Module 4: Workload and Query Optimization

**Overview**

In this module, you’ll explore some of the performance improvements that have been recently added to SQL Data Warehouse, and how to use the new workload importance feature to ensure predictable performance on your most important workloads.

**Pre-requisites:**

* Existing Azure SQL Data Warehouse
* Azure subscription

If you haven’t run through the earlier modules in the series, launch PowerShell and run the Module4Init.ps1 PowerShell script in the Module 4\Resources folder. This will configure your data warehouse with the pre-requisites needed to complete this lab.

**Load and optimize your staging tables**

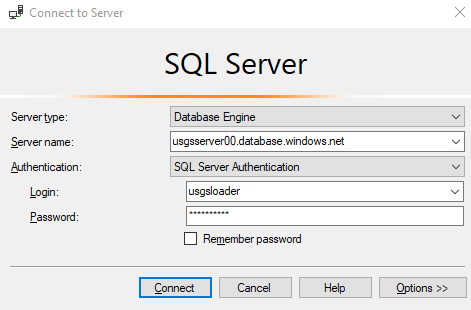
In the previous module, you loaded data into round-robin staging tables. This table distribution optimizes for loading data into SQL Data Warehouse as fast as possible. But this is not optimal for getting query and analytics performance out of these tables. Consider using alternate distribution methods paired with clustered columnstore indices to make your queries efficient and performant.

The easiest way to do this is by using the ‘Create Table AS Select’ (CTAS) statement to create new tables from your staging tables – this allows you to change the table distribution and also build clustered columnstore indices. There are a few options for distributing your data warehouse tables:

* Replicating a table duplicates it across all storage nodes and is best for small dimension tables that will be joined frequently.
* Hash distributed tables are distributed according to the specified distribution column, and significantly boost performance on joins and aggregations against that column. When choosing a candidate for a distribution column, look for the column you’ll be joining or aggregating on, and one that has many unique values. For these labs, we’ll use the [FipsCountyCode] column.
* Round robin tables are still a great choice for larger tables where there is no clear distribution key, or where a column distribution could create uneven data skew.

Optimize staging tables:

1. Launch the Object Explorer and connect with the credentials provided below – making sure to replace ‘##’ with your participant number:  
   Server name: usgsserver##.database.windows.net  
   Authentication: SQL Server Authentication  
   Username: usgsloader  
   Password: P@ssword##



1. Run the following script. Once you create the new tables, notice the different icons used for distributed, round robin, and replicated tables.

--Weather Data

--factWeatherMeasurements

CREATE TABLE [prod].[factWeatherMeasurements]

WITH

(

CLUSTERED COLUMNSTORE INDEX,

DISTRIBUTION = HASH(fipscountycode)

)

AS SELECT \*, CONVERT(UNIQUEIDENTIFIER,0x0) AS ELTID FROM [STG].[factWeatherMeasurements]

--dimWeatherObservationTypes

CREATE TABLE [prod].[dimWeatherObservationTypes]

WITH

(

CLUSTERED COLUMNSTORE INDEX,

DISTRIBUTION = REPLICATE

)

AS SELECT \*, CONVERT(UNIQUEIDENTIFIER,0x0) AS ELTID FROM [STG].[dimWeatherObservationTypes];

--dimUSFIPSCodes

CREATE TABLE [prod].[dimUSFIPSCodes]

WITH

(

CLUSTERED COLUMNSTORE INDEX,

DISTRIBUTION = REPLICATE

)

AS SELECT \*, CONVERT(UNIQUEIDENTIFIER,0x0) AS ELTID FROM [STG].[dimUSFIPSCodes];

--dimWeatherObservationSites

CREATE TABLE [prod].[dimWeatherObservationSites]

WITH

(

CLUSTERED COLUMNSTORE INDEX,

DISTRIBUTION = REPLICATE

)

AS SELECT \*, CONVERT(UNIQUEIDENTIFIER,0x0) AS ELTID FROM [STG].[dimWeatherObservationSites];

