Execution Planning: 2ms  
Starting: 3ms  
Running: 105ms

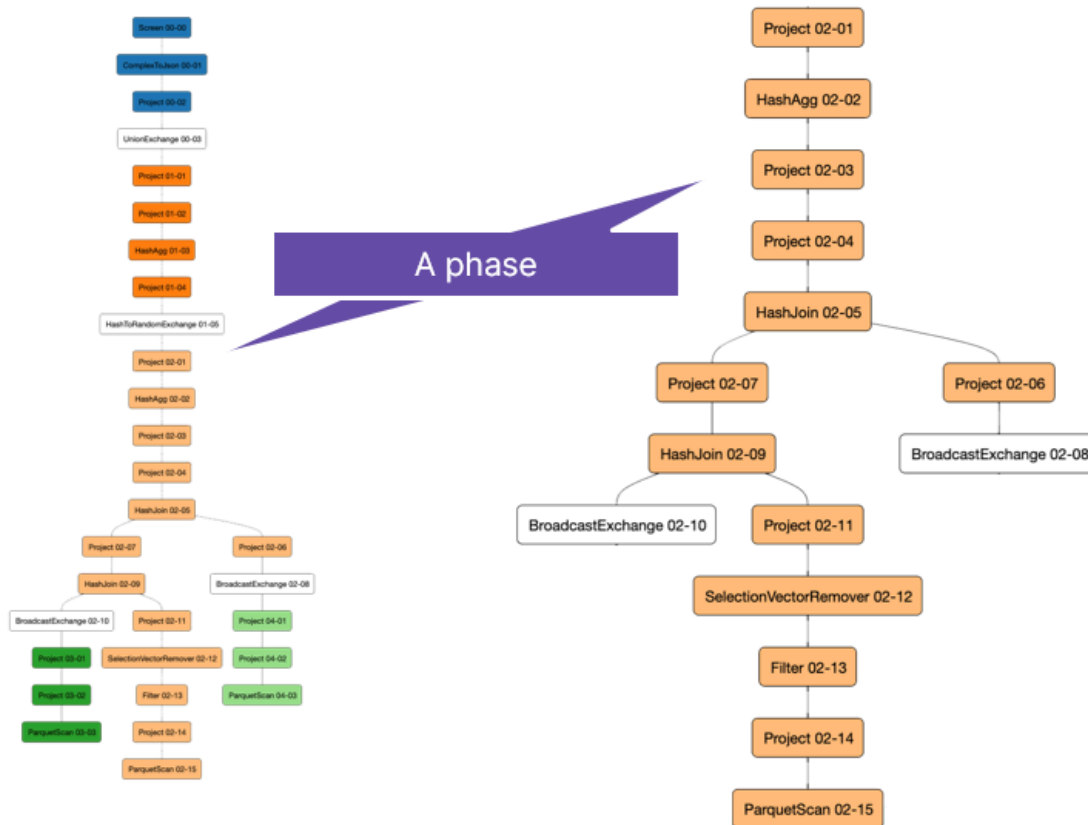
**Threads**

## Query Processing Architecture

The statistics shared in the [job details](#) are derived from data stored in the job profile. Since a profile goes into much more depth than what is summarized in the job details, it is essential first to understand three key concepts of query execution and how they are interrelated.

### Phases and Operators

Every query submitted to Dremio is broken down by the query planner into execution phases, also known as Major Fragments. Each phase represents one part of the execution of the entire query and each phase will typically, but not always, have multiple stages, known as operators, within it. In the Dremio User Interface, you can visualize the query plan as a tree-like structure and see that each phase is represented as a different color, so boxes with the same color are operators in the same phase, as shown in the left-hand side of the diagram below:

