|  |
| --- |
|  |

## Query Troubleshooting process

This section is to assist in the process you should go through when you have identified a “problem query” and want to work on optimizing it. The instructor needs to be comfortable with troubleshooting queries as this is not a comprehensive guide, but more a starting point.

### Compare the estimated and actual execution plan

When troubleshooting a query it’s useful to start with comparing the estimated plan, obtained by placing the keyword EXPLAIN in front of the query text and the actual execution plan, obtained by viewing the data in sys.dm\_pdw\_request\_steps or category == ‘RequestSteps’ in log analytics. You are looking for instances where the estimation is orders of magnitude off from the actual execution. This is an indicator that there is some issue with the query and after you make changes you can review this information again to see if a change “fixed” the issue.

1. Open Visual Studio – SQL Server data Tools and connect to your Synapse SQL Pool database
2. Place the keyword EXPLAIN in front of the query text and execute it. The query itself will not execute, it will just run through compilation and generate a query plan
3. If you are using SSDT, then results should be in a click-able link in the bottom pane. Click it to open up the query plan in a new tab
4. Look at DMV (sys.dm\_pdw\_request\_steps) or Log analytics (Category == ‘RequestSteps’) data for the actual execution of the query
5. Compare the two plans, take special note of the estimations made in the explain version of the plan compared to how many rows actually came out of those query steps as seen in the Log analytics or DMV data.

### Statistics

**Row Count**

You may use the ‘statistics accuracy’ query to compare how many rows SQL Server believes there are based on statistics compared to how many rows actually exist in the table. These should always remain no more than 20% different. Also consider the overall size of the tables involved. If you see a table that has 1 billion rows that is being broadcasted, there is probably something wrong with statistics.

**Proper Statistics Columns**

Even if the statistics row count is correct, you still need to make sure you have statistics created on the proper columns in order for the optimizer to make correct decisions. Statistics should exist on “involved” columns. This means columns that are used in:

1. Joins – any columns that are used in joins
2. Predicates, especially ORDER BY, WHERE, GROUP BY

**Full Scan vs Default Sampled Statistics**

In most cases, default sampled stats will be sufficient. FULLSCAN should only be considered in cases where estimations seem to be an issue, in which case FULLSCAN can be tested to see how it affects estimations/plan choice/performance. Full scan may be useful if the data is unique enough that the data in a random sample is not indicative of the whole data set. The sample % is chosen by SQL Server and will vary with different table sizes. FULLSCAN statistics take longer to create and update, so should only be used when needed.

**Auto Create Statistics**

When the database AUTO\_CREATE\_STATISTICS option is on, SQL Data Warehouse analyzes incoming user queries for missing statistics. If statistics are missing, the query optimizer creates statistics on individual columns in the query predicate or join condition to improve cardinality estimates for the query plan. Automatic creation of statistics is currently turned on by default.

In a production environment, it’s still a good idea to manually create statistics. This means you do not have a query waiting at execution time for statistics to be created.

You may read more about statistics in Azure Synapse SQL Pools here: [Table Statistics for dedicated SQL Pool in Azure Synapse Analytics](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/sql-data-warehouse-tables-statistics)

### Update statistics

Statistics are the most likely reason a query is not performing well and we should not move on to any other optimization steps until statistics are in place and up to date. The optimizer uses statistics for the MPP plan and the individual SQL nodes use statistics for the plans on each distribution. Poor statistics accuracy can lead to improper choices at the MPP level like choosing a broadcast instead of a shuffle, or at the SQL level like not allocating enough memory to a join causing tempdb spilling.

The first thing to do with any query you are investigating is to update existing statistics on all involved tables. To do this you can go through the query and pull out all of the table names, which is fine when the query is small, but as the query gets larger or you have views and nested views this can become unmanageable. For these instances you can use the PowerShell script ‘GetObjectFromExplain.ps1’ that will parse the explain plan and pull out all involved tables for you.

