Query Performance Tuning

Author: Serdar Altug ([saltug@microsoft.com](mailto:saltug@microsoft.com))

Embedded Escalation Engineer

Nick Salch (nicksalc@microsoft.com)

Support Escalation Engineer

Date: April 2017

Table of Contents

[Summary 3](#_Toc482195554)

[Pre-requisites: What you need to get started 6](#_Toc482195555)

[Part 1: Queries running slowly 8](#_Toc482195556)

[Part 2: Joins 12](#_Toc482195557)

[Part 3: Troubleshooting 15](#_Toc482195558)

[Part 4: Query performance improvements 18](#_Toc482195559)

[Part 5: Query analysis 20](#_Toc482195560)

## Summary

Over the course of this lab you will look for inefficiencies that are causing sub-optimal performance.

The tables in this lab use the TPCH database.

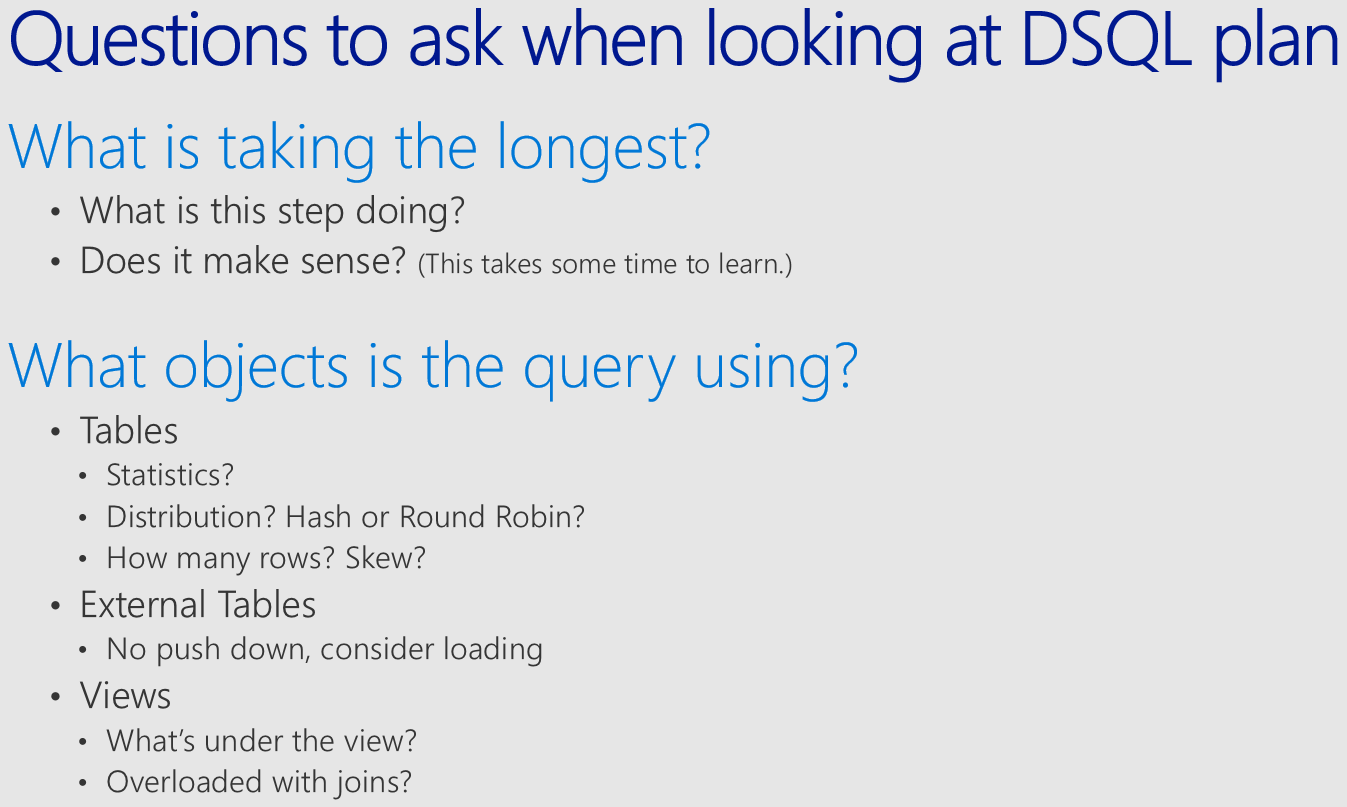
In each of the examples, there will be two versions of same query. One that is running a poor performing scenario (denoted as slow) and one that is running a more optimal scenario (denoted fast). Your goal should be to study the slow scenario using the DMV queries and SSMS object explorer to decide what is wrong with the slow scenario. Once you feel you understand the issue in the ‘slow’ scenario, connect to the ‘fast’ version to verify your findings.