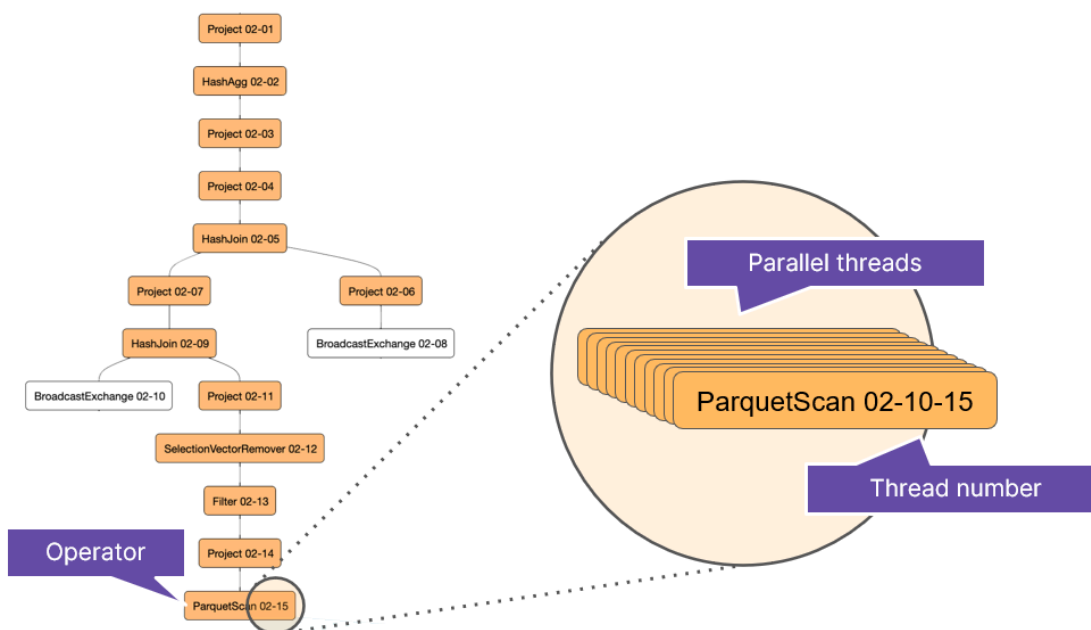


When determining what is happening in a query, we start at the bottom in the leaf nodes and work our way through the phases, which can be processed in parallel, until we reach the top of the tree, which is always called Phase 0, as depicted in the left-hand diagram above. So, using this query as an example, we can see at the lowest level, we are reading data from Parquet files, represented by the ParquetScan operators. Then the query works its way up through a series of filters, projects, joins, aggregations and transformations. Finally, at the very top, we return the data to the client who issued the query. Where the data is returned is called the Screen operator.

Similarly, in any one phase, we read the operators in the phase from bottom to top. The tree structure on the right side of the diagram above is a zoomed-in version of the light-orange phase from the left-hand tree. We can see that this particular phase starts with a ParquetScan operator, it travels through a couple of projects, a filter, and then there's a HashJoin, which is where the data from this phase is being joined with data coming from a different phase. Then there's another HashJoin a bit further up, which joins the data in this phase to another table in another different phase. Towards the top the data is being aggregated and then there's a final Project which will send the data into the next phase.

## Threads

Within phases, at the individual Operator level, we have multiple single-threaded instances that parallelize the work of that phase, also known as Minor Fragments. In the zoomed-in depiction of the ParquetScan operator below, we can see how several parallel threads will process the ParquetScan. Each thread processes a different set of data through the same series of operators, which is exchanged from one phase to another. When looking at a Dremio profile, when you see three sequential numbers like in the zoomed-in ParquetScan, the middle number represents the thread number with the highest number indicating there are (highest number - 1) threads in the phase.



These threads will typically run across all nodes in the engine that executes the query, so in a scale-out MPP system like Dremio, we will have many Executor nodes in an engine, and each Executor node is running the Dremio process, which is running on a number of physical cores and can therefore run a number of threads within an operator on each physical node.

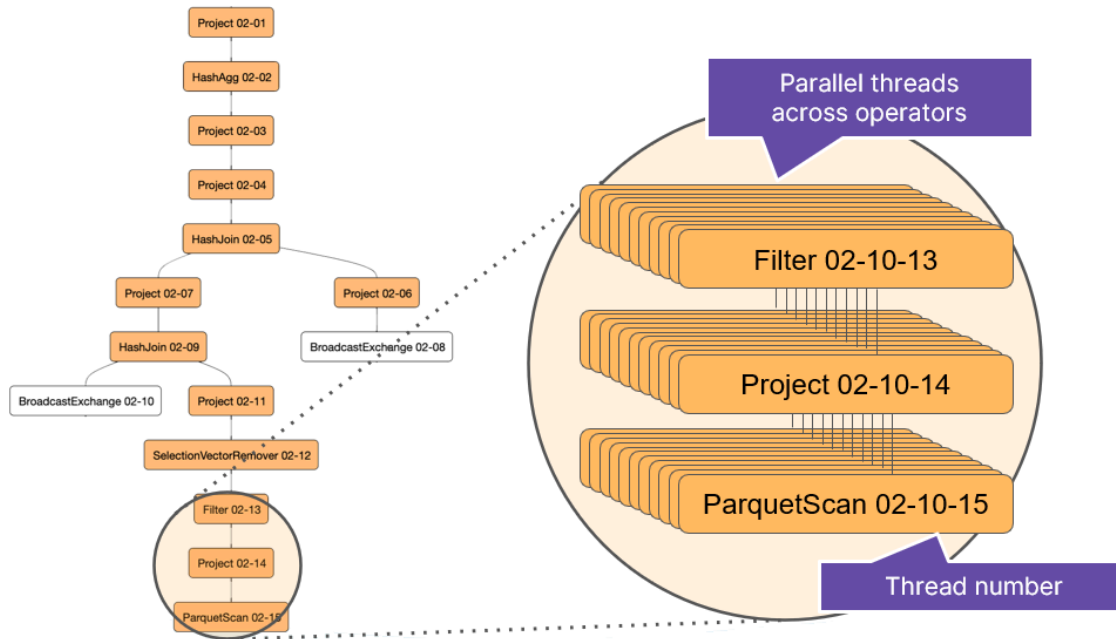
For very wide parallelization, it's expected that Dremio could have hundreds of threads working on a particular query phase.

