

**Query Analyzer Documentation**

## Version Control :

| # | Changes | Author | Date | Reviewed By |
| --- | --- | --- | --- | --- |
| 1 | Initial Draft | Simran Jain | 7-July-2023 |  |
| 2 | Incorporated Review Comments from Joshi | Simran Jain | 19-July-2023 |  |

## **Table of Contents**

[**Version Control : 1**](#_heading=h.3dy6vkm)

[**Table of Contents 1**](#_heading=h.1t3h5sf)

[Objective: 3](#_heading=h.4d34og8)

[**Technical Overview : 3**](#_heading=h.2s8eyo1)

[Metrics Used : 4](#_heading=h.17dp8vu)

[Additional Categorical / Informational Parameters : 5](#_heading=h.3rdcrjn)

[Output: 6](#_heading=h.26in1rg)

[**Technical Details: 6**](#_heading=h.35nkun2)

[How to use / run the tool : 7](#_heading=h.1ksv4uv)

[Corrective Actions: 7](#_heading=h.44sinio)

[**Troubleshooting: 8**](#_heading=h.2jxsxqh)

[**List of Artifacts: 8**](#_heading=h.z337ya)

## Objective:

To provide query level analysis in a report to be consumed by the right people to take needed action. A consolidated view of all pain points of a particular query is not visible in a single place.

With Query Analyzer, we provide a holistic analysis of a query on a combination of metrics for cost, performance or query optimization.

## Technical Overview :

1. The wrapper Stored Procedure: QUERY\_ANALYZER\_REPORT() takes Long running queries from MONITOR\_DB.PERFORMANCE\_SCHEMA.BAD\_QUERY table (PERFORMANCE = ‘LONG’)
2. For each of the QUERY\_IDs fetched, a child stored procedure QUERY\_ANALYZER( qid varchar) is called which does the following :
   1. Key KPI / metrics (a combination of calculated and static information shown in section [Metrics Used](#_heading=h.17dp8vu)) are read from QUERY\_HISTORY stored in the QUERY\_PERFORMANCE\_KPI table.
   2. Top 3 most expensive nodes of the query are also recorded in the same table
3. The QUERY\_PERFORMANCE\_KPI table is consumed in 2 ways :
   1. SNOWSIGHT dashboard to show the details and summary
   2. An alert mechanism :
      1. using NOTIFICATION INTEGRATION
      2. Any other mechanisms can be integrated. (yet to be explored)
4. RESTART Capability : To have the restart / periodic run capabilities, QUERY\_ANALYZER\_REPORT() updates the last\_max\_start\_time in the CONTROL\_TABLE in MONITOR\_DB database with the START\_TIME of the query processed last.
5. SCHEDULE : runs at 3 AM UTC everyday under QUERY\_ANALYZER\_TASK.
6. If the same query is running multiple times, the analysis report of the latest run of the query is retained and the older ones are purged
7. Design Considerations:
   1. Duplicate queries (running daily or frequently), will be removed using a string similarity logic( JAROWINKLER\_SIMILARITY () function) so that there are no duplicate entries for the same query.
   2. QUERY\_TEXT is not included in the QUERY\_PERFORMANCE\_KPI table to make sure that the size of the table is under check. However, the MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_HISTORY backup table can always be used to retrieve the query\_text.
   3. A Control table records the START\_TIME of the last analyzed query in the previous run as LAST\_QUERY\_START\_TIME so that the subsequent run knows the already processed queries.
   4. We only fetch the long queries from BAD\_QUERY (existing table for dashboards) to provide relevant recommendation
   5. The QUERY\_ANALYZER\_TASK is scheduled to run 1 hour later than the update of the query history backup table (QUERY\_HISTORY\_TABLE) so that no data is missed.
   6. Instead of picking directly from Query history table, we intend to keep backup upto 2 years in MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_HISTORY table and stored procedure fetches all the information from that table.
   7. We have kept the Query Analyzer code and the wrapper stored procedure(QUERY\_ANALYZER\_REPORT) separate for efficient update in KPI table
   8. Jaro Winkler algorithm is used to identify similar queries and eliminate duplicates in the report