/\*----fireevents----\*/

----dimOrganizationCode

CREATE TABLE [prod].[dimOrganizationCode]

WITH

(

CLUSTERED COLUMNSTORE INDEX,

DISTRIBUTION = REPLICATE

)

AS SELECT \*, CONVERT(UNIQUEIDENTIFIER,0x0) AS ELTID FROM [STG].[dimOrganizationCode];

--dimAreaProtectionCategory

CREATE TABLE [prod].[dimAreaProtectionCategory]

WITH

(

CLUSTERED COLUMNSTORE INDEX,

DISTRIBUTION = REPLICATE

)

AS SELECT \*, CONVERT(UNIQUEIDENTIFIER,0x0) AS ELTID FROM [STG].[dimAreaProtectionCategory];

1. Now, you will test the performance difference between queries running on external tables vs queries running on distributed tables in SQL Data Warehouse. Before you run the queries below, look at the execution plans by hitting CTRL+L and notice how the hash-distributed and replicated tables create simpler execution plans – this is because these tables help minimize data movement.

--Multi-join query over hash-distributed and replicated tables

SELECT TOP(1000)

m.FipsCountyCode,

s.NWSRegion,

t.ObservationTypeName,

AVG(m.ObservationValue) AS AvgObservationValue

FROM prod.factWeatherMeasurements AS m

JOIN prod.dimWeatherObservationSites AS s

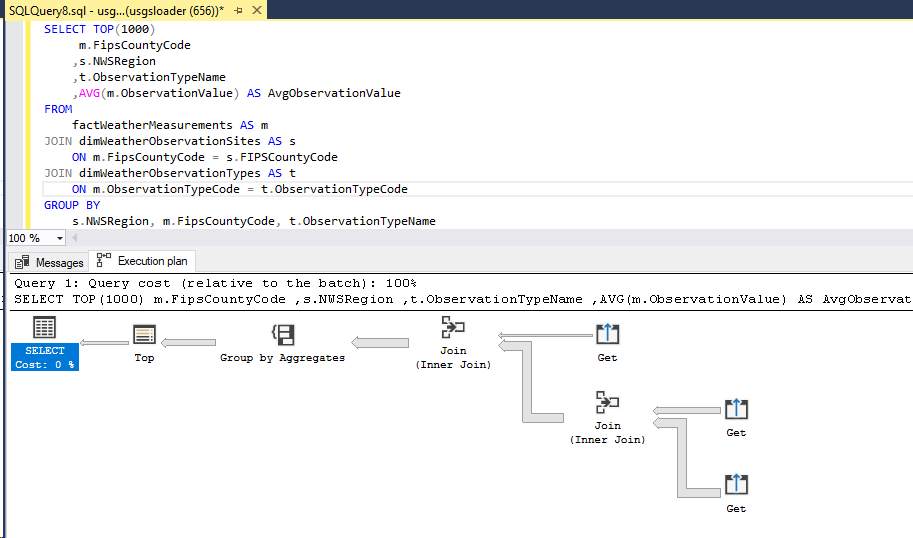
ON m.FipsCountyCode = s.FIPSCountyCode

JOIN prod.dimWeatherObservationTypes AS t

ON m.ObservationTypeCode = t.ObservationTypeCode

GROUP BY s.NWSRegion, m.FipsCountyCode, t.ObservationTypeName

ORDER BY t.ObservationTypeName ASC, AvgObservationValue DESC



--Multi-join query over external staging tables

SELECT TOP(1000)