1. Save explain.xml to your computer and copy the path
2. Run GetObjectsFromExplain.ps1 from a PowerShell window
   1. Provide the path to the explain.xml when prompted

Text

Description automatically generated

1. View the files generated in the datestamped output path listed.. Update statistics commands for all involved tables are in tsql\_UpdateStatsCommands.txt

A screenshot of a cell phone screen with text

Description automatically generated

1. Run the update stats commands provided
2. Run an explain again and see if the plan steps or estimations changed. If there are any changes, try running the query again to see if performance improved.

### Create missing statistics

1. After running the GetObjectsFromExplain.ps1 in the previous step, look at all of the tables it listed and make sure there is at least one stat on each table. If there is not, then create it.
2. The tool also pulls involved columns and separately parses out shuffle columns. All columns listed should have stats. It is easiest to just create statistics on all columns listed for all tables listed. It is very difficult to correlate certain column names to corresponding tables, so the tool just pulls an all-inclusive list. You can use these lists to generate Create statistics commands.

### Evaluate wait vs execution time

Use the sys.dm\_pdw\_exec\_requests DMV or the LogAnalalytics query where Category == ‘Execrequests’ to evaluate if the query spent any time queued. Evaluate the intervals between:

**Submit\_time and end\_compile\_time:** This is considered compilation time (plan generation), but it also includes any time to create auto statistics if that feature is enabled. If plan generation is taking a long time not related to auto-stats, you can try providing a hint like “FORCE ORDER” to the query so limit the search space for plans. Just note that it may not be able to use the most-optimal plan if you do this, but if it runs faster it may be a better choice regardless.

**End\_compile\_time and Start\_time:** This is time spend queued. It may be queued due to concurrency limitations

**Start\_time and End\_time:** This is the time the query actually spent executing. Ideally you want nearly all of the execution time to be in this window.

### Evaluate concurrency

At the time of the query run, use the active vs queued queries and workload group metric charts to determine if there was a concurrency issue at the time. If there are queued queries, then the combination and number of workload groups that were running at the time exceeded the maximum. You can address this a few ways:

1. Reduce the workload group allocation for some queries – allowing more to run at a given time
2. Spread out the workload over a longer timeframe so you don’t have all the heavy queries hitting at the same time

You should also look at the DWU usage and TempDB charts. These can help indicate if there were “large” queries running at the time the problem query was executed.

NOTE: you can also set priority for a query using the classifier or workload group assignment. Higher priority will cause a particular query to jump to the front of the queue and spend less time waiting.

### Consider Hints

Hints can be added to a query by appending the OPTION clause to the end of the query. In particular, the FORCE ORDER hint has multiple uses when it comes to SQL Pool queries. This hint forces the joins to be executed in the order they are written instead of the optimizer moving join criteria to different parts in the plan to optimize the overall work. There are two main scenarios this has a big impact in SQL Pools:

* When a query is taking a long time to compile – the time between submit\_time and end\_compile\_time in exec requests.
  + FORCE ORDER shrinks the search space so the optimizer does not have to spend as long looking for the best query plan
* When a query has a step that seems to produce a cartesian product. You can look for things like a shuffle or broadcast move that is moving billions of rows, when that is not expected.
  + Sometimes the optimizer can be over-aggressive in optimizations that cause a few query patterns to produce cartesian products.

To try the query with the force order hint, add the following clause to the end of the query and re-run to test results:

OPTION(FORCE ORDER)

### Consider Workload Group Usage

You should balance the size of the workload group granted to a query vs overall concurrency that can be achieved with larger or smaller workload groups running. A larger workload group may allow a large query, like a large CTAS, to complete faster, but certain other queries may not need that many resources, meaning you are lowering concurrency unnecessarily. Testing with different resource allocations may be necessary.