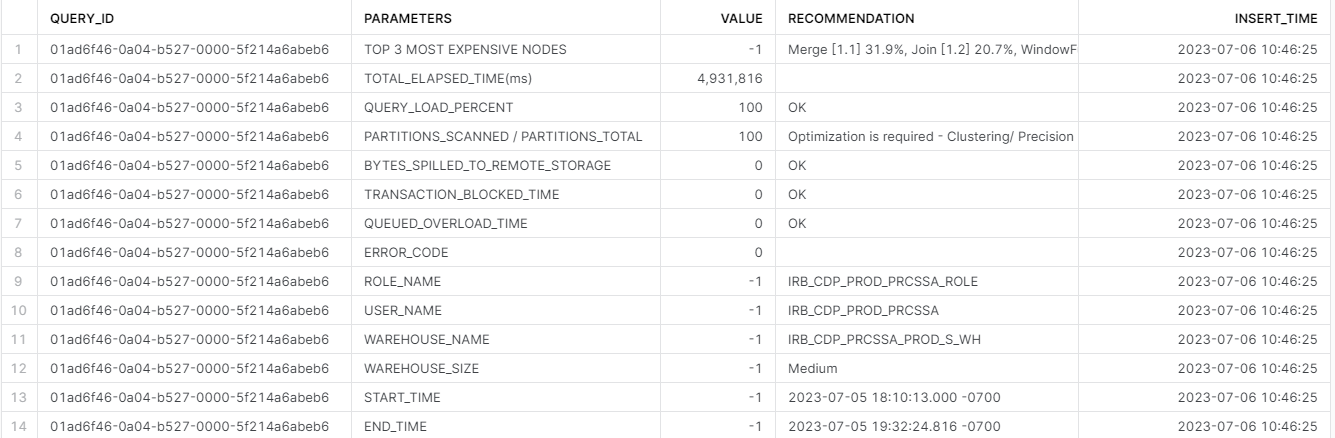
## Metrics Used :

| **Metric** | **Source** | **Recommendation** | **Output Strings** |
| --- | --- | --- | --- |
| TOP 3 MOST EXPENSIVE NODES | GET\_QUERY\_OPERATOR\_STATS() | We pick the 3 most expensive nodes from get\_query\_operator\_stats() function and display them in the below format :  The 3 entries are separated by comma.  This would help identify what the most credit consuming nodes and where to look for query improvement | Operator(Step\_id.Effective\_operator\_id) Overall Percentage [3 times] |
| TOTAL\_ELAPSED\_TIME(in ms) | QUERY\_HISTORY | Displaying time as-is in milliseconds  If the overhead time is more than 60% of the total elapsed time, a specific message is shown | * OK * Time spent on overhead activities is too high : x% * Long Running Query |
| QUERY\_LOAD\_PERCENT | QUERY\_HISTORY | If QUERY\_LOAD\_PERCENT < 50:  Recommendation = "If query load percentage is frequently less than 50 percent, adjust warehouse size to a smaller size or move this query to a different warehouse " | * OK * If query load percentage is frequently less than 50 percent, adjust warehouse size to a smaller size |
| PARTITIONS\_SCANNED / PARTITIONS\_TOTAL | QUERY\_HISTORY | if partition\_scanned > 0.80\*partition\_total:  recommendation = "Optimization is required - Do Clustering on the tables involved in query or Precision in where condition" | * OK * Optimization is required - Clustering/ Precision in where condition |
| BYTES\_SPILLED\_TO\_REMOTE\_STORAGE | QUERY\_HISTORY | if Bytes\_spilled\_to\_remote > 5\*bytes\_scanned:  recommendation = "Spillage is happening. Use a larger warehouse" | * OK * Excessive remote spillage observed |
| TRANSACTION\_BLOCKED\_TIME | QUERY\_HISTORY | if transaction\_blocked\_time >0:  recommendation = "If this repeats for the query frequently,take action accordingly" | * OK * If this repeats for the query frequently,take action accordingly |
| QUEUED\_OVERLOAD\_TIME | QUERY\_HISTORY | if queued != 0:  recommendation = "Optimization required - Increase cluster number may help.Helps in performance, not cost reduction" | * OK * Increase the number of clusters on the warehouse to optimize performance. (Helps in performance, not cost reduction) |

### Additional Categorical / Informational Parameters :

| **Metric** | **Source** | **Information** |
| --- | --- | --- |
| ROLE\_NAME | QUERY\_HISTORY | Display the ROLE used to submit the query |
| USER\_NAME | QUERY\_HISTORY | Display the USER who submitted |
| ERROR\_CODE | QUERY\_HISTORY | Display ERROR message if any |
| WAREHOUSE\_NAME | QUERY\_HISTORY | Displays the WAREHOUSE name on which query was submitted |
| WAREHOUSE\_SIZE | QUERY\_HISTORY | Displays the WAREHOUSE SIZE of the warehouse during the submission (null for CSL queries) |
| START\_TIME | QUERY\_HISTORY | Query Start Time |
| END\_TIME | QUERY\_HISTORY | Query End Time |