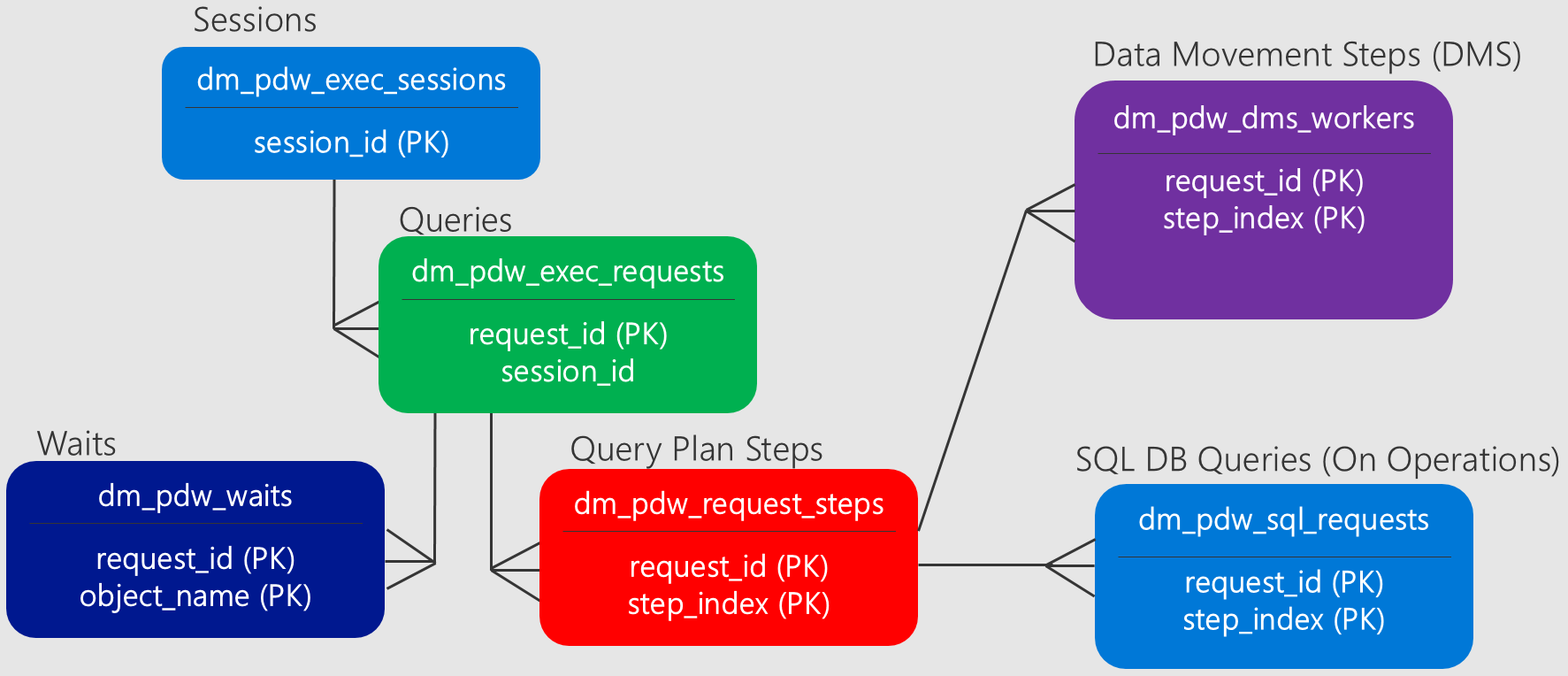
##### SQL DW Resources

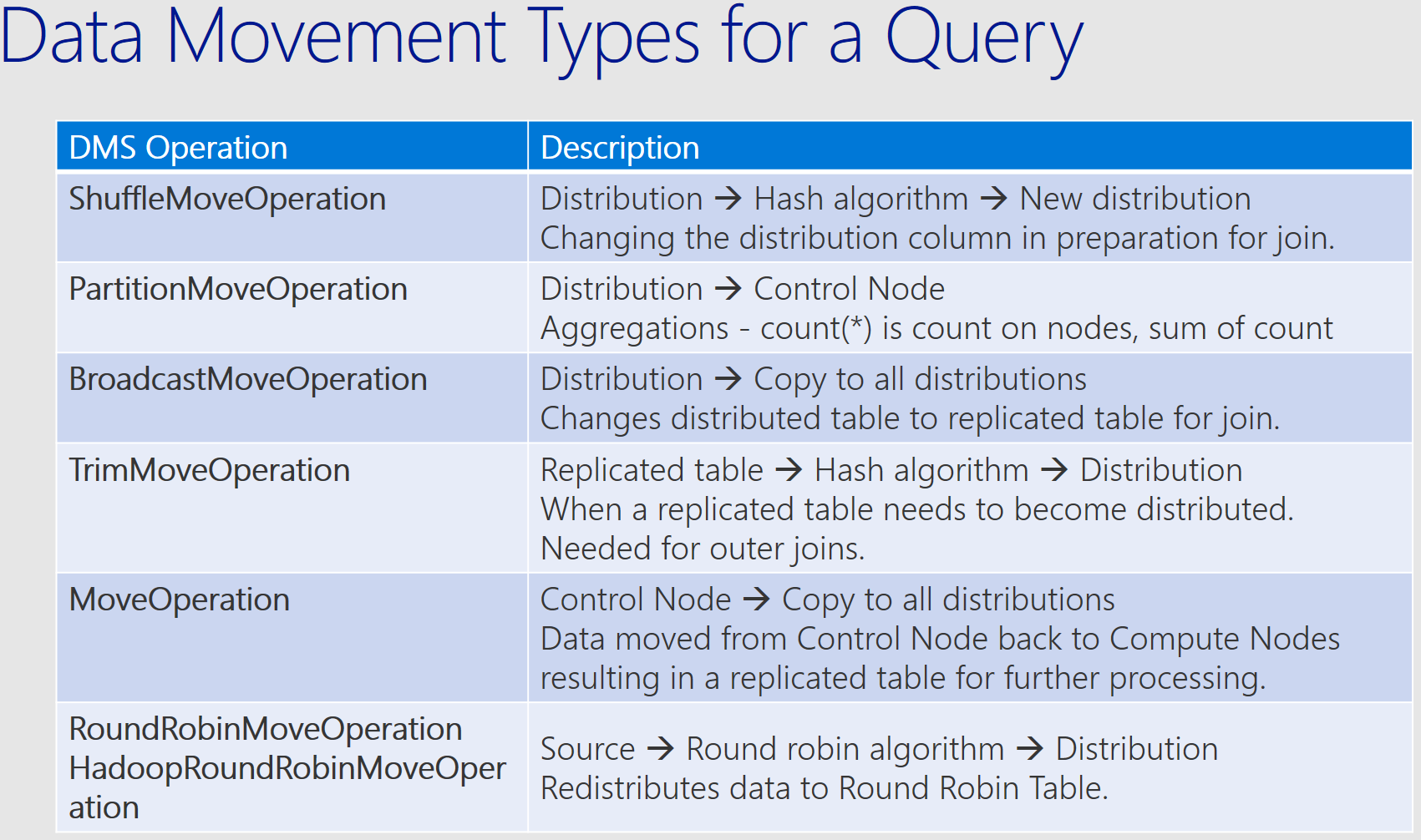
These articles to help you solve the scenarios presented in this lab.

