

Query Analyzer User Guide

June 1st, 2022

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| 1.0 | 02/24/2020 | Mark Hoerth | Initial release |
| 1.1 | 03/19/2020 | Deane Harding | Updated document format. |
| 1.2 | 03/27/2020 | Deane Harding | Added further VDS definitions |
| 1.3 | 05/22/2020 | Deane Harding | Minor update to queries.json scrubbing process. |
| 1.4 | 07/21/2020 | Deane Harding | Addition of functionality to retrieve error messages related to failed queries |
| 1.5 | 10/08/2020 | Deane Harding | Removed reliance on Java. Added more analysis VDSs. |
| 1.6 | 03/03/2022 | Deane Harding | Introduced splitting of error messages into 16k chunks |
| 1.7 | 5/18/2022 | Maz Mohammadi | Added new VDSs for Ntile and hints in the SelectQueryData |
| 2.0 | 6/1/2022 | Maz Mohammadi | Reorganizing the Query Analyzer. All VDSs in the application folder are specified as examples in this doc.  Column renaming and casting is now done in the Preparation layer |
| 2.1 | 10/4/2022 | Maz Mohammadi | Adding new VDSs, merging the work back fieldeng |
| 2.5 | 11/18/2022 | Maz Mohammadi | Adding new VDSs for Reflection management |
| 3.0 | 9/11/2023 | Maz Mohammadi | Adding MDRefresh and ReflectionRefresh DailyRreportByOutcomePivoted VDS |

Related Documents

|  |  |
| --- | --- |
| **Name** | **Version** |
| semantic\_layer\_best\_practices.pdf | 1.0, September 2019 |

Supported Versions

|  |  |
| --- | --- |
| **Name** | **Version** |
| Dremio | 3.2.8 or later |
| Python | 3.5+ |

Table of Contents

[Introduction 4](#_Toc145916803)

[Purpose 4](#_Toc145916804)

[Setup and Configuration 5](#_Toc145916805)

[Install Dremio Query Analyzer 5](#_Toc145916806)

[Configure gather\_queries.sh 7](#_Toc145916807)

[Execution 8](#_Toc145916808)

[Query Analysis 9](#_Toc145916809)

[Create the PDSs 9](#_Toc145916810)

[Create the VDSs 11](#_Toc145916811)

[Automated VDS creation 11](#_Toc145916812)

[Manual VDS creation 12](#_Toc145916813)

[Analyze Results 14](#_Toc145916814)

[Examples 16](#_Toc145916815)

[Getting Support 28](#_Toc145916816)

# 

# 

# Introduction

## Purpose

Dremio provides a log of completed queries called queries.json that provides a rich and extensive view of queries on the system. The queries.json logs can be queried by Dremio itself or another tool for monitoring and analytics about the queries on the system. Such analytics can reveal new insight into query acceleration opportunities, most frequently run queries, or highest cost queries.

Dremio Query Analyzer is a tool that automates the retrieval, preparation and copying of current and historic queries.json files into S3/ADLS/HDFS file storage ready for analysis in Dremio. The tool also provides a set of VDS definitions that can be loaded into Dremio in order to enable immediate analysis of the query data.

# Setup and Configuration

## Install Dremio Query Analyzer

Dremio Query Analyzer is delivered as a compressed file called dremio-query-analyzer.tar.gz.

* Copy the file to the Dremio Coordinator node
* Unpack the file into a location of your choice, here we choose the dremio user’s home directory

|  |
| --- |
| tar xvzf dremio-query-analyzer.tar.gz -C /home/dremio |

* The tool also requires python in order to successfully execute. Therefore, ensure python is present on the Dremio Coordinator; first issue the following command to see if python is already installed.

|  |
| --- |
| Python -V |

* If you are not presented with details of the currently installed python version, then you will need to install Python. Version 3.5 or above is recommended. The following site provides an explanation if required:

<https://docs.python-guide.org/starting/installation/>

* **(Optional)** If you are going to be using dremio-query-analyzer to copy the test results files to Amazon S3, then you may need to install and configure the AWS CLI on the Dremio Coordinator. First issue the following command to see if the AWS CLI is already installed:

|  |
| --- |
| aws –version |

If you are not presented with details of the currently installed AWS CLI version, then use the following links as guidance.

* + To install the AWS CLI, follow these instructions:

<https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html>

* + To configure the AWS CLI, follow these instructions:

<https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html#cli-quick-configuration>

* **(Optional)** If you are going to be using dremio-query-analyzer to copy the test results files to Azure Storage, then you may need to install and configure the Azure CLI on the Dremio Coordinator. First issue the following command to see if the Azure CLI is already installed:

|  |
| --- |
| az –version |

If you are not presented with details of the currently installed Azure CLI version, then use the following links as guidance:

<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>

* **(Optional)** If you are going to be using dremio-query-analyzer to copy the test results files to Hadoop Storage, then you need to make sure the following command successfully executes on the Dremio Coordinator:

|  |
| --- |
| hdfs version |

This dremio-query-analyzer package includes scripts for the one-time or continuous preparation of queries.json files from Dremio.

The queries.json log is located on the cluster coordinator, at the location /var/log/dremio and /var/log/dremio/archive on most clusters. See the Dremio documentation for additional detail.

Dremio-query-analyzer is responsible for cleaning up (“scrubbing”) the queries.json file prior to its consumption in Dremio, ensuring rules regarding lengths of data fields when they are imported into Dremio are adhered to. The dremio-query-analyzer tool is also responsible for copying the scrubbed queries.json files into a user-defined cloud storage container (S3, ADLS, HDFS) that is accessible to Dremio in order to analyze the data, the script assumes that there will be two folders available in the designated storage location to store the scrubbed queries.json files, one called “results” and another called “chunks”.