## Threads Cross Operators

The processing threads in Dremio cross operators in a phase, which means that the number of threads remains consistent for every operator in a phase, as depicted in the zoomed-in set of operators on the right side of the diagram below. This also means that a single thread will cross each operator, so thread number 10 in this diagram will start by performing a



ParquetScan, then the same thread number 10 will work on the Project operator, then on its part of the Filter etc.. until it gets to the Project at the top of the tree.



We mentioned in the previous section that the middle number in the detailed profile represents the thread number. The left-hand number represents the phase number, so we’re focussing on phase 02 in the diagram above. The right-hand number is the operator number within the phase.

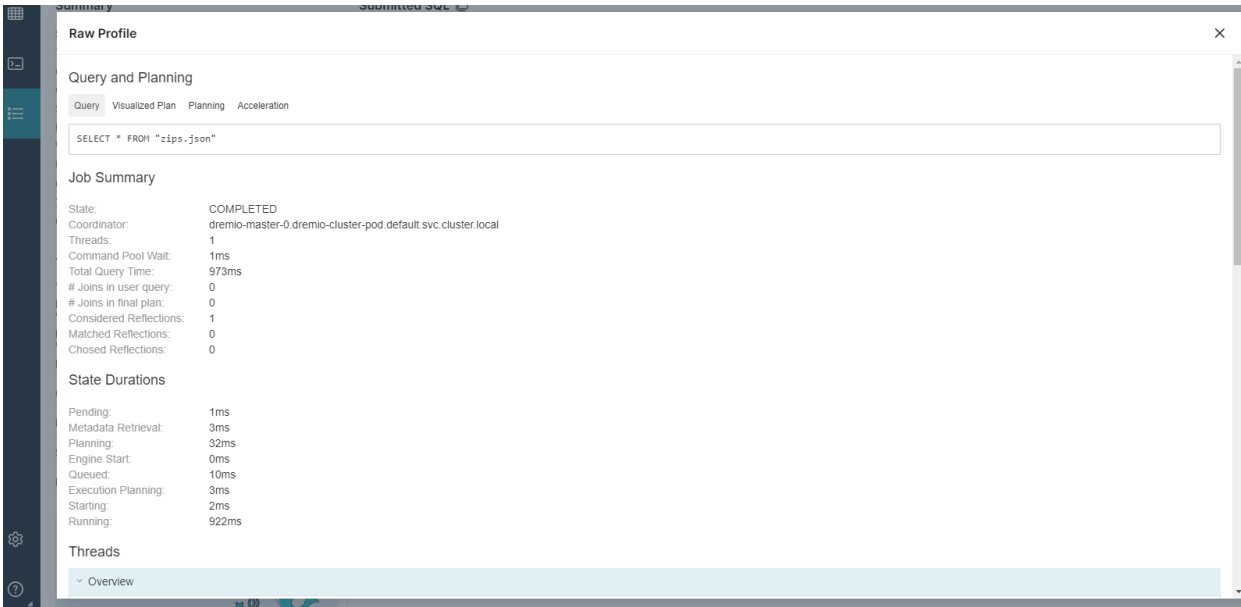
Earlier, it was stated that to determine what happens within a phase, we read the phase from bottom to top; it must also be noted that the operator numbers decrease to 01 as you go up the chain, which indicates we have 15 operators in the phase of the query shown above.

## Raw Profiles

The raw profile provides a window into the deep technical details of a query's planning and execution. When selected, the raw profile appears as a dialog on top of the main Dremio User Interface. It has four tabs and sometimes five if the query produces an error. Each tab contains static information that can be searched and traversed to determine where the bottlenecks might be in a query. The contents of each tab are described in more depth below.

### Query Tab

The query tab is the default tab displayed when opening the raw profile. At the top, it shows the selected query statement, followed by some overall job metrics and how long the query spent in each phase of execution.