### Evaluate Join Criteria

1. Is the distribution column included in the join criteria?
   1. This is not applicable for round robin or replicated tables, but could lead to questioning if the table SHOULD be a round robin or replicated tables. (see section on ‘Table Geometry’ for more info).
   2. Many times, the distribution columns is left out of the join criteria because it is not logically needed to complete the query, but in some of those cases, it can be easily added to the join criteria without changing the result set. If it can be included, it should be.
2. Are join fields calculated?
   1. Calculated fields in join criteria cannot leverage statistics and will almost always cause a broadcast move even if the tables involved have the same hash key. This is because the query engine uses a default value of 1000 for that half of the join and the tables are not technically being joined on the distribution column even if the calculated fields leverage the distribution column.
   2. Better to materialize these columns before joining
3. Are statistics in place and accurate for individual fields included in the join criteria?
   1. Join predicates will leverage single column stats and not composite stats. Generally, the accuracy of table value statistics are a good indication of the accuracy of the user defined statistics.
4. Join order of tables should be reviewed to support reduced data movement
   1. The joins are not necessarily executed in the order written, the optimizer may choose to change the order of joins because it believes it is more efficient. If you want to see how the plan is affected, you can add ‘OPTION(FORCE ORDER)’ to the end of the query to force the joins in the order written.

### Evaluate Aggregations

1. Are there GROUP BY clauses that do not include the distribution columns of the tables?
   1. Queries are always written to aggregate data based upon only the specific fields required for business logic. In some cases, the distribution column can be added into the group by clause and the logical result set is unchanged. If you can include the distribution column in the aggregation, you should.

### Evaluate Complex or sub-queries

1. Look for repeated subqueries that should be a single temp table instead .
   1. Temp tables also have the benefit as being held on the same NVMe storage as the adaptive cache, so accessing a temp table is faster than accessing a table in a distribution.
2. Look for complex subqueries with aggregations or windowing functions that could/should be moved to a temp table to gain clarity to the query and gain performance by executing smaller units of work thus requiring less memory per query.
3. Also remember, if you run a subquery, then join on the result of that query, you are building estimations upon estimations and the harder the subquery is to estimate, the more likely you are to see performance issues. In this case also consider using temp tables.

NOTE: up-to-date statistics are also important for temp tables

### Views

Views are not much different than multi-level nested subqueries. The MPP engine actually will still execute the query as a single unit of work and can require more memory than if the view was filtered and materialized in a temp table or intermediate physical table. Think of views as pasting the text of the view into the original query.

1. The query will generally require less memory if the view is filtered and materialized into a temp table.
2. Views make tuning efforts more complicated as the view must be opened and treated as a sub-query
   1. Table geometry, indexing, statistics, and partitioning of underlying tables still needs to be understood to effectively apply join and filter predicates and perform aggregations.

### Windowing Functions

Windowing functions like ROW\_NUMBER typically cause a broadcast move and a sort operation which may be unavoidable. This can be extra data movement as well as CPU-intensive

1. Avoid Windowing functions when possible
2. Always run windowing functions on the final select of a query.

### Case statements and Data Type Conversions

Case statements and data type conversions are CPU-intensive and can cause significantly increased CPU usage on compute nodes. It is better to do evaluations during the load process before the data actually makes it to the table.

### DISTINCT operator

The DISTINCT statement should be avoided where possible because it is possible for them to mask logic errors elsewhere in the query. It would be better to write the join criteria in such a way that the distinct is not needed.

### UNION vs UNION ALL operator

The UNION operator will concatenate two resultsets together and remove duplicates. In order to do this it has to order the data, which can be expensive depending on the data volume. If you have unions in your query you should evaluate if you can ensure there are no duplicates in the source data, then use a UNION ALL, which will perform much faster as it does not need to order the data.

### Index Health

Index health is perhaps the second-biggest cause of slow performance. Clustered Columnstore Indexes (CCI) in particular are largely managed by the system, but they still need to be monitored periodically and maintained to allow for better compression and performance. Sometimes index choice may need to be changed for particular tables based on the details around that table and how it is used.

To evaluate CCI health, you can start with the query CCI Health included with the delivery materials

1. Open Row Groups
   1. At all times you want to keep rows in OPEN row groups to a minimum. These are rows that have not yet been compressed into the columnstore and are sitting in the “delta store” with no index. This fragments reads between the compressed rowgroups and the deltastore and if a large number of rows are in the delta store you do not benefit from any indexing.
   2. You want to keep the percentage of rows in OPEN rowgroups to less than 5% of the total rows in the table
   3. Over-partitioning or not enough rows can cause rows to remain in open rowgroups.
2. Compressed Row groups
   1. You want all rows to be in compressed rowgroups with the average size as close to 1,048,576 rows (the maximum size of a rowgroup) for gest compression. Clustered indexes and heaps may become fragmented over time and may benefit from a rebuild occasionally. This fragmentation is not currently exposed in a DMV.