The main scripts of this package are defined below:

|  |  |
| --- | --- |
| **Script** | **Description** |
| gather\_queries.sh | This is the primary driver script for the dremio-query-analyzer. The script can be optionally installed in cron to regularly copy queries.json from the Dremio cluster to a location on S3\HDFS\ADLS. Requires customization with your configuration. Once a day is usually suitable for most users. |
| scrub-queries-json.py | Scrubs a set of queries.json files in a specified directory to truncate the queryText field into a number of equally sized chunks. The first chunk is written back into a header.queries\*.json file, all other chunks are written into a chunks.queries\*.json file. The scrubbed output files are written to a specified output directory. This is primarily because Dremio VARCHAR fields can be no greater than 32000 characters in length. |
| get-error-messages.py | Reads each queries.json file, identifies queries that failed and for each failed queries makes an API call to Dremio to retrieve a more descriptive message for why the query failed. Records get written to a new file called errormessages.queries.\*.json. |
| refresh-pds.py | Refreshes metadata for a specified PDS in Dremio. Used in gather\_queries.sh to refresh the metadata for results, chunks and errormessages PDSs in Dremio after new files have been uploaded to the data lake storage. |

*Queries.json analysis scripts provided in the dremio-query-analyzer package from Dremio Professional Services*

## Configure gather\_queries.sh

The driving script of the dremio-query-analyzer, called gather\_queries.sh requires the following variables inside the script to be specified correctly prior to use:

|  |  |
| --- | --- |
| **Variable** | **Description** |
| DREMIO\_LOG\_DIR | The directory on the Dremio Coordinator where all the Dremio log files are stored. Default value is “/var/log/dremio” |
| dremio\_url | The base URL for accessing the Dremio REST API, e.g. <http://localhost:9047>. Used when gather\_queries.sh calls get-error-messages.py and refresh-pds.py |
| user\_name | Name of an administrative user in Dremio that can access the REST API. Used when gather\_queries.sh calls get-error-messages.py |
| pwd | Password associated with an administrative user in Dremio that can access the REST API. Used when gather\_queries.sh calls get-error-messages.py. The password can be entered directly or stored in a file only accessible by the user running gather\_queries.sh. |

*Variables in gather\_queries.sh configured prior to use*

# Execution

At execution time, gather\_queries.sh, requires the following three parameters as inputs on the command line:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| storage\_type | The type of storage where we want to write the scrubbed queries.json file to. Valid values are currently **s3**, **adls**, **hdfs**. If no valid value is supplied, then the scrubbed files will remain on the Dremio Coordinator and will need to be manually copied to a storage location. |
| storage\_path | The path on the storage to a “results” folder e.g. s3://mybucket |
| num\_archive\_days | The number of days of archived queries.json files to also scrub and copy into storage. If no value is supplied, then only queries.json files for the current day will be scrubbed and copied. |

*Runtime parameters in gather\_queries.sh*

To make the results available for analysis, execute gather\_queries.sh on the Dremio coordinator with the desired input parameter values for your chosen storage type, storage location and number of historical queries.json files that you wish to process. This script reads the desired queries.json and queries.YYYY-MM-DD.N.json.gz files and prepares them for analysis. e.g.:

|  |
| --- |
| cd /home/dremio/dremio-query-analyzer/scripts/  ./gather\_queries.sh s3 s3://mybucket/dremio 2 |

gather\_queries.sh copies the queries.json and queries.YYYY-MM-DD.N.json.gz files locally to the dremio-query-analyzer in /home/dremio/dremio-query-analyzer/scripts/dremio\_queries, it then generates one or more files called header.<filename>.json, chunks.<filename>.json, errorheader.<filename>.json and errorchunks.<filename>.json in the /home/dremio/dremio-query-analyzer/scripts/dremio\_queries/scrubbed/ directory.

The scrubbed files then get automatically copied into the desired cloud storage location. By default, the header.<filename>.json files will get copied into a results sub-directory, the chunks.<filename>.json files will get copied into a chunks sub-directory and the errorheader.<filename>.json files will get copied into an errormessages sub-directory and the errorchunks.<filename>.json files will get copied into an errorchunks sub-directory in the cloud storage.

If a desired cloud storage location is not specified or if the storage type is invalid then the files will remain in the /home/dremio/dremio-query-analyzer/scripts/dremio\_queries/scrubbed/ directory and they will require manually copying to some other storage location that will be accessible to Dremio as a data source.

# Query Analysis

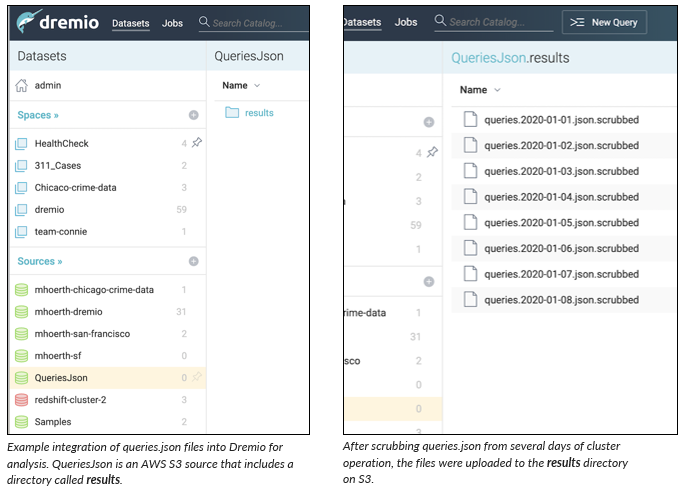
Before analysis begins there are several one-off tasks to perform in order to enable the analysis. Firstly, a data source and Physical Data Set (PDS) needs to be created in Dremio that will connect directly to the queries.json files. Secondly, a set of Virtual Data Sets (VDS) need to be created to query and make sense of the data in the PDS.