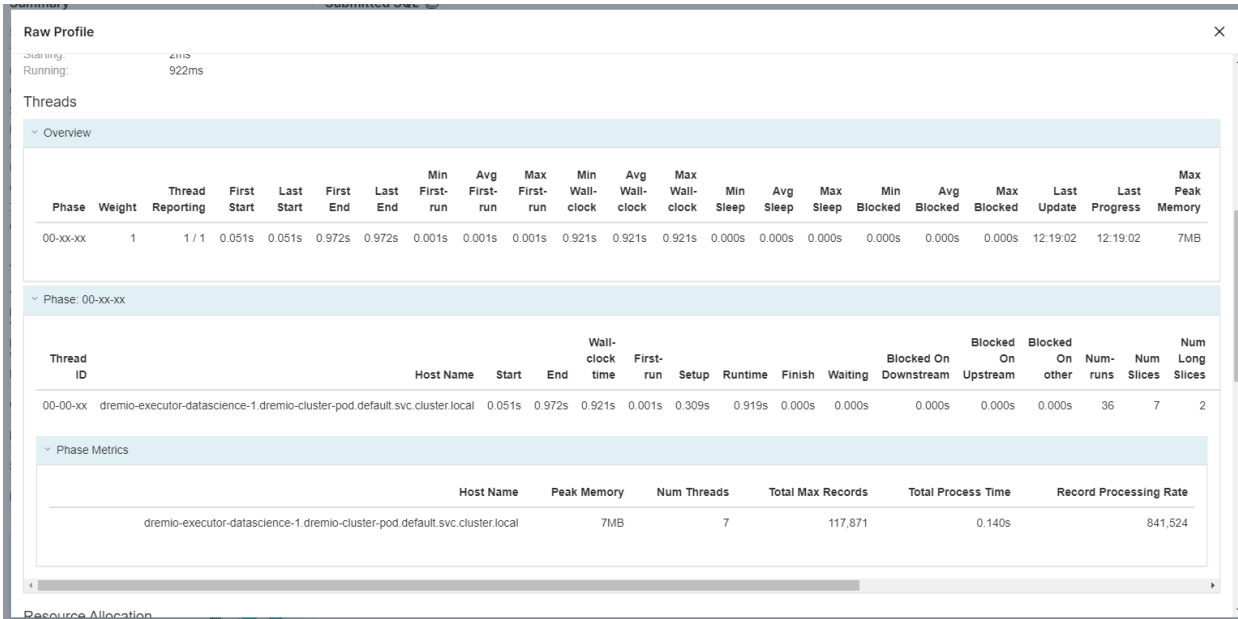
## State Durations

The following table describes each state that a query goes through, as shown in the State Durations section of the Query tab in a raw profile, and what it could mean if any of those states had a particularly long duration.

State	Description	A high value here could mean
Pending	Time spent waiting in the <b>command pool</b> before a job can begin	<ul style="list-style-type: none"> <li>High concurrency on the <i>coordinator</i></li> </ul>
Metadata Retrieval	Time spent in SQL parsing, permission checks, and inline metadata refresh	<ul style="list-style-type: none"> <li>Bad source</li> <li>Inline metadata refresh</li> </ul>
Planning	Time spent in physical and logical planning, including partition pruning and reflection matching	<ul style="list-style-type: none"> <li>Complex query</li> <li>Lots of partitions/reflections</li> </ul>
Engine Start	Time spent waiting for an engine to start (elastic engines only)	<ul style="list-style-type: none"> <li>Engine was idle &amp; needed to be started</li> <li>AWS having high deploy times with EC2</li> </ul>
Queued	Time spent waiting until a WLM/engine slot becomes available for the query to run	<ul style="list-style-type: none"> <li>Too many concurrent queries for the queue/engine</li> <li>Cancelled queries that haven't cleaned up yet</li> </ul>
Execution Planning	Time spent in parallelizing the fragments across nodes in the cluster	<ul style="list-style-type: none"> <li>Lots of splits in kvstore</li> <li>High concurrency on the coordinator</li> </ul>
Starting	Time spent in transmitting the fragments to the executors	<ul style="list-style-type: none"> <li>One or more executors having network issues or slow due to GC issue</li> </ul>
Running	Time spent in the query execution on executors	<ul style="list-style-type: none"> <li>See Threads and Operators sections</li> </ul>

## Threads - Overview and Phase Details

Below the State Durations section are details of each thread. The Threads Overview section shows the aggregates – minimum, average, maximum, firsts and lasts – across all the fragments (threads) in each Phase. The Phase Details can be expanded (as shown below) to show these metrics for individual fragments (threads).



The following table describes the columns you see in the Threads section and what it could mean if those columns had relatively high values.

Column	Description	A high value here could mean
First Start / Last Start	Delta between the query submission time in the coordinator and the creation of the fragment on the executor.	<ul style="list-style-type: none"> <li>Large value for all fragments means that some earlier state (before RUNNING) had a longer duration. Look at the section “State Durations” again.</li> <li>Big variation for some fragments means that some executors received the fragment late. It could be a network issue or a long GC on the executor.</li> </ul>
First End / Last End	Completion time for the fragment	
First-run (Min/Avg/Max)	Delta between the creation of the fragment and the first time it ran (for setup)	<ul style="list-style-type: none"> <li>Large value could mean a scheduler issue or too much CPU contention.</li> </ul>
Sleep (Min/Avg/Max) - Overview Waiting - Phase detail	Duration where the fragment was runnable but could not be scheduled on a <b>slicing thread</b> .	<ul style="list-style-type: none"> <li>High concurrency or resource contention on the executors</li> </ul>