### Partitioning

Over-partitioned tables can be a big problem, tables with too many partitions consume more memory than necessary at query time. The DW engine must perform many context switches while piecing shards of data back together to complete and this will often spill to disk (TempDB). This can be costly for all queries in the environment when they go to use TempDB.

### Evaluate Heavy query operations

Sometimes when a query is reported as slow it’s not actually the reported query that has a problem. There are scenarios you can get into where an especially bad query is using up a large percentage of shared resources, namely tempdb or CPU. When you have a system that is “running slow” you can run the ‘Data Movement Heavy Hitters’ query included in the delivery materials. This will show queries that are currently moving the largest amount of data. These are the queries that are most likely to be impacting other queries.

### Evaluate external table usage

Ensure that when external tables are used the tables are just imported once and not multiple times. It is much more efficient to import the tables once, then reference the imported table in subsequent queries. This removes the need to import the entire table multiple times into the database.

## SLO Analysis

Determining if you are at the proper SLO is based on a few different metrics and you should take these metrics together as one big picture instead of just focusing on a single metric. In this section we will get those metrics and analyze what different values may mean as it pertains to scaling up or down and how it will affect workload performance. Scaling affects the following metrics:

* Linearly improves performance of the system for scans, aggregations, and CTAS statements
* Increases the number of readers and writers for loading data
* Maximum number of concurrent queries and concurrency slots

If we recommend scaling up it is either because the metrics show there may be a significant performance increase due to resource or queue utilization or to avoid issues of running out of resources. If we recommend scaling down it’s primarily to save cost. We can analyze whether or not there will likely be failures if the customer scales down, but we cannot quantify the overall workload performance impact of scaling down since it is very workload-dependent.

### DWU Usage

Look at the DWU usage chart created over the past 3 days, 7 days, and 30 days to identify how much DWU usage a typical workload uses.

Scale-up indicators

* DWU usage is frequently at 100% for more than just a few minutes at a time
* Better performance is required

Scale-down enablers

* DWU usage remains relatively low at all times with few spikes over 60-70%.
* If DWU usage is low in general with just a few spikes then it’s possible to optimize those queries, then scale down. If the spikes always occur at the same time it’s easier to identify the quereis causing it. Sometimes you can spread workload out instead of running multiple operations at the same time.

### tempdb usage

Analyze tempdb utilization over the past 3 days, 7 days, and 30 days. tempdb is a resource that when it runs out queries will fail, so it’s very important to keep tempdb from running out. Also note that the tempdb metric is an average of tempdb usage across all nodes in the SQL Pool. It’s possible if there is significant skew in the workload that the customer could receive out of space errors for tempdb even if the metric is showing space available – for instance 70%.

Scale-up indicators

* Users get errors that they are out of tempdb frequently
* Tempdb usage is regularly high – above 60-70%

Scale-down enablers

* TempDB never gets above 50%. This is an indicator that the customer probably CAN scale down without hitting errors.

### Adaptive Cache Usage

Adaptive cache caches columnstore segments that are frequently used so that they can be more quickly accessed by future queries. You basically don’t want the cache space used % to fill up

Scale-up indicators

* Adaptive cache space usage at 100% but cache hit % is not in the high 90’s. – this is a clear indicator that scaling up will increase performance through more efficient cache usage

Scale-down enablers

* Adaptive cache hit % is high, but space usage remains low after the workload has been running for multiple days

### Query Queueing

There are multiple ways to address query queueing, but scaling can help in multiple ways. At DW6000 and above it’s possible to run the maximum of 128 concurrent queries depending on your workload group configuration, but even as you scale beyond that queueing may be reduced by quereis executing faster, thus requiring queries to spend less time in the queue.