m.FipsCountyCode,

s.NWSRegion,

t.ObservationTypeName,

AVG(m.ObservationValue) AS AvgObservationValue

FROM ext.factWeatherMeasurements AS m

JOIN ext.dimWeatherObservationSites AS s

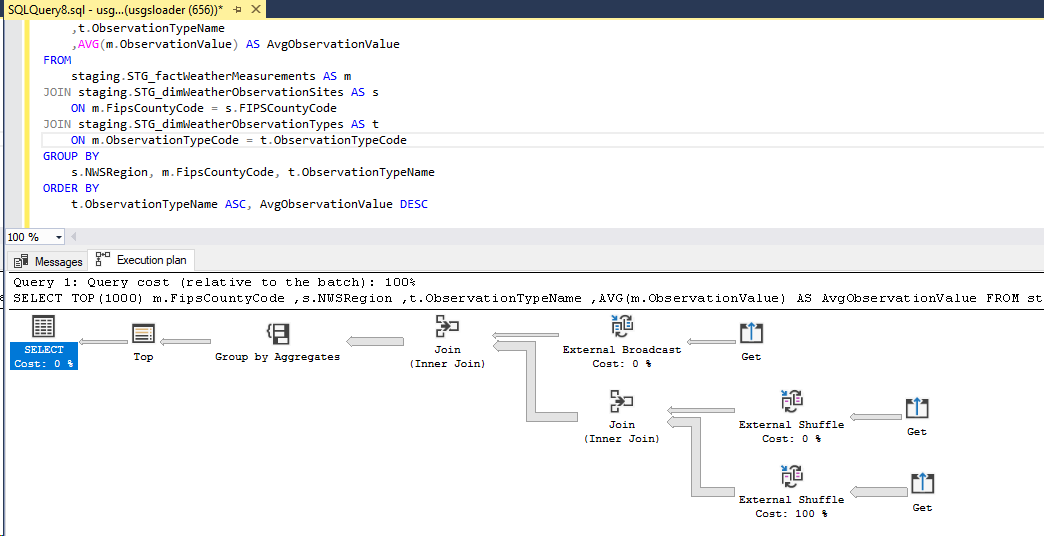
ON m.FipsCountyCode = s.FIPSCountyCode

JOIN ext.dimWeatherObservationTypes AS t

ON m.ObservationTypeCode = t.ObservationTypeCode

GROUP BY s.NWSRegion, m.FipsCountyCode, t.ObservationTypeName

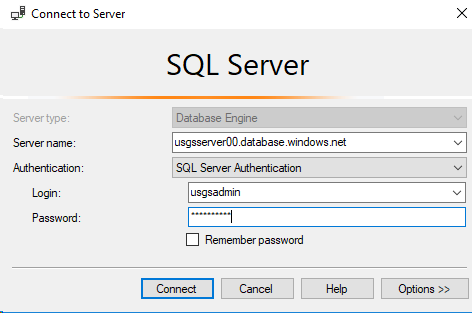
ORDER BY t.ObservationTypeName ASC, AvgObservationValue DESC



**Workload Importance and Classifiers**

When your SQL Data Warehouse is fully utilized, any new queries are automatically queued and granted resources in a first-in, first-out basis – the first query in the queue is the next to get available resources. However, not all queries are created equal, and sometimes you may want a higher priority query to get access to resources faster. Workload importance allows you to grant users and workloads higher (or lower) priority, ensuring they get resources as quickly as possible.

In SQL Server Management Studio (SSMS) as usgsadmin:

1. Connect the Object Explorer with the credentials provided below – making sure to replace ‘## with your participant number:  
   Server name: usgsserver##.database.windows.net  
   Authentication: SQL Server Authentication  
   Username: usgsadmin  
   Password: P@ssword##  
   
2. Create two classifiers as the admin users – classifiers allow you to assign importance and resource groups to different users.