Column	Description	A high value here could mean
Blocked (Min/Avg/Max)	Duration during which the fragment was blocked on some upstream or downstream fragment.	<ul style="list-style-type: none"> <li>If blocked on downstream for phase 0 then this typically means the client tool is too slow ingesting the data from Dremio.</li> </ul>
First Start / Last Start	Delta between the query submission time in the coordinator and the creation of the fragment on the executor.	<ul style="list-style-type: none"> <li>Large value for all fragments means that some earlier state (before RUNNING) had a longer duration. Look at the section "State Durations" again.</li> <li>Big variation for some fragments means that some executors received the fragment late. It could be a network issue or a long GC on the executor.</li> </ul>

## Operators - Overview

At the bottom of the Query tab is an overview of each operator and a set of per-operator metrics. In most cases, the overview section should be sufficient to explain or deduce what impacts the query performance in terms of which operator has the most significant effect. You can then expand the operator as necessary to see more details; bear in mind each operator has its own list of metrics.

SqlOperatorImpl ID	Type	Min Setup Time	Avg Setup Time	Max Setup Time	Min Process Time	Avg Process Time	Max Process Time	Min Wait Time	Avg Wait Time	Max Wait Time	Avg Peak Memory	Max Peak Memory
00-xx-00	SCREEN	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	976KB	976KB
00-xx-01	PROJECT	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	976KB	976KB
00-xx-02	WRITER_COMMITTER	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	976KB	976KB
00-xx-03	ARROW_WRITER	0.000s	0.000s	0.000s	0.001s	0.001s	0.001s	0.003s	0.003s	0.003s	976KB	976KB
00-xx-04	PROJECT	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	976KB	976KB
00-xx-05	PROJECT	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	976KB	976KB
00-xx-06	JSON_SUB_SCAN	0.001s	0.001s	0.001s	0.139s	0.139s	0.139s	0.773s	0.773s	0.773s	976KB	976KB

The following table describes the columns in the Operators Overview section and what it could mean if those columns had relatively high values.

Column	Description	A high value here could mean
Setup time	Time taken to setup the operator for the first time	<ul style="list-style-type: none"> <li>Complex expression causing time spent in code generation (Gandiva, NLJ for Java)</li> <li>First query for a plugin, requiring some initialization on the executor</li> </ul>
Process time	CPU processing time for the operator	In most cases, this is expected - try to optimize the query, use reflections, ..
Wait time	Time spent in reading/writing data to the source/filesystem	<ul style="list-style-type: none"> <li>Slow source</li> <li>Slow local disks on executors</li> <li>Async reads switched off or not supported for the source</li> </ul>

## Resource Allocation

Resource Allocation is another section on the Query tab. It details why or how the query was routed to a particular queue.

Resource Allocation

▼ Overview

Queue Name: Low Cost User Queries  
 Queue Id: 8da10584-6f3a-4562-83b3-91648ffa4f5c  
 Rule Name: Low Cost User Queries  
 Rule Content: query\_cost() < 30000000  
 Rule Action: PLACE  
 Query Cost: 1191658  
 Query Type: UI Run

## Nodes

The Nodes section on the Query tab summarises the hostname of the executors that ran one or more fragments for the query and what peak memory was used on that host.