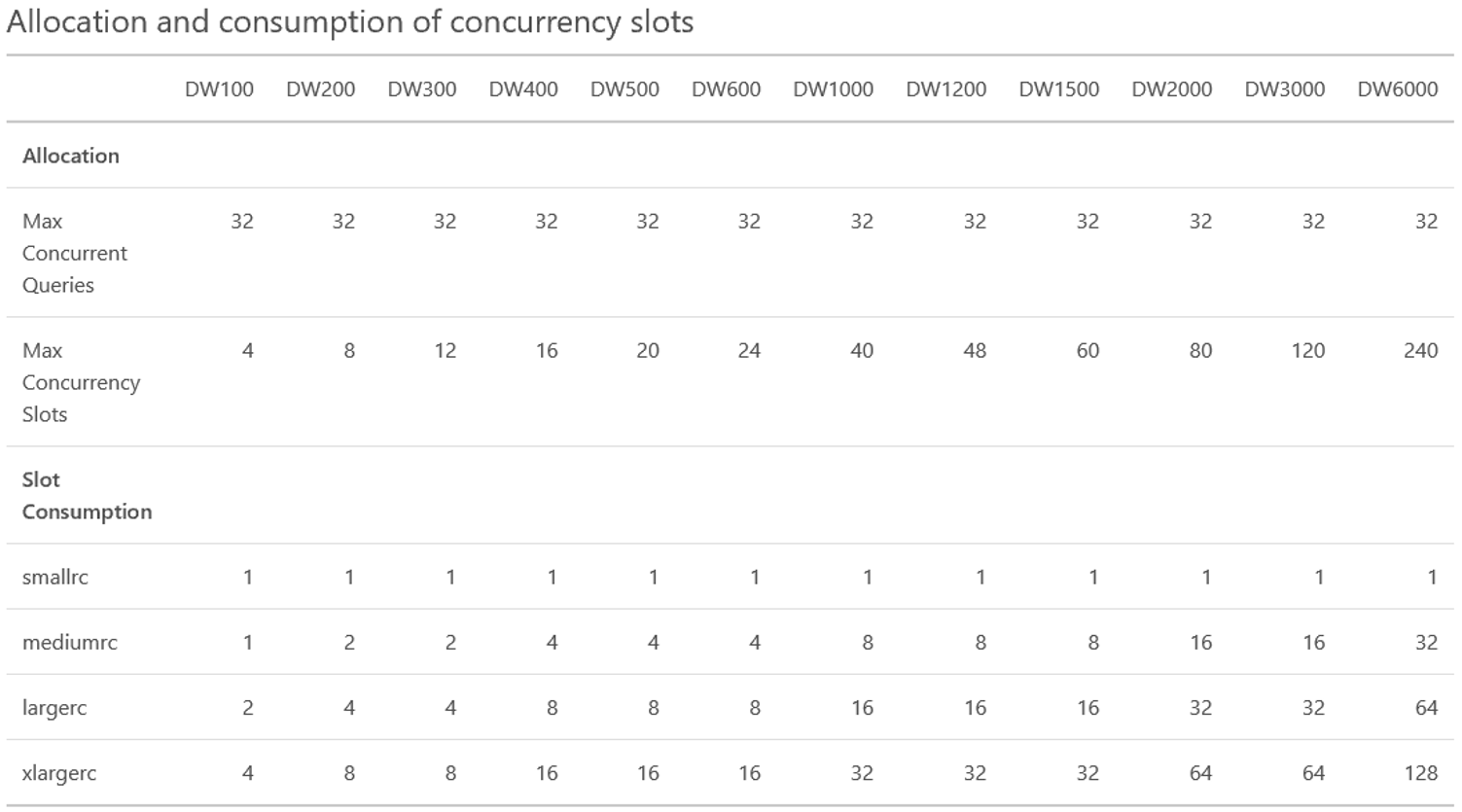
* Query investigation: <https://azure.microsoft.com/en-us/documentation/articles/sql-data-warehouse-manage-monitor/>
* Best Practices: <https://azure.microsoft.com/en-us/documentation/articles/sql-data-warehouse-best-practices/>

The following is selected from the presentation that may help you complete this lab.









|  |  |  |
| --- | --- | --- |
| Pre-requisites: What you need to get started | | |
| **Scenario** | | |
| This lab requires you to load a new data source to the Azure Data Warehouse server created in the previous labs. Please follow below steps to load the sample dataset to your server.  The login that you use for running this script should have “Create Login” permission on your server!  This script will create multiple versions of customer, orders, lineitem, part, partsupp, supplier, nation and region tables. These tables will be used during your lab.  You will also edit the PowerShell script and add your server and database names. This will be used during exercises. | | |
| **Commentary / Notes** | **Steps** | **Screenshots** |
| Loading new data to your server | 1. Open a PowerShell window. 2. Change directory to Query Performance Tuning lab content folder. 3. Change directory to Prep sub folder. 4. Run PrepLab.ps1 script with you Azure Data Warehouse details. |  |
| Change exercise script to add your server and database names | 1. Open your Query Performance Tuning lab content folder. 2. Change directory to Lab sub folder. 3. Edit “RunExercise.ps1” script. 4. Replace “<your\_server>” with your server name. (Without database.windows.net) |  |
|  | 1. Replace “<your\_database> with your database name. |  |
|  | End of pre-requisites. |  |