## Output:



## 

## Technical Details:

| Language of Stored Procedure | SQL |
| --- | --- |
| Performance | 9 to 10s / query  *Initial tests showed that processing 1 query takes about 10 seconds. Out of the 10 seconds, 70% if the time is used by the GET\_QUERY\_OPERATOR\_STATS() function* |
| Schedule | Daily at 03:00 UTC |
| Source Tables | * Backup of QUERY\_HISTORY - to improve performance, we use a backup * BAD\_QUERY under MONITOR\_DB - to identify the long running queries * CONTROL\_TABLE - to get info about the last query processed |
| Restart Capability on Failure / Timeout | In case the Task fails due to timeout or any other issue, check for the max start\_time in the QUERY\_PERFORMANCE\_KPI table and update the CONTROL\_TABLE. The next scheduled run should pick the data correctly  If information is needed urgently, manually trigger the stored procedure QUERY\_ANALYZER\_REPORT() using the role MONITOR\_IRB\_ADMIN or a parent role |
| Security | * QUERY\_ANALYZER is coded as “EXECUTE AS OWNER”. Any user can execute the procedure for a specific query\_id * To run QUERY\_ANALYZER\_REPORT(), user must have MONITOR\_IRB\_ADMIN or parent role |
| Notification | An email is sent to the Stakeholders' email id <to be added> who will be responsible to take the needed action on the query |

## How to use / run the tool :

There are 3 ways to use the utility

* The tool is scheduled to run Daily at a fixed time using a scheduled task called QUERY\_ANALYZER\_TASK. The report generated for all PROD queries is sent by email. *The recipient can check the Notification received and take the needed action.*
* QUERY ANALYZER DASHBOARD on Snowsight can also be used to see the results for PROD and non-PROD queries
* If an ad-hoc analysis is needed for a specific non-PROD query, the core engine QUERY\_ANALYZER can be run by passing the specific query\_id

## [Corrective Actions](https://docs.snowflake.com/en/sql-reference/account-usage/query_history):

1. If Total Elapsed Time is more, we have additionally calculated time spent on overhead to analyze if the query is spending more time processing or waiting for resources.

Ref : [Query Performance](https://docs.snowflake.com/en/user-guide/performance-query-warehouse)

1. Query\_load\_percent

Ref: [Clustering](https://docs.snowflake.com/en/user-guide/tables-clustering-keys)

1. Start\_Time, End\_time : Could be used as filters for dashboards and analysis.
2. Role name, User name, Warehouse name and Warehouse size are given to identify which environment or internal companies groups the query triggered belongs to and would later work on suggestions.
3. Spillage : We additionally give the spillage ratio as well to help understand the recommendation better

Ref: [Performance-impact-from-local-and-remote-disk-spilling](https://community.snowflake.com/s/article/Performance-impact-from-local-and-remote-disk-spilling)

[Resolving Memory Spillage](https://docs.snowflake.com/en/user-guide/performance-query-warehouse-mem)

## Troubleshooting:

* If the task failed due to timeout
  + Check if the data to be processed was too high (query count run in last 24 hours)
  + Increase the time-out of the Task QUERY\_ANALYZER\_TASK
  + Current value is 5 hours.
* Task failed due to some other reason
  + Check the task failure reason. Most times the reason could be due to the failure of the stored procedure run within
  + Check the query history of the last run of the Task and take relevant action. It could mostly be access issues etc.
* Report received is empty
  + Check from the Query Analyzer Dashboard if the data for the “Last Day” is empty
  + If not empty,
    - the issue is with the notification mechanism
  + If empty,
    - check if there were no long running queries in last 1 day (BAD\_QUERY table / Performance Dashboard)
      * If empty, no problem. There was genuinely NO data to report
      * If not empty, check if the QUERY\_ANALYZER\_TASK is suspended or if it failed
* Dashboard is not showing any data when filter is set to Last 7 Days
  + check if there were no long running queries in last 1 day (BAD\_QUERY table / Performance Dashboard)
    - If empty, no problem. There was genuinely NO data to report
    - If not empty, check if the QUERY\_ANALYZER\_TASK is suspended or if it failed
* Not able to view Dashboard
  + Contact DBA / Admin to provide the needed access

## List of Artifacts:

1. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.CONTROL\_TABLE
2. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.DURATION\_FILTER(created for dashboard)
3. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_HISTORY
4. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_PERFORMANCE\_KPI
5. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_SIMILARITY\_COUNT
6. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_ANALYZER\_TASK
7. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_HISTORY\_TASK
8. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_ANALYZER("QID" VARCHAR(16777216))
9. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_ANALYZER\_REPORT()
10. MONITOR\_DB.COMMON\_TABLE\_SCHEMA.QUERY\_HISTORY\_SP()