## Create the PDSs

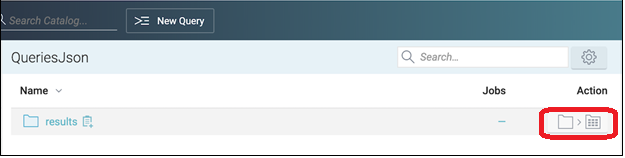
The assumption in this document is that the queries.json files have been copied into either S3, ADLS or HDFS storage, although equally they could have been placed on a user’s local filesystem and then uploaded into an individual user’s home space.

While not required, for the most straightforward integration with the pre-created scripts supplied in dremio-query-analyzer Dremio Professional Services recommends the queries.json files are placed into a folder called **results** in the chosen storage device.

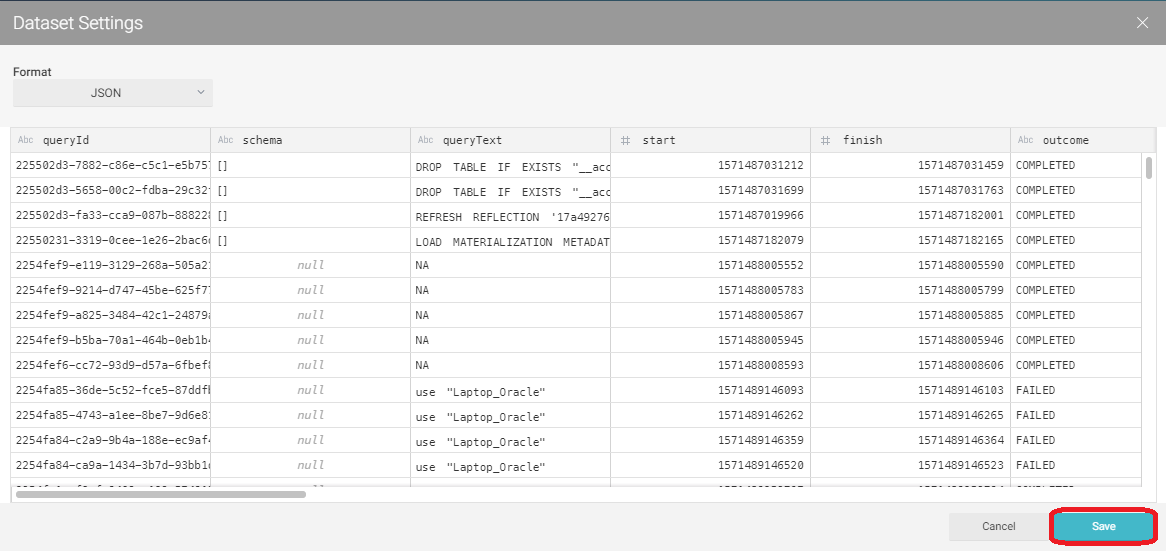
* Create a data source in Dremio applicable to the storage device where the queries.json files are stored, e.g. Amazon S3, Azure Storage, HDFS.
* For the chosen data source, enter the relevant connection credentials.
* Navigate to the Advanced Options panel of the data source and enter the Root Path to be the **parent path** of the results folder on the storage device.
* Set up the path of the data source to point to the directory immediately above the results directory. The following screenshots show an Amazon S3 data source called QueriesJson configured with access to the results folder in Dremio (left) and also the contents of the results folder (right):



* Use the Folder Format button to convert the results directory to a PDS



* In the resulting dialog, notice that Dremio automatically recognizes the format of the file inside this folder as JSON and has formatted the data table appropriately. Click Save.



* The same steps can be followed to convert the chunks, errormessages and errorchunks folders into PDSs in the same data source.

## Create the VDSs

Create a set of VDSs that will be used to analyze the data in the queries.json files. There are two approaches to this, automated VDS creation, or manual. Both options are described in the sections below. Dremio recommends the automated approach.

### Automated VDS creation

The primary file that is used to automatically create the relevant spaces, folders and VDSs in Dremio is called run\_vdscreator.sh. This file is located at /home/dremio/dremio-query-analyzer/scripts/vdscreator/run\_vdscreator.sh and has several parameters that need to be specified before VDSs can be created.

* Specify the following in run\_vdscreator.sh. All other parameters in the file must not be edited.

|  |  |
| --- | --- |
| **Variable** | **Description** |
| dremio\_host | The hostname or IP address of the Dremio Coordinator |
| dremio\_port | The HTTP port used by Dremio, default is 9047. |
| user\_name | The name of an administrative user in Dremio that can create spaces, folders and VDSs. |
| pwd | The password for the user user\_name above, read from local.pwd or embedded directly. This password will be used to log into Dremio via the REST API and issue calls to create the relevant objects. |

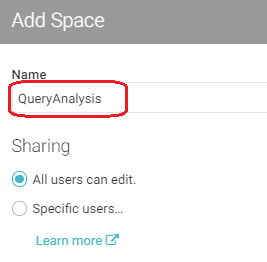
*Variables in run\_vdscreator.sh*

* Execute run\_vdscreator.sh to generate the required space, folders and VDSs in Dremio