|  |  |  |
| --- | --- | --- |
| Part 1: Queries running slowly | | |
| **Scenario** | | |
| Your user comes to you and says “My query is running slow. I’m not sure why, because I’m not selecting very many rows. Can you help me figure out why? | | |
| **Commentary / Notes** | **Steps** | **Screenshots** |
| Run “Slow” version | 1. Open a PowerShell window. 2. Change directory to Query Performance Tuning lab content folder. 3. Change directory to Lab sub folder. 4. Run “RunExercise.ps1” script with following parameters   “.\RunExercise.ps1 -Name Exercise1 -Type Slow” |  |
|  | 1. This will execute a query on your server and show the result |  |
|  | 1. Open **Query editor** of SQL Data Warehouse in Azure Portal. 2. Check the query execution details with using DMVs. 3. You can use the labels to search for your specific query. Powershell window shows the “Label” that was used during query execution. | SELECT \* FROM sys.dm\_pdw\_exec\_requests  WHERE [Label] like 'Exercise1 | Slow%'  ORDER BY submit\_time DESC |
|  | 1. Look for most recent execution of Exercise 1 query (“Running” or “Completed”) |  |
|  | 1. Once you’ve identified the problematic query ID for this scenario, take a deeper look into it by using dm\_pdw\_request\_steps: | SELECT \* FROM sys.dm\_pdw\_request\_steps  WHERE request\_id = 'QID####'  ORDER BY step\_index |
|  | 1. After running these queries, come up with a hypothesis about why the operation may be taking a long time. What are the longest running operations? What might they tell you about how the workflow is structured? | **Hint:** *Moving data will typically be one of the most expensive operations within SQL DW. How is this query moving data and how could it move it more effectively? See the Appendix for more info about data movement types.*  **Hint:** *Why is this query moving data? What is the source of the moved data?*  **Hint:** *If you’re still stuck, look at your tables in the object explorer (or sys.tables) – what’s different about the table the user is querying? Can you find it where you expect in the object explorer tree? Why not? What type of table is this table?* |
| Run “Fast” version | 1. After you have a firm understanding of this slow scenario, run the same query with Fast option.   “.\RunExercise.ps1 -Name Exercise1 -Type Fast” |  |
|  | 1. This will execute a query on your server and show the result |  |
|  | 1. Compare the results. | SELECT \* FROM sys.dm\_pdw\_exec\_requests  WHERE [Label] like 'Exercise1 | Fast%'  ORDER BY submit\_time DESC  SELECT \* FROM sys.dm\_pdw\_request\_steps  WHERE request\_id = 'QID####'  ORDER BY step\_index |
| Discussion After steps 1-5, you should have taken a look at the query that was being executed and think about what kind of plan you may expect to see.  At step 10, you should have been able to find your query in exec\_requests using the label and determine: Request\_ID, total elapsed time, resource class, and first 4,000 characters of the command text. Looking at the command you can see that you are running a select count\_big(\*) against the 'dbo.lineitem\_0' table. You may want to look in object explorer or sys.tables at this point to see a few general things: what type of table is it? If it's distributed, what is the distribution key/column? Are there multi column statistics? Have they been updated? If it's a view, what are the underlying tables? These are all general things you will want to know for most query performance issues.    At step 11, you want to pick out the long-running step based on total\_elapsed\_time, which in this case was a HadoopRoundRobinOperation. If you look up this movement, it is querying data in an external table and storing the results in PDW in a round robin table. Also notice that the row count is large at about 60 million rows. This is because when copying data from an external table we copy a full version of the table into SQLDW - the predicate is not used to trim this data down yet.  At step 14, you should be able to use exec\_requests in the same way you did before to get the query text and see that we are now querying table 'dbo.lineitem\_3'. You should look at sys.tables or object explorer in SSMS/SSDT to see the differences between this table and the previous one. You should notice that lineitem\_3 is a distributed table whereas Lineitem\_2 was an external table. The new plan in request\_steps no longer has the Hadoop shuffle because the data is already in SQLDW.    This should illustrate that when you are going to be using external tables repeatedly, it is much more efficient if you first import the table(s) into SQLDW then run the queries on the local tables. Otherwise, every query that touches the external table will have to import the table into tempdb as part of the plan before being able to execute the query. | | |
| End of Exercise 1. | | |