Nodes

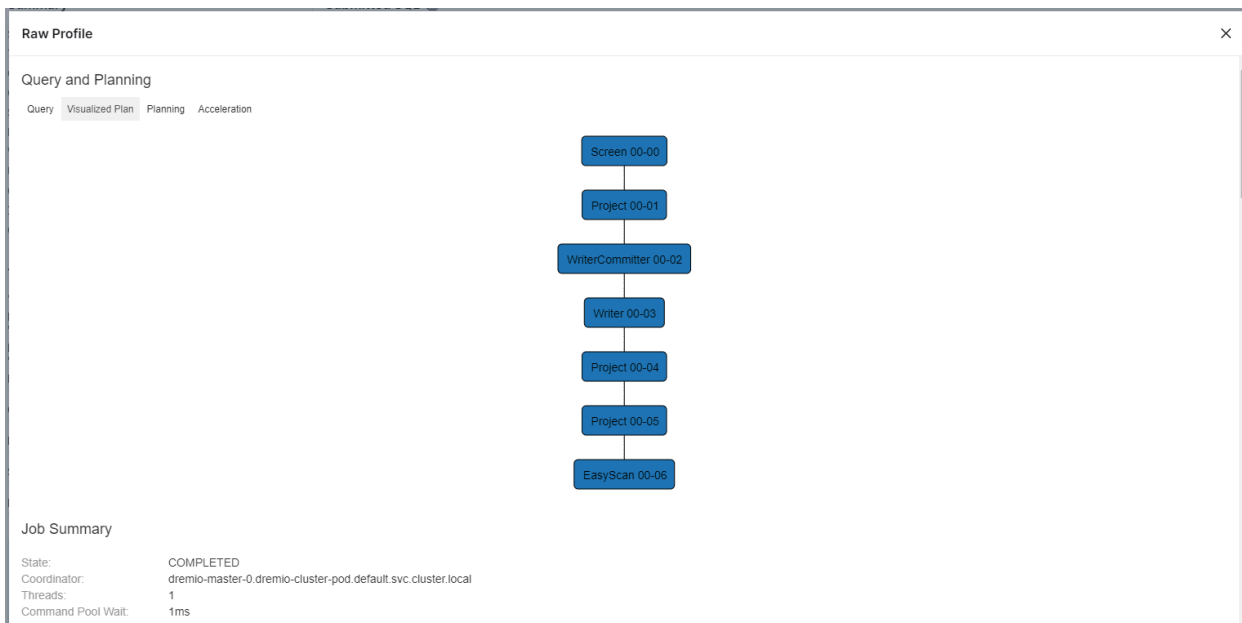
▼ Overview

Host Name	Resource Waiting Time	Peak Memory
dremio-executor-datascience-1.dremio-cluster-pod.default.svc.cluster.local	0.000s	7MB

## Visualized Plan Tab

The Visualized plan was described in this document's [Phases and Operators](#) section. It is an often colourful graphic that visually illustrates how a query will be executed using a tree-like structure. It is read from the bottom up and the different colours represent the execution phases.

Below the visual diagram, you will see the same textual information sections available on the [Query tab](#).



## Planning Tab

The planning tab details how long Dremio spends in each part of the planning phase of a query. It will show textually how it breaks the query down into its component relational pieces and eventually shows the Final Physical Transformation, which details the phases of execution and the operators within the phases it has settled upon as the most optimal plan. It provides statistics about the actual cost of the query operations in terms of memory, input/output and CPU processing.

You can use this tab to identify which operations consumed the majority of resources during a query so you can address cost-intensive operations. In particular, the following information from the Planning tab is useful:

- Non Default Options – See if non-default parameters\support keys are used, which may affect the query performance.
- Metadata Cache Hits and Misses with durations

- Final Physical Transformation – Look for pushdown queries for RDBMS, MongoDB, or Elasticsearch, filter pushdowns or partition pruning for parquet, and view usage of stripes for ORC.
- Compare the estimated row count versus the actual scan, join, or aggregate result.
- Row Count – See if row count (versus rows) is used. Row count can cause an expensive broadcast.
- Build – When performing joins, there will be a Build side of a query and a Probe side. The build side of the query is brought into memory, so you typically want the smaller side of the join to be on the Build side.

## Acceleration Tab

The Acceleration tab helps us understand the optimizer's thought process regarding whether to use a reflection to accelerate the query and indeed which reflection to use, if any.

### Reflection Outcome

This section tells us at a glance whether the query was or was not accelerated and shows which reflections were considered, matched and chosen by the optimizer to accelerate the query.

**Raw Profile**

---

Query and Planning

Query   Visualized Plan   Planning   **Acceleration**

Reflection Outcome

Query was accelerated

- test\_taxis ("@alexc")
  - Aggregation Reflection (Type: agg, Reflection Id: fe3d297d-3c57-47b3-936b-79c015067b4a): considered, not matched.
- taxi-w ("@dave.anderson@dremio.com")
  - Raw Reflection (Type: raw, Reflection Id: 60e96e95-bed7-4de6-99ef-12c31fcc9b2): considered, matched, not chosen.
- NYC Trips (Business.Transportation)
  - Aggregation Reflection\_new (Type: agg, Reflection Id: 643a5b53-0f8e-4192-a3cb-30b9519df215): considered, matched, chosen.
  - Aggregation Reflection (Type: agg, Reflection Id: c8fa4900-54ca-456c-a65f-443e7904b1a6): considered, matched, not chosen.

The following considerations determine the acceleration process:

- Considered, Matched, Chosen – The query is accelerated.
- Considered, Matched, Not Chosen – The query is not accelerated because either a costing issue or an exception during substitution occurred.
- Considered, Not Matched, Not Chosen – The query is not accelerated because the reflection does not have the data to accelerate.

## Reflection Details

This section provides more detailed information on why the optimizer decided to use or not use a reflection. This could be because not all fields required by the query were captured in the reflection, for example, or maybe there was an alternative reflection that was deemed more cost-efficient. The second reflection was chosen in the example below because it was deemed the best cost of the matched reflections.