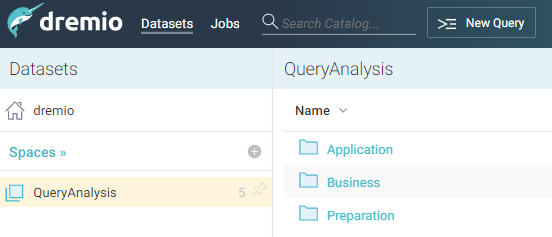
|  |
| --- |
| # Change user if necessary su – dremio cd /home/dremio/dremio-query-analyzer/scripts/vdscreator/ ./run\_vdscreator.sh |

### Manual VDS creation

* Create a new Space in Dremio called QueryAnalysis



* Inside the QueryAnalysis space create three new sub-folders: Application, Business, and Preparation



* On the Dremio Coordinator, navigate to the vdsdefinition directory:

|  |
| --- |
| cd /home/dremio/dremio-query-analyzer/vdsdefinition |

This directory contains the following sql files:

01\_QueryAnalysis.Preparation.chunks.sql

01\_QueryAnalysis.Preparation.errorchunks.sql

01\_QueryAnalysis.Preparation.errormessages.sql

01\_QueryAnalysis.Preparation.results.sql

02\_QueryAnalysis.Business.MDRefreshData.sql

02\_QueryAnalysis.Business.QueryErrorMessages.sql

02\_QueryAnalysis.Business.QueryTextChunks.sql

02\_QueryAnalysis.Business.ReflectionRefreshData.sql

02\_QueryAnalysis.Business.SelectQueryData.sql

04\_QueryAnalysis.Application.AcceleratedSelects.sql

09\_QueryAnalysis.Application.Top20ExecutionTimes.sql

15\_QueryAnalysis.Application.PctAccelerated.sql

19\_QueryAnalysis.Application.ActiveUsers.sql

20\_QueryAnalysis.Application.DailyRreportByOutcomePivoted.sql

29\_QueryAnalysis.Application.SummaryByOutcome.sql

30\_QueryAnalysis.Application.Range\_EnQueuedTime.sql

30\_QueryAnalysis.Application.SummaryQueryExecTimeByQueue.sql

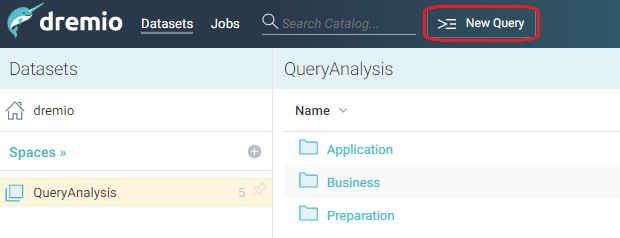
31\_QueryAnalysis.Application.Top20DataSetsQueried.sql

32\_QueryAnalysis.Application.NtileQueryCost.sql

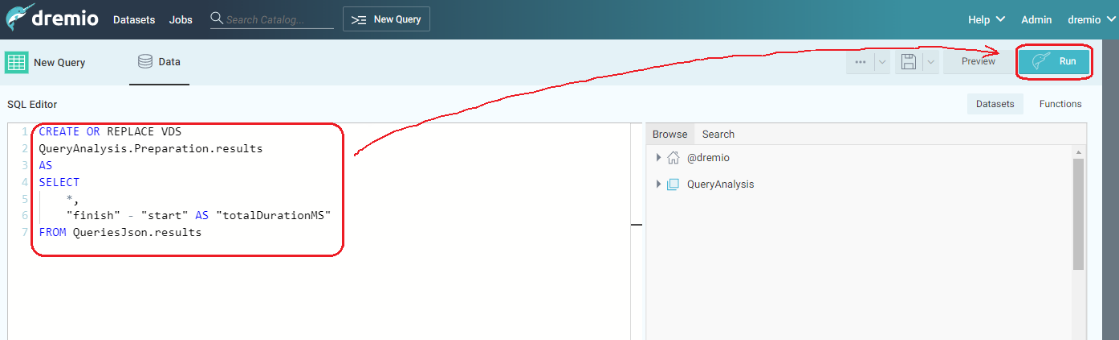
33\_QueryAnalysis.Application.NtileQueryDuration.sql

For each of the files above in numeric order do the following:

* Copy the SQL out of the file
* In the Dremio UI, click on New Query



* Paste the SQL into the SQL Editor and click Run. This will create the VDS.



## Analyze Results

Dremio is now available to query the VDSs in order to start analyzing the queries that have been flowing through your Dremio cluster. The VDSs supplied in this package are common examples of the sorts of questions a Dremio DBA might ask about the platform usage. Dremio encourages you to create your own VDSs as well in order to gain further insights.

Details of each of the VDSs in this package are shown below:

| **Create Order** | **VDS in QueryAnalysis space** | **Description** |
| --- | --- | --- |
| 1a | Preparation.results | Directly queries the QueriesJson.results PDS.  *Important: You may need to alter the FROM clause to point to the PDS on your system containing your scrubbed query data if the name is different than* ***QueriesJson.results*** |
| 1b | Preparation.chunks | Directly queries the QueriesJson.chunks PDS.  *Important: You may need to alter the FROM clause to point to the PDS on your system containing your chunks of query text data if the name is different than* ***QueriesJson.chunks*** |
| 1c | Preparation.errormessages | Directly queries the QueriesJson.errormessages PDS.  *Important: You may need to alter the FROM clause to point to the PDS on your system containing your query error message data if the name is different than* ***QueriesJson.errormessages*** |
| 1d | Preparation.errorchunks | Directly queries the QueriesJson.errorchunks PDS.  *Important: You may need to alter the FROM clause to point to the PDS on your system containing your query error message data if the name is different than* ***QueriesJson.errorchunks*** |
| 2a | Business.SelectQueryData | Builds upon the results VDS, it filters out all except SELECT queries.  *Important: Replace <DREMIO\_HOST> in the query with the DNS name or IP address of the cluster. This is to identify this cluster in queries* |
| 2b | MDRefreshData | All METADATA\_REFRESH jobs. JSON, CSV, deltallake tables do not have MDREFRESH jobs… they are all processed on the coordinator node. ONLY parquet based datasets have MD refresh jobs. |
| 2c | ReflectionRefreshData | All refresh refresh jobs. All LOAD MATERIALIZATION jobs are excluded. |
| 2d | Business.QueryTextChunks | Builds upon the chunks VDS and selects all fields from it. Can be joined with other VDSs via the queryId field in order to present the entire queryText for a query if it is longer than 4000 characters. |
| 2e | Business.QueryErrorMessages | Builds upon the errormessages VDS and selects all fields from it. Can be joined with other VDSs via the queryId field. |
| 3 | Application.AcceleratedSelects | All accelerated queries |
| 9 | Application.Top20Execution Times | List of the top 20 queries with the highest execution times |
| 15 | Application.PctAccelerated | Summary of the percentage of completed queries that were accelerated by reflections |
| 19 | Application.ActiveUsers | Summary per day of the total number of queries executed by each user and the total wall clock time spent running those queries. |
| 20 | **DailyRreportByOutcomePivoted** | Daily count of completed, failed and canceled queries across all job types (user queries, reflections and metadata refresh) |
| 32 | Application.NtileQueryCost | 100 percentile buckets of all queries by QueryCost |

*VDS definitions in this package*

**Note**: These VDS are designed to be examples. Consider modifications or additional datasets you could create:

* QueryAnalysis.Business.SelectQueryData is an example of how to filter out INSERTS, UPDATES, and DELETES. Use other fields of the Preparation.results VDS to filter on selected query types, durations, user names, or other characteristics.
* The application layer VDSs are typical of the datasets for end-users and could be analyzed with Tableau or other front-end tools.
* Please share your new VDS and query analysis insights with others by contacting Dremio Professional Services.

# Examples

1. AcceleratedSelect

SELECT \*

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "accelerated" = TRUE

1. ActiveUsers

SELECT "TO\_DATE"("startTime") AS "queryStartDate", "username",

COUNT(\*) AS "totalQueries", SUM("totalDurationMS") AS "totalWallclockMS"

FROM "QueryAnalysis"."Business"."SelectQueryData"

GROUP BY "username", "queryStartDate"

ORDER BY "queryStartDate" DESC, "totalQueries" DESC, "totalWallclockMS" DESC

1. FailedQueriesPerUser

SELECT "TO\_DATE"("startTime") AS "queryStartDate",

"username", COUNT(\*) AS "failedQueries"

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "OUTCOME" = 'FAILED'

GROUP BY "username", "queryStartDate"

ORDER BY "queryStartDate" DESC, "failedQueries" DESC

1. HighCostSelects

SELECT \*

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "queueName" = 'High Cost User Queries'

1. MostUsedDatasetsPerDay

SELECT "queryStartDate", "dataset",

COUNT(\*) AS "totalQueries",

COUNT(DISTINCT "username") AS "totalUsers"

FROM (SELECT "queryStartDate",

CAST("convert\_from"("convert\_to"("dataset", 'JSON'), 'UTF8') AS VARCHAR) AS "dataset", "username"

FROM (

SELECT CAST(startTime as DATE) AS queryStartDate,

"FLATTEN"("parentsList") AS "dataset", "username"

FROM "QueryAnalysis"."Business"."SelectQueryData"

) AS "nested\_0") AS "RawFlattenDataset"

GROUP BY "dataset", "queryStartDate"

ORDER BY "queryStartDate" DESC, "totalQueries" DESC

1. NonAcceleratedSelects

SELECT \*

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "accelerated" = FALSE AND "totalDurationMS" > 1000

ORDER BY "totalDurationMS" DESC

1. NtileQueryCost

SELECT "pct", MAX("queryCost") AS "maxQueryCost"

FROM (SELECT "queryCost",

NTILE(100) OVER (ORDER BY "queryCost") AS "pct"

FROM "QueryAnalysis"."Business"."SelectQueryData") AS "a"

GROUP BY "pct"

ORDER BY "pct"

1. NtileQueryDuration

SELECT "pct", MAX("totalDurationMS") AS "maxTotalDurationMS"

FROM (SELECT "totalDurationMS",

NTILE(100) OVER (ORDER BY "totalDurationMS") AS "pct"

FROM "QueryAnalysis"."Business"."SelectQueryData") AS "a"

GROUP BY "pct"

ORDER BY "pct"

1. OverallWorkloadDistribution

SELECT "pct", MAX("queryCost") AS "maxQueryCost"

FROM (SELECT "queryCost",

NTILE(100) OVER (ORDER BY "queryCost") AS "pct"

FROM "QueryAnalysis"."Business"."SelectQueryData") AS "a"

GROUP BY "pct"

ORDER BY "pct"

1. PctQueueUsageAllSartedQueries

SELECT "queueName", COUNT("queueName") AS "numQueriesPerQueue",

CAST(COUNT("queueName") \* 100 AS FLOAT) / (SELECT COUNT(\*)

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "outcome" IN ('COMPLETED', 'FAILED', 'CANCELED')

AND "executionTime" IS NOT NULL AND "queryCost" > 10) As pctOfTotal

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "outcome" IN ('COMPLETED', 'FAILED', 'CANCELED')

AND "executionTime" IS NOT NULL AND "queryCost" > 10

GROUP BY "queueName"

1. QueriesNotCompleted

SELECT \*

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "outcome" NOT IN ('COMPLETED')