|  |  |  |
| --- | --- | --- |
| Part 2: Joins | | |
| **Scenario** | | |
| Now that you’ve got the hang of things, let’s try the same process on the next exercise.  Again, your user comes to you with questions, complaining that they are joining two of their most important tables together, and SQL DW just isn’t performing as well as they had expected. | | |
| **Commentary / Notes** | **Steps** | **Screenshots** |
| Run “Slow” version | 1. Open a PowerShell window. 2. Change directory to Query Performance Tuning lab content folder. 3. Change directory to Lab sub folder. 4. Run “RunExercise.ps1” script with following parameters   “.\RunExercise.ps1 -Name Exercise2 -Type Slow” |  |
|  | 1. Open **Query editor** of SQL Data Warehouse in Azure Portal. 2. Check the query execution details with using DMVs. 3. You can use the labels to search for your specific query. Powershell window shows the “Label” that was used during query execution. |  |
|  | 1. Look for most recent execution of Exercise 2 query (“Running” or “Completed”) |  |
|  | 1. Once you’ve identified the problematic query ID for this scenario, take a deeper look into it by using dm\_pdw\_request\_steps: | Some steps of the DSQL plan are mostly overhead and can generally be ignored for purposes of optimizing the plan. These steps include the RandomIDOperation and the creation of the temporary tables for DMS. It can often help to add additional predicates to the above query to remove some of the overhead steps thus allowing you to focus on the heavy lifting operations. AND operation\_type NOT IN ('RandomIDOperation') AND command NOT LIKE 'CREATE %' AND command NOT LIKE 'DROP %' |
|  | 1. Check the steps and determine which one(s) might be the problematic steps. |  |
| Run “Fast” version | 1. Run the same query with Fast option.   “.\RunExercise.ps1 -Name Exercise2 -Type Fast” |  |
|  | 1. Compare the 2 query execution plans and determine what would be the reason for query slowness. | **Hint:** *Look at the tables that are being joined with the query. Take a look at the table distribution types in the SSMS object explorer. The icon for each table tells you if the table is hash distributed or round robin distributed? What occurs when two round robin tables are joined?*  Graphical show plan |
| Discussion You should have followed the same workflow as you did in exercise 1 up until step 9. In Step 9 we introduce some helpful additions to the query to filter out some of the SQL OnOperations that are not helpful for troubleshooting. This makes request\_steps easier to read.  At step 10, you can see that we are performing 5 broadcast moves and 1 shuffle move. Most of the time was spent in the shuffle move, but the large rowcount on the first broadcast is a point of interest, because remember we generally do not want to be broadcasting large tables.  At step 12, you are comparing the fast plan to the slow plan. You can see in the fast plan that we now have 4 broadcast moves (instead of 5) and 1 shuffle. The table that is no longer being broadcasted is that large table we noticed in step 10. We can get the tables being queries from exec\_requests, then look at sys.tables or object explorer to see what kind of tables they are. You will see that the fast version has all hash distributed tables, while the slow version has round robin tables.  In general, you want large fact tables to be distributed tables. In this query both the orders and lineitem tables are large fact tables. If we are joining them together then it is best if they are distribution-compatible, which means distributed on the same key. This way each distribution has just a small slice of data to work with. In the fast version, both of these tables are distributed on orderkey. Round robin tables are never distribution-compatible, so the slow plan has to perform some sort of movement, like a broadcast, to make them distribution compatible before performing the join. The fast version shuffle will be faster because of the smaller input data volume. | | |
| End of Exercise 2. | | |