--Normal importance for Employee

CREATE WORKLOAD CLASSIFIER ForestryEmployee

WITH

(WORKLOAD\_GROUP = 'smallrc'

,MEMBERNAME = 'CaliforniaForestryEmployee'

,IMPORTANCE = NORMAL);

--High importance for Manager

CREATE WORKLOAD CLASSIFIER ForestryManager

WITH

(WORKLOAD\_GROUP = 'smallrc'

,MEMBERNAME = 'CaliforniaForestryManager'

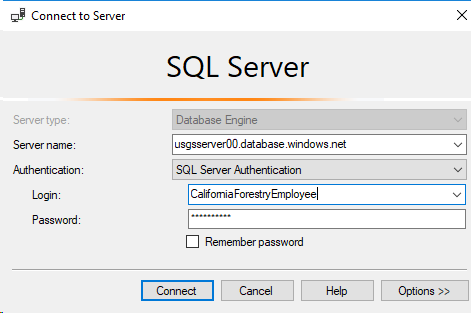
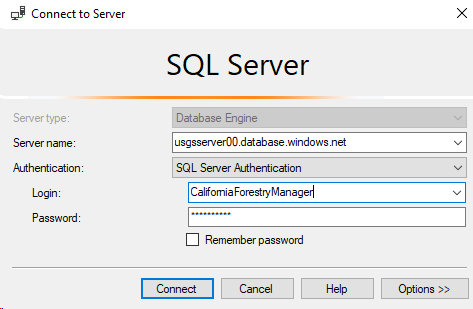
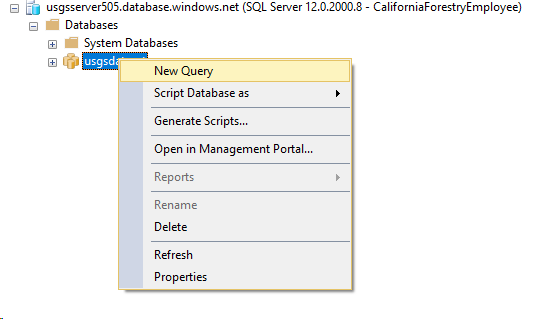
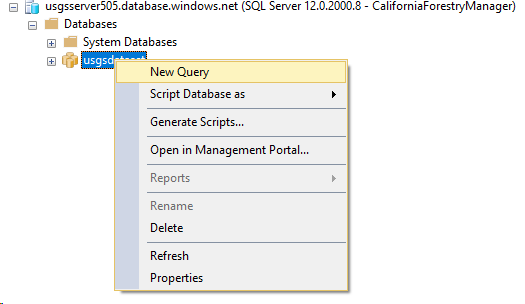
,IMPORTANCE = HIGH);

Simulate a busy system prioritizing queries by using workload importance:   
  
The easiest way to test workload importance yourself is to look for differences in the submission time of a query (when it enters the queue) and the start time of a query (when it’s granted resources and starts running).

With ‘normal’ importance across all queries, the order of the submission times and start times will be the same since it’s a FIFO queue.

However, a query with ‘high’ importance may have a later submission time but an earlier start time than less important queries because it can jump the queue.

In SQL Server Management Studio (SSMS):

1. Connect to the SQL Data Warehouse using the ‘CaliforniaForestryEmployee’ credentials provided below – making sure to replace ‘##’ with your participant number:  
   Server name: usgsserver##.database.windows.net  
   Authentication: SQL Server Authentication  
   Username: CaliforniaForestryEmployee  
   Password: P@ssword##  
   
2. Open another connection to the SQL Data Warehouse using the ‘CaliforniaForestryManager’ credentials provided below – making sure to replace ‘##’ with your participant number:  
   Server name: usgsserver##.database.windows.net  
   Authentication: SQL Server Authentication  
   Username: CaliforniaForestryManager  
   Password: P@ssword##  
   
3. Right-click on the ‘usgsdataset’ data warehouse and create 5 new query windows using the CaliforniaForestryEmployee connection.  
   
4. Right-click on the ‘usgsdaset’ data warehouse and create 1 new query window using the CaliforniaForestryManager connection.   
   