1. QueryConcurrency

SELECT "this"."queryId" AS "thisQueryId", "this"."queryText" AS "thisQueryText",

"this"."startTime" AS "thisStartTime", "this"."finishTime" AS "thisFinishTime",

"others"."queryId" AS "otherQueryId", "others"."startTime" AS "otherStartTime",

"others"."FinishTime" AS "otherFinishTime",

CASE WHEN "others"."StartTime" < "this"."startTime"

AND "others"."finishTime" > "this"."finishTime" THEN 'Other Started Before This and Finished After This'

WHEN "others"."StartTime" < "this"."startTime"

AND "others"."finishTime" <= "this"."finishTime" THEN 'Other Started Before This and Finished During This'

WHEN "others"."StartTime" >= "this"."startTime"

AND "others"."finishTime" <= "this"."FinishTime" THEN 'Other Ran During This'

WHEN "others"."StartTime" >= "this"."startTime"

AND "others"."finishTime" > "this"."FinishTime" THEN 'Other Started During This and Finished After This'

ELSE NULL END AS "relation"

FROM "QueryAnalysis"."Business"."SelectQueryData" AS "this"

LEFT JOIN "QueryAnalysis"."Business"."SelectQueryData" AS "others" ON "others"."startTime" < "this"."startTime"

AND "others"."FinishTime" > "this"."StartTime" OR "others"."StartTime" >= "this"."startTime"

AND "others"."startTime" <= "this"."FinishTime" AND "this"."queryId" <> "others"."queryId"

AND "this"."outcome" IN ('COMPLETED', 'FAILED')

1. QueryConcurrencyCount

SELECT "thisQueryId", COUNT("thisQueryId") AS "concurrency"

FROM "QueryAnalysis"."Application"."QueryConcurrency"

GROUP BY "thisQueryId"

ORDER BY "concurrency" DESC

1. QueryCostVsExecutionTime

SELECT "queryId", "queryText", "executionTime", "queryCost", "accelerated"

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "outcome" = 'COMPLETED'

1. QueueMaxQueriesPerSecond

SELECT "qc"."queueName", MAX("qc"."queriesPerSecond") AS "maxQPS"

FROM (SELECT "queueName", "DATE\_TRUNC"('second', "startTime") AS "startSecond",

COUNT(\*) AS "queriesPerSecond"

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "queueName" <> '' AND "queueName" <> 'UI Previews'

GROUP BY "queueName", "startSecond"

ORDER BY "queueName", "startSecond") AS "qc"

GROUP BY "qc"."queueName"

1. Range\_EnQueuedTime

WITH "T1" AS (SELECT CASE WHEN "enqueuedTime" <= 1000 THEN 'R00 < 1000ms i.e Less than 1 sec'

WHEN "enqueuedTime" <= 10000 THEN 'R01 < 10000ms i.e Less than 10 secs'

WHEN "enqueuedTime" <= 60000 THEN 'R02 < 60000ms i.e. Less than 60 secs'

WHEN "enqueuedTime" <= 120000 THEN 'R03 < 120000ms i.e. Less than 2 minutes'

WHEN "enqueuedTime" <= 180000 THEN 'R04 < 180000ms i.e. Less than 3 minutes'

WHEN "enqueuedTime" <= 300000 THEN 'R05 < 300000ms i.e. Less than 5 minutes'

WHEN "enqueuedTime" <= 600000 THEN 'R06 < 600000ms i.e. Less than 10 minutes'

WHEN "enqueuedTime" <= 1200000 THEN 'R07 < 1200000ms i.e. Less than 20 minutes'

WHEN "enqueuedTime" <= 1800000 THEN 'R08 < 1800000ms i.e. Less than 30 minutes'

WHEN "enqueuedTime" <= 3600000 THEN 'R09 < 1800000ms i.e. Less than 60 minutes (1 hour)'

ELSE 'R10 > 1800000ms i.e. Greater than 60 minutes(1 hours)' END AS "RangeEnQueuedTime"

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "enqueuedTime" IS NOT NULL), "T2" AS (SELECT SUM(1) AS "TotalQ"

FROM "T1") (SELECT "RangeEnQueuedTime", COUNT(\*) AS "querycount",

CAST(COUNT(\*) \* 100.00 / "T2"."TotalQ" AS DECIMAL(5, 2)) AS "PerCentQs"

FROM "T1",

"T2"

GROUP BY "RangeEnQueuedTime", "T2"."TotalQ")

ORDER BY "RangeEnQueuedTime"

1. Range\_ExecutionTime

WITH "T1" AS

(SELECT CASE WHEN "executionTime" <= 1000 THEN 'R00 < 1000ms i.e Less than 1 sec'

WHEN "executionTime" <= 10000 THEN 'R01 < 10000ms i.e Less than 10 secs'

WHEN "executionTime" <= 60000 THEN 'R02 < 60000ms i.e. Less than 60 secs'

WHEN "executionTime" <= 120000 THEN 'R03 < 120000ms i.e. Less than 2 minutes'

WHEN "executionTime" <= 180000 THEN 'R04 < 180000ms i.e. Less than 3 minutes'

WHEN "executionTime" <= 300000 THEN 'R05 < 300000ms i.e. Less than 5 minutes'

WHEN "executionTime" <= 600000 THEN 'R06 < 600000ms i.e. Less than 10 minutes'

WHEN "executionTime" <= 1200000 THEN 'R07 < 1200000ms i.e. Less than 20 minutes'

WHEN "executionTime" <= 1800000 THEN 'R08 < 1800000ms i.e. Less than 30 minutes'

WHEN "executionTime" <= 3600000 THEN 'R09 < 3600000ms i.e. Less than 60 minutes (1 hour)'

ELSE 'R10 > 3600000ms i.e. Greater than 60 minutes(1 hours)'