|  |  |  |
| --- | --- | --- |
| Part 3: Troubleshooting | | |
| **Scenario** | | |
| Again, your user comes to you with questions, saying “I’ve just loaded my data and my queries are running slow than on SQL Server! What am I missing here?” | | |
| **Commentary / Notes** | **Steps** | **Screenshots** |
| Run “Slow” version | 1. Open a PowerShell window. 2. Change directory to Query Performance Tuning lab content folder. 3. Change directory to Lab sub folder. 4. Run “RunExercise.ps1” script with following parameters   “.\RunExercise.ps1 -Name Exercise3 -Type Slow” |  |
|  | 1. Open Query editor of SQL Data Warehouse in Azure Portal. 2. Check the query execution details with using DMVs. 3. You can use the labels to search for your specific query. Powershell window shows the “Label” that was used during query execution. |  |
|  | 1. Look for most recent execution of Exercise 3 query (“Running” or “Completed”) |  |
|  | 1. Once you’ve identified the problematic query ID for this scenario, take a deeper look into it by using dm\_pdw\_request\_steps: |  |
|  | 1. Check the steps and determine which one(s) might be the problematic steps. |  |
| Run “Fast” version | 1. Run the same query with Fast option.   “.\RunExercise.ps1 -Name Exercise3 -Type Fast” |  |
|  | 1. Compare the 2 query execution plans and determine what would be the reason for query slowness. | **Hint:** *Look at our best practices (in order) to narrow down what issues cause queries to run slowly.*  **Hint:** *The “orders” table is one of the two largest tables and generally too big for a broadcast move. Why did the optimizer choose to create a copy of these rows on all nodes?*  **Hint:** *If you’re still having trouble, look at the orders table in the object explorer tree to see what statistics are on the tables in this query. You can see the SQL that is running in the DMV sys.dm\_pdw\_exec\_requests.* |
| Discussion You should be able to reach step 10 using the same method you did in the first 2 exercises. In step 10, you should look through the steps for the longest-running step. In this case it's the BroadcastMove that took over 30 seconds. Next you can see that the rowcount for this broadcast is 60 million rows. This is a red flag because broadcasting such a large table will be very expensive. You should have used the original query or the exec\_requests DMV to get the command and see what tables you are querying.  In step 12, you can see that this large broadcast is no longer there. If you compare the table types in the slow version to the tables in the fast version you will see that they are the same. However in the fast version you can see in object explorer that there are statistics on the distribution and join columns.  Further, if you run the query provided to get details about the table, you will see that for lineitem\_1, the CTL\_row\_count is 1,000, but the cmp\_row\_count is ~60 million. 1,000 is the default value for statistics on the control node, so this means that statistics were never manually created. The distributed plan was created assuming it could broadcast this table because there were only 1,000 rows, but in reality there were 60 million rows, which caused our long-running step.  This illustrates how the absence of statistics or statistics not being up to date can affect the distributed plan. | | |
| End of Exercise 3. | | |

|  |  |  |
| --- | --- | --- |
| Part 4: Query performance improvements | | |
| **Scenario** | | |
| Now that your user has got all of their data loaded and organized they are trying out some of their more complex queries. Check this exercise to see if there are any improvements they can make to decrease their query runtime. | | |
| **Commentary / Notes** | **Steps** | **Screenshots** |
| Run “Slow” version | 1. Open a PowerShell window. 2. Change directory to Query Performance Tuning lab content folder. 3. Change directory to Lab sub folder. 4. Run “RunExercise.ps1” script with following parameters   “.\RunExercise.ps1 -Name Exercise4 -Type Slow” |  |
|  | 1. Open Query editor of SQL Data Warehouse in Azure Portal. 2. Check the query execution details with using DMVs. 3. You can use the labels to search for your specific query. Powershell window shows the “Label” that was used during query execution. |  |
|  | 1. Look for most recent execution of Exercise 4 query (“Running” or “Completed”) |  |
|  | 1. Check the steps and determine which one(s) might be the problematic steps. |  |
| Run “Fast” version | 1. Run the same query with Fast option.   “.\RunExercise.ps1 -Name Exercise4 -Type Fast” |  |
|  | 1. Compare the 2 query execution plans and determine what would be the reason for query slowness. | **Hint:** *In this example, the query plan is optimal. This query could benefit if it was given more memory. How much memory has been allocated to this query? How can you use sys.dm\_pdw\_exec\_requests to determine the memory grant? How can you change the memory allocation for a query?* |
| Discussion In this exercise you will follow the same method to get to step 9. This time we see that the plan is a single step plan - a return operation. From a distributed plan perspective this is ideal because no data movement occurred. We simply were able to run the distributed queries on each distribution and return the results.  For a query like this we cannot improve the MPP plan, so the next option to look at is resource class. You should have noticed that exec\_requests shows that the query was running in smallRC. Certain queries will benefit from the larger memory allocation of larger resource classes and this usually requires testing to find the ideal balance of resource class usage for a query vs concurrency.  Once you are at step 11, you should have looked at exec\_requests and noticed that it was now running in LargeRC and the execution time was faster. These test queries are pretty fast running because the execution time is low, but for larger queries this can make a big difference. The default resource class is small. Remember, as you increase resource class you also decrease concurrency, so testing is required.  If you want to change the resource class for a query you would use sp\_addrolemember and sp\_droprolemember | | |
| End of Exercise 4. | | |