Scale-up Indicators

* Frequent queueing that is not addressed by workload group configuration changes

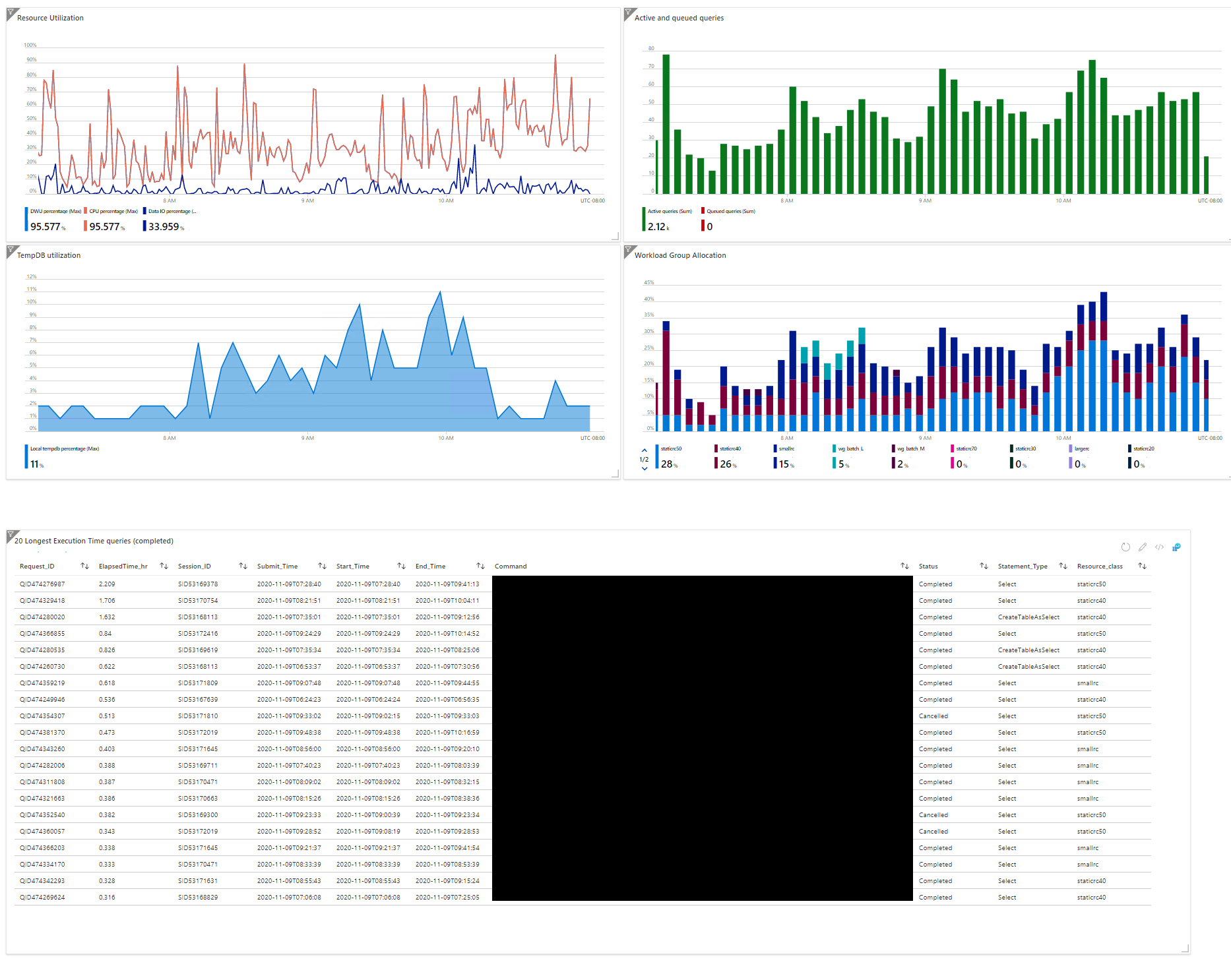
Scale-down enablers

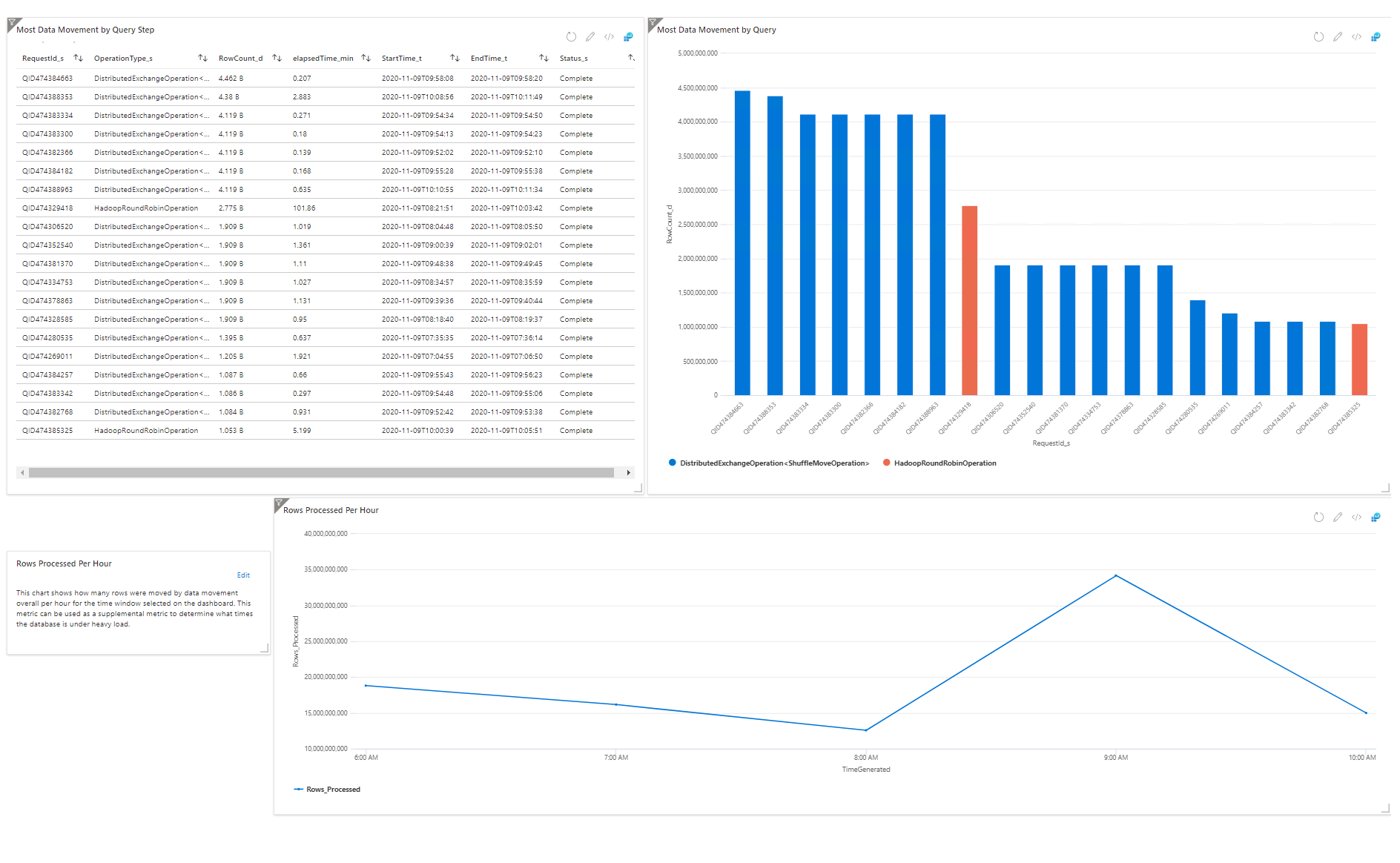
* Instance rarely sees any queueing

# Appendix

## Full Dashboard Screenshot

You want to size the charts so that they can be easily viewed on a single screen. In my example I zoomed out of the browser a little to fit some more information (CTRL +/-). Most importantly, you want to fit the first 4 charts onto a single pane where no scrolling is necessary. Here is an example layout (Some data obfuscated for privacy).





Graphical user interface, application, table, Excel

Description automatically generated

# Additional Resources

**Datasheet:**