END AS "RangeExecutionTime"

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "executionTime" IS NOT NULL), "T2" AS (SELECT SUM(1) AS "TotalQ"

FROM "T1") (SELECT "RangeExecutionTime", COUNT(\*) AS "querycount",

CAST(COUNT(\*) \* 100.00 / "T2"."TotalQ" AS DECIMAL(5, 2)) AS "PerCentQs"

FROM "T1",

"T2"

GROUP BY "RangeExecutionTime", "T2"."TotalQ")

ORDER BY "RangeExecutionTime"

1. SelectWorkloadDistribution

WITH "t1" AS (SELECT "TO\_DATE"("startTime") AS "queryStartDate",

"DATE\_PART"('hour', "StartTime") AS "queryStartHour",

"DATE\_TRUNC"('second', "startTime") AS "queryStartSecond",

COUNT(\*) AS "queriesPerSecond"

FROM "QueryAnalysis"."Business"."SelectQueryData"

GROUP BY "queryStartDate", "queryStartHour", "queryStartSecond")

(SELECT "queryStartDate", "queryStartHour",

SUM("queriesPerSecond") AS "queriesPerHour",

MAX("queriesPerSecond") AS "peakQueriesPerSecond"

FROM "t1"

GROUP BY "queryStartDate", "queryStartHour")

ORDER BY "queryStartDate" DESC, "queryStartHour"

1. SummaryQueueExecTimeByQueue

WITH "T1" AS (SELECT "queueName", 1 AS "QueryCount",

CASE WHEN COALESCE("enqueuedTime") / 1000 <= 0 THEN 0 ELSE 1 END AS "WaitQuery",

COALESCE("enqueuedTime", 0) / 1000 AS "QueueTimeSec",

COALESCE("executionTime", 0) / 1000 AS "ExecTimeSec"

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "queueName" <> '' AND "outcome" = 'COMPLETED')

(SELECT "queueName", SUM("QueryCount") AS "TotalQueryCount",

SUM("WaitQuery") AS "QueuedQueries",

100 \* SUM("WaitQuery") / SUM("QueryCount") AS "PercentageQueued",

MAX("QueueTimeSec") AS "MaxQueueTimeSec",

MAX("ExecTimeSec") AS "MaxExecTimeSec",

MAX("QueueTimeSec") / 60 AS "MaxQueueTimeMinutes",

MAX("ExecTimeSec") / 60 AS "MaxExecTimeMinutes"

FROM "T1"

GROUP BY "queueName")

ORDER BY 2 DESC

1. Top20DataSetsQueried

SELECT "dataSet", "dataSetType", COUNT(\*) AS "countTimesQueried"

FROM (

SELECT "list\_to\_delimited\_string"("nested\_0"."parentsList"."datasetPathList", '.') AS "dataSet",

"nested\_0"."parentsList"."type" AS "dataSetType"

FROM (

SELECT "FLATTEN"("parentsList") AS "parentsList"

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "parentsList" IS NOT NULL

) AS "nested\_0"

) AS "nested\_1"

GROUP BY "dataSet", "dataSetType"

ORDER BY "countTimesQueried" DESC

FETCH NEXT 20 ROWS ONLY

**Note**: in Dremio v21, *datasetPathList* has changed to *datasetPath*

1. Top20ExecutionTimes

SELECT \*

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "executionTime" IS NOT NULL

ORDER BY "executionTime" DESC

FETCH NEXT 20 ROWS ONLY

1. Top20PlanningTimes

SELECT \*

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "planningTime" IS NOT NULL

ORDER BY "planningTime" DESC

FETCH NEXT 20 ROWS ONLY

1. Top20TotalDurationTimes

SELECT \*

FROM "QueryAnalysis"."Business"."SelectQueryData"

ORDER BY "totalDurationMS" DESC

FETCH NEXT 20 ROWS ONLY

1. TopQueriesPerMinute

SELECT "DATE\_TRUNC"('minute', "startTime") AS "startMinute",

COUNT(\*) AS "queriesPerMinute"

FROM "QueryAnalysis"."Business"."SelectQueryData"

GROUP BY "startMinute"

ORDER BY "startMinute"

1. TotalQueriesPerMinutePerUser

SELECT "DATE\_TRUNC"('minute', "startTime") AS "startMinute",

"username", "outcome", COUNT(\*) AS "queriesPerMinute"

FROM "QueryAnalysis"."Business"."SelectQueryData"

GROUP BY "startMinute", "username", "outcome"

ORDER BY "startMinute", "username", "outcome"

1. TopQueriesPerMinutePerQueue

SELECT "DATE\_TRUNC"('minute', "startTime") AS "startMinute",

"queueName", COUNT(\*) AS "queriesPerMinute"

FROM "QueryAnalysis"."Business"."SelectQueryData"

WHERE "queueName" <> ''

GROUP BY "startMinute", "queueName"

ORDER BY "queriesPerMinute" DESC, "startMinute"

1. TotalQueriesPerUser

SELECT "username", "outcome", COUNT("outcome") AS "countQueries"

FROM "QueryAnalysis"."Business"."SelectQueryData"

GROUP BY "username", "outcome"

ORDER BY "username", "outcome"

1. TotalQueriesStartedPerSecond

SELECT "DATE\_TRUNC"('second', "startTime") AS "startSecond",

COUNT(\*) AS "queriesPerSecond"

FROM "QueryAnalysis"."Business"."SelectQueryData"

GROUP BY "startSecond"

ORDER BY "startSecond"