5. To simulate multiple active queries on the system, you’ll submit the same query to each of the active windows you have open.   
   Submit the employee queries first, and the manager query last:

-- Yearly temperature averages and precipitation per state for years between 2004 and 2019

SELECT StateName,

StatePostalCode,

Year(ObservationDate) AS MeasurementYear,

AVG(CASE ObservationTypeCode WHEN 'TAVG' THEN ObservationValueCorrected END) AS AvgTemperature\_degCelsius,

AVG(CASE ObservationTypeCode WHEN 'TMIN' THEN ObservationValueCorrected END) AS AvgMinTemperature\_degCelsius,

AVG(CASE ObservationTypeCode WHEN 'TMAX' THEN ObservationValueCorrected END) AS AvgMaxTemperature\_degCelsius,

AVG(CASE ObservationTypeCode WHEN 'PRCP' THEN ObservationValueCorrected END) AS AvgPrecipitation\_mm

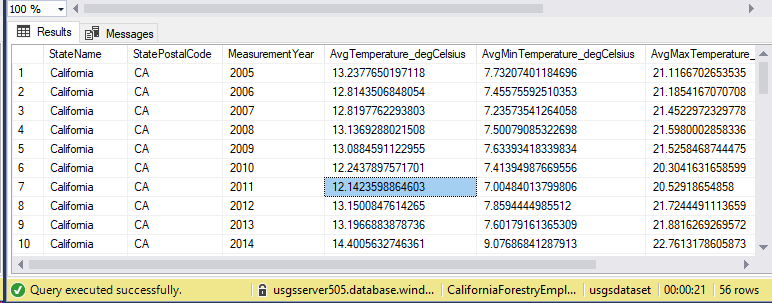
FROM prod.factWeatherMeasurements weather

JOIN prod.dimusfipscodes fips on weather.fipscountycode = fips.fipscode

WHERE Year(ObservationDate) > 2004 AND Year(ObservationDate) < 2019

GROUP BY StateName, StatePostalCode, Year(ObservationDate)

ORDER BY StatePostalCode, MeasurementYear



1. Open a new query window using the ‘usgsadmin’ login. Run the query below to analyze the order in which the queries were handled by SQL DW. Notice how long the ForestryManager’s query has to wait in the queue before being worked on. That’s workload importance!

--View query importance and request information

SELECT DISTINCT a.request\_id,

a.session\_id,

b.login\_name,

a.resource\_class,

a.status,

b.app\_name,

a.submit\_time,

a.start\_time,

a.end\_time,

DATEDIFF(s,a.submit\_time,a.start\_time) AS TimeInQueue\_secs,

a.total\_elapsed\_time/1000.0 AS TotalElapsedTime\_secs,

a.total\_elapsed\_time/60000.0 AS TotalElapsedTime\_mins,

a.command

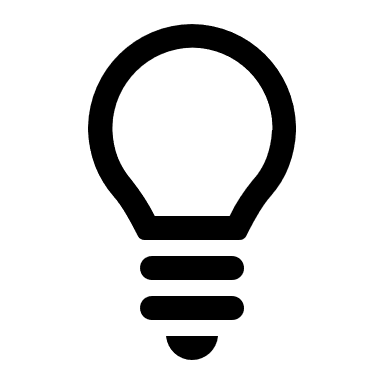
FROM sys.dm\_pdw\_exec\_requests a

JOIN sys.dm\_pdw\_exec\_sessions b ON b.session\_id = a.session\_id

LEFT OUTER JOIN sys.dm\_pdw\_waits c on c.request\_id = a.request\_id and c.session\_id = a.session\_id

WHERE b.login\_name IN ('CaliforniaForestryEmployee', 'CaliforniaForestryManager') AND resource\_class IS NOT NULL

ORDER BY submit\_time ASC

**Feedback**

Congratulations! You have completed the lab on workload optimization for SQL DW.   
Was the module helpful? Any improvements you’d suggest?   
[Take this short 1 min survey to share your thoughts with us](https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR90qBpvyfe5Mkaxhz8QBg_hUMVo2NDNMSDFKWlBRRENaRUc3UDRKME9CTCQlQCN0PWcu).