|  |  |  |
| --- | --- | --- |
| Part 5: Query analysis | | |
| **Scenario** | | |
| Now that you’ve helped your user with some of their initial issues, they’re beginning to bring some of their analysts onto the system – but some analysts are complaining that their queries are taking very long to run or don’t seem to be running at all. | | |
| **Commentary / Notes** | **Steps** | **Screenshots** |
| Run “Slow” version | 1. Open a PowerShell window. 2. Change directory to Query Performance Tuning lab content folder. 3. Change directory to Lab sub folder. 4. Run “RunExercise.ps1” script with following parameters   “.\RunExercise.ps1 -Name Exercise5 -Type Slow” |  |
|  | 1. This script will create a workload simulation on your server. It will create 20 background jobs which will send queries to your system. 2. It will wait for 60 seconds for all background jobs properly starts and then will start your problematic query. |  |
|  | 1. Open Query editor of SQL Data Warehouse in Azure Portal. 2. Check all the active queries. | SELECT \* FROM sys.dm\_pdw\_exec\_requests  WHERE status not in ('completed', 'failed', 'cancelled') AND session\_id <> session\_id()  ORDER BY request\_id DESC; |
|  | 1. Can you tell what is happening? | **Hint:** *What is the state of this query in sys.dm\_pdw\_exec\_requests? Why?*  **Hint:** *Run this query. What does it tell you about the state of the query?*    **Hint:** What can be changed to ensure these small queries run? After you investigate connect to ‘demo5\_fast’ to see the changes in action. |
|  | 1. You need to kill the background jobs before continuing. 2. Cancel the running process on current PowerShell window. 3. Run “.\Kill.ps1” 4. Make sure you close the PowerShell window. |  |
| Run “Fast” version | 1. Open a PowerShell window. 2. Change directory to Query Performance Tuning lab content folder. 3. Change directory to Lab sub folder. 4. Run “RunExercise.ps1” script with following parameters Run the same query with Fast option.   “.\RunExercise.ps1 -Name Exercise5 -Type Fast” |  |
|  | 1. Compare the 2 query execution plans. | Are the pros/cons of resource classes clear after completing exercise 4 and 5? |
|  | 1. You need to kill the background jobs before continuing. 2. Cancel the running process on current PowerShell window. 3. Run “.\Kill.ps1” 4. Make sure you close the PowerShell window. |  |
| Discussion This exercise tries to simulate an active workload on your data warehouse. It creates 20 background sessions which sends queries constantly.  When you reached to step 9 on Slow execution exercise, you will notice that your query is in the queue (“Suspended”) and does not “Running”. You can check wait stats and see that what is your query is waiting on “UserConcurrencyResourceType” which means that it is waiting for enough concurrency slots become available.    When you check dm\_pdw\_exec\_requests you will notice that this query is running on largerc resource class. In previous example we talk about using higher resource classes allow your query to have more memory resources. But this will result in more memory consumption from the overall system and resulted in less concurrent query executions. So you need to be careful about which resource classes you are using for executing your queries. Always test your queries with your actual workload.  On faster version of this exercise you will notice that your queries might again queued but once there is enough concurrency slots available it will go through the execution. You will see that your query runs 3 times but at every execution it waits on the queue. You can check the queue waiting time by comparing start\_time and submit\_time in dm\_pdw\_exec\_requests DMV. | | |
| End of Exercise 5. | | |

**IMPORTANT: AVOID INCURRING EXTRA CHARGES BY PAUSING YOUR SUBSCRIPTION RESOURCES**