1. ReflectionRefreshData

SELECT queryId, queryText, reflection\_id, queryChunkSizeBytes, nrQueryChunks, startTime, finishTime, totalDurationMinutes, totalDurationSeconds, totalDurationMS, rownumByTotalDurationMS, outcome, username, requestType, queryType, parentsList, queueName, poolWaitTime, planningTime, context, engineName, enqueuedTime, executionTime, accelerated, inputRecords, inputBytes, outputRecords, outputBytes, queryCost, memoryAllocated, outcomereason, profileUrl

FROM (

SELECT \*

FROM (SELECT "queryId", "queryText", CASE WHEN length(substr(queryText, 21, 36)) > 0 THEN substr(queryText, 21, 36) ELSE NULL END AS reflection\_id,"queryChunkSizeBytes", "nrQueryChunks", "startTime", "finishTime", CAST("totalDurationMS" / 60000.000 AS DECIMAL(10, 3)) AS "totalDurationMinutes", CAST("totalDurationMS" / 1000.000 AS DECIMAL(10, 3)) AS "totalDurationSeconds", "totalDurationMS", ROW\_NUMBER() OVER (ORDER BY "totalDurationMS" DESC) AS "rownumByTotalDurationMS", "outcome", "username", "requestType", "queryType", "parentsList", "queueName", "poolWaitTime", "planningTime", "context", "engineName", "enqueuedTime", "executionTime", "accelerated", "inputRecords", "inputBytes", "outputRecords", "outputBytes", "queryCost",memoryAllocated, "outcomereason", "CONCAT"('http://<DREMIO\_HOST>:9047/jobs?#', "queryId") AS "profileUrl"

FROM "QueryAnalysis"."Preparation"."results" AS "results"

--WHERE "SUBSTR"(UPPER("queryText"), "STRPOS"(UPPER("queryText"), 'SELECT')) LIKE 'SELECT%' AND UPPER("queryText") NOT LIKE 'CREATE TABLE%' AND "requestType" IN ('RUN\_SQL', 'EXECUTE\_PREPARE')) AS "t"

WHERE "SUBSTR"(UPPER("queryText"), "STRPOS"(UPPER("queryText"), 'REFRESH REFLECTION')) LIKE 'REFRESH REFLECTION%') AS "t"

WHERE "rownumByTotalDurationMS" > 0

and startTime > '2022-11-11'

) nested\_0

1. ReflectionThresholds\_2Queues

SELECT MAX(CASE WHEN "a"."pct" = 80 THEN "a"."queryCost" ELSE 0 END) AS "upper\_low\_cost\_threshold"

FROM (SELECT "queryCost", NTILE(100) OVER (ORDER BY "queryCost") AS "pct"

FROM QueryAnalysis.Business.SelectQueryDataReflections

WHERE "outcome" = 'COMPLETED' AND "requestType" IN ('RUN\_SQL', 'EXECUTE\_PREPARE') AND "queryCost" > 10 AND "queueName" <> 'UI Previews') AS "a"

1. ReflectionThresholds\_3Queues

SELECT MAX(CASE WHEN "a"."pct" = 75 THEN "a"."queryCost" ELSE 0 END) AS "upper\_low\_cost\_threshold", MAX(CASE WHEN "a"."pct" = 90 THEN "a"."queryCost" ELSE 0 END) AS "upper\_medium\_cost\_threshold"

FROM (SELECT "queryCost", NTILE(100) OVER (ORDER BY "queryCost") AS "pct"

FROM QueryAnalysis.Business.SelectQueryDataReflections

WHERE "outcome" = 'COMPLETED' AND "requestType" IN ('RUN\_SQL', 'EXECUTE\_PREPARE') AND "queryCost" > 10 AND "queueName" <> 'UI Previews') AS "a"

1. applcation.SummaryDurationByReflection

select ReflectionRefreshData.reflection\_id,

ReflectionRefreshData.queueName,

count(\*) as refresh\_cnt,

AVG(ReflectionRefreshData.enqueuedTime / 1000) avgQueueTimeSec,

AVG(ReflectionRefreshData.executionTime / 1000) avgExecTimeSec,

AVG(ReflectionRefreshData.queryCost)

AVGReflectionCost,

AVG(memoryAllocated) as avgMemoryAllocated

FROM QueryAnalysis.Business.ReflectionRefreshData

where outcome = 'COMPLETED'

group by ReflectionRefreshData.reflection\_id, ReflectionRefreshData.queueName

order by avgMemoryAllocated

1. applcation.SummaryReflectionExecTimeByQueue

WITH "T1" AS (SELECT "queueName", 1 AS "QueryCount", (CASE WHEN (((COALESCE("enqueuedTime")) / 1000) <= 0) THEN 0 ELSE 1 END) AS "WaitQuery",

((COALESCE("enqueuedTime", 0)) / 1000) AS "QueueTimeSec",

((COALESCE("executionTime", 0)) / 1000) AS "ExecTimeSec"

FROM QueryAnalysis.Business.ReflectionRefreshData

WHERE (("queueName" <> '') AND ("outcome" = 'COMPLETED'))) (

SELECT "queueName", SUM("QueryCount") AS "TotalQueryCount",

SUM("WaitQuery") AS "QueuedQueries", ((100 \* SUM("WaitQuery")) / SUM("QueryCount")) AS "PercentageQueued",

MAX("QueueTimeSec") AS "MaxQueueTimeSec", MAX("ExecTimeSec") AS "MaxExecTimeSec",

(AVG("QueueTimeSec")) AS "AvgQueueTimeSec", (AVG("ExecTimeSec")) AS "AvgExecTimeSec"

FROM "T1"

GROUP BY "queueName")

ORDER BY 2 DESC

# Getting Support

Dremio Query Analyzer is provided by Dremio Professional Services. Issues should be directed to Dremio Professional Services, who provide support during the engagement on a best-effort basis.