### Reflection Details

#### Reflection Definition: Aggregation Reflection

Matched: 0, Chosen: 0, Match Latency: 0 ms

Reflection Id: fe3d297d-3c57-47b3-936b-79c015067b4a, Materialization Id: 34df9a7f-1b30-4cc6-a642-9543fd582c6

Expiration: 3023-03-05T14:35:58Z

Dataset: "@alexc".test\_taxis

Age: 61 days 2 hours 10 minutes 2 seconds

Dimensions: pickup\_date,

Measures:

- passenger\_count ( SUM, COUNT, )

#### Canonicalized Reflection Plans:

Matching Hints:

Replacement Plans:

Best Cost Replacement Plan:

#### Reflection Definition: Aggregation Reflection\_new

Matched: 3, Chosen: 1, Match Latency: 0 ms

Reflection Id: 643a5b53-0f8e-4192-a3cb-30b9519df215, Materialization Id: 2f8b3ac6-a2cd-41f9-ae22-24c1134a27e5

Expiration: 3022-09-16T14:43:48Z

Dataset: Business.Transportation."NYC Trips"

Age: 231 days 2 hours 2 minutes 12 seconds

Dimensions: vendor\_id, pickup\_date,

Measures:

- passenger\_count ( SUM, COUNT, )

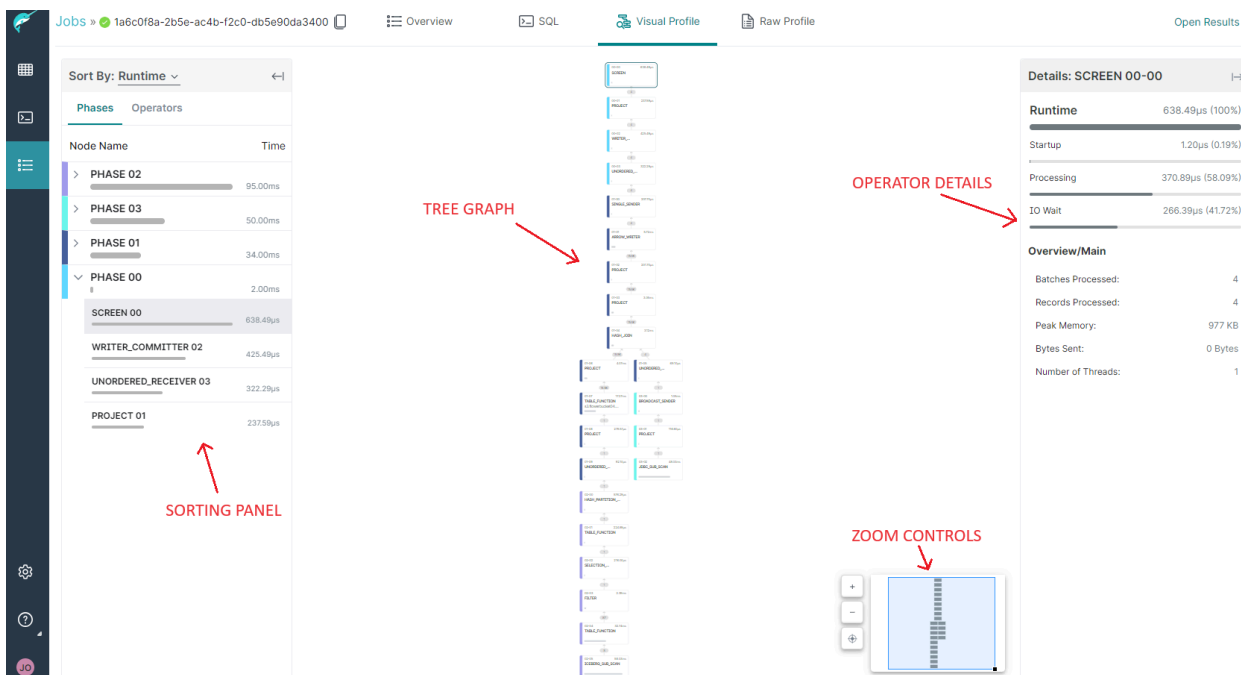
## Error Tab

This tab will only appear if there is an error whilst running the query. You will see more information about why the query failed. For example, it could get cancelled for exceeding memory limits set by the administrator or one or more nodes lost connectivity during query execution. The information in the Error tab should be sufficient for you to determine why it occurred and give you an idea of how to fix it.

## Visual Profiles

The visual profile provides a more easy-to-understand and somewhat dynamic view of what is happening in the profile by allowing you to click around and sort it by various telling indicators.





The main components of a visual profile are described below.

## Sorting Panel

The sorting panel is helpful because it allows you to see almost instantly which phases, and more importantly which operators, are the most expensive in terms of execution time, memory consumption, and number of rows processed. Whilst these metrics alone are not indicators of a problem in the query, they are often an excellent first step to understanding where it might be worthwhile looking in a query to see if something more sinister is happening. By identifying some candidate operators, you can then select the individual operator in the [tree graph](#) to see more details of the operator in the operator details.

Identifying these operators can lead us to determine operators with the highest processing and wait times, which can be caused by:

- Memory issues, where specific parts of SQL execution require more memory than what is available, usually the result of a high cardinality aggregation or join operation; spilling the join to disk can alleviate this, but will cause a slowdown in the query
- Slow external source, which is the query primarily waiting for an external source to return data

## Tree Graph

The tree graph is structured much the same as described in the [Phases and Operators](#) section of this document and the flow is essentially from bottom to top. In the top left of each operator, you see its phase and operator numbers as two digits each, e.g. 01-04 would be phase 01,

operator 04. You also see the operator's name in the middle of each operator and a key metric in the top right, which matches whichever metric you have chosen to sort by in the [Sorting Panel](#).

Clicking on any operator in the tree changes the data displayed in the [Operator Details](#) panel on the right.

Between each operator, as shown below, you can see a circled number with a grey background (e.g. 9.7K); this gives you a visual indication of the number of rows sent between one operator and the next. Understanding the data volume handled by the operator, both entering and exiting it, helps you to:

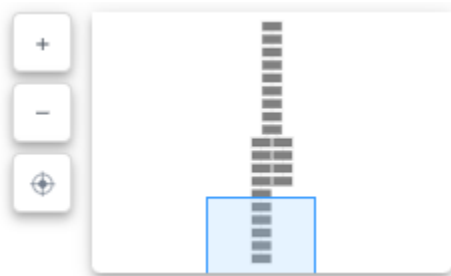
- Identify exploding joins, where the result of a join operation significantly increased the data volume to be processed beyond the initial size of the tables selected
- Spot insufficient pruning for very large datasets, where too much data is returned from the scan operations to be processed, and additional or more effective filters are needed to prune data before processing, including partition, row group pruning, and pushdown filtering
- Spot insufficient predicate pushdown, where a query returned a large amount of data from a relational database source due to insufficient pushdown applied



If you see the lightning bolt icon ( ⚡ ) it means a reflection was used to serve the data into that operator.

## Zoom Controls

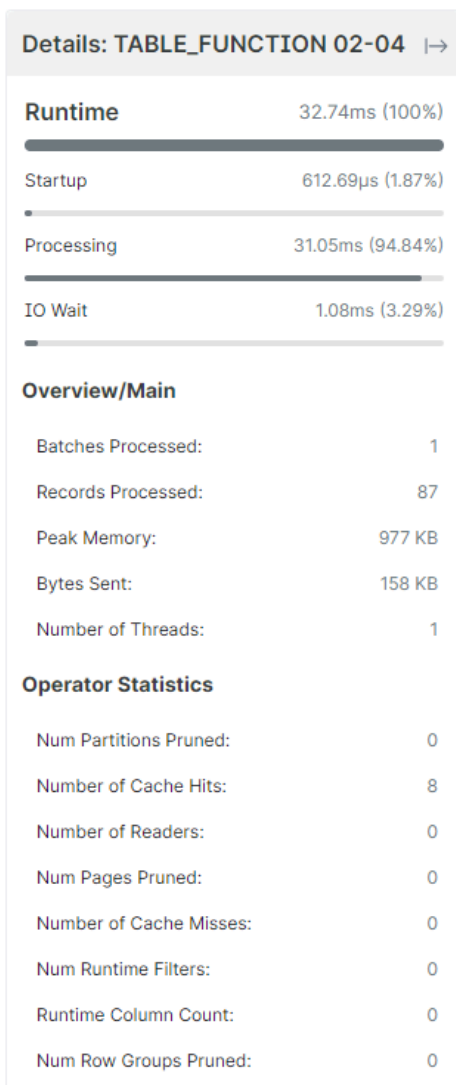
The zoom controls are helpful when you have a large tree graph that doesn't fully fit into the area provided to display it; in that case, you can use the zoom control to help quickly navigate around the entirety of the tree graph to help focus onto the specific area of interest.



## Operator Details

The Operator Details panel shows the details and statistics about the selected operator. This information could help to identify the root cause of performance issues, including

- High CPU wait times that indicate issues with infrastructure capacity or workload management
- Slow response times from BI tools
- Slow response times from external data sources
- Sub-optimal data formats or file structures in your data lake
- Sub-optimal cache utilization



## Summary

In this document we have considered what statistics are presented in the Dremio User Interface for every query that gets processed by Dremio. We started by investigating the information on the Job Details Overview page, which helps give us the highest-level overview of how Dremio spent its time planning and executing the query. We can click through tabs from the Job Details page that give us a different, often deeper perspective on what happened during the query execution. We saw how we can drill into the finer details of the query visually using the Visual Profile and also how we can get into the lowest level of statistics using the Raw Profile. Between these two profile views, it is almost certain that we can ascertain where the bottlenecks lie in a query and thus we glean some pointers on how to resolve them.

To understand the details in the profiles, it is also essential to understand the relationships between phases, operators and threads, which is why the document also explains the Dremio query processing architecture.