|  |  |
| --- | --- |
|  |  |
|  | SQL Server 2016 CTP2 Query Store HOL |

Contents

[Overview and setup 3](#_Toc421782745)

[Explore the Query Store user interface in SSMS 5](#_Toc421782746)

[Explore Query Store database settings 9](#_Toc421782747)

[Explore Query Store data in TSQL 10](#_Toc421782748)

[Terms of use 13](#_Toc421782749)

|  |
| --- |
| Overview and setup |

### Estimated time to complete lab is 15-25 minutes.

### Overview

Query Store is a new SQL Server component that captures queries, query plans, runtime statistics, etc. in a persistent store inside the user database. You can think of it as a flight recorder, or black box, for your database. It can also enforce policies to direct the SQL Server query processor to compile some queries to be executed in a specific manner, such as forcing plans. It is a database-scoped persistent store of query workload history. Query Store is primarily targeting administrative scenarios for performance troubleshooting, identifying regressed workloads, or identifying issues with upgrades.

Query Store collects query texts and all relevant properties, as well as query plan choices and performance metrics. This collection process works across restarts or upgrades of the server and across recompilation of indexes. It provides many configurable options for customization. Query Store integrates with existing query execution statistics, plan forcing, and manageability tools.

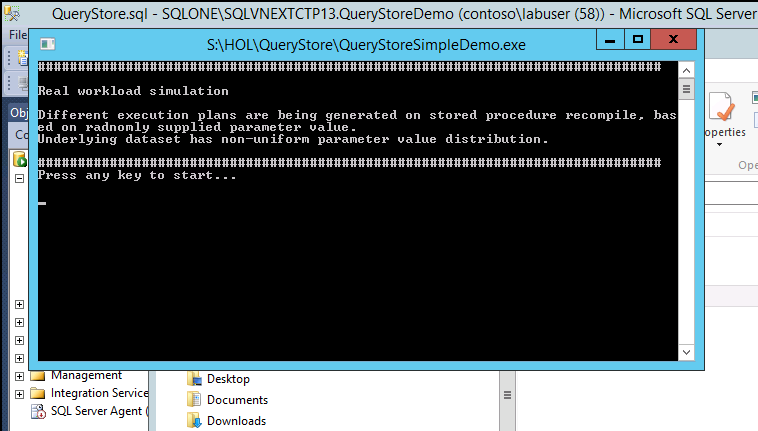
This hands on lab will familiarize you with Query Store and show you how it can be used for troubleshooting and performance tuning. In particular, you will learn:

1. How to work with the Query Store user interface in SSMS;
2. How to identify queries based on several criteria, including those that generate more than one query plan;
3. How to force a query to use a specific plan;
4. How to modify Query Store settings on the database;
5. How to run custom queries based on the Query Store DMVs.

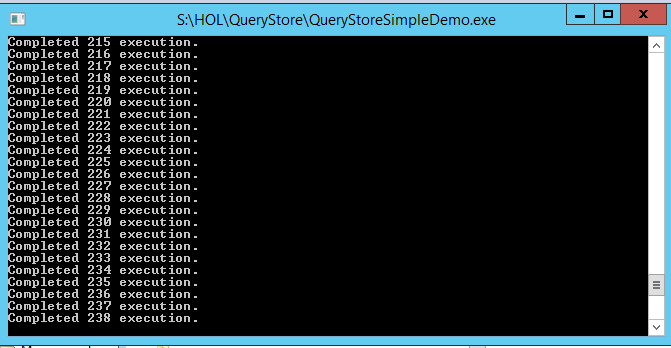
At the end of this lab, you will have worked through some of the most common scenarios involved with Query Store and how you can use this technology to run advanced troubleshooting and performance tuning in SQL Server 2016 CTP2.

### Initial setup

1. During the setup you will need to record credentials and server locations. Open **notepad.exe** to keep track of information
2. Open SQL Server Management Studio on the lab machine.
3. The script file that contains the scripts to be used in this lab are located at **C:\SQL Server 2016 CPT2 HOLs\Query Store\QueryStore.sql**. Open this script now in SSMS.
4. In the **C:\SQL Server 2016 CPT2 HOLs\Query Store** folder, double click on the file called **QueryStoreSimpleDemo.exe**. Once the executable program window opens, click the Enter key to start the workload. Let it run. It will remain running in the background for much of the rest of the lab.



1. Click **Enter** to run the load generator. The Visual Studio solution for this load generator is located in the **C:\SQL Server 2016 CPT2 HOLs\Query Store\QueryStoreSimpleDemo\QueryStoreSimpleDemo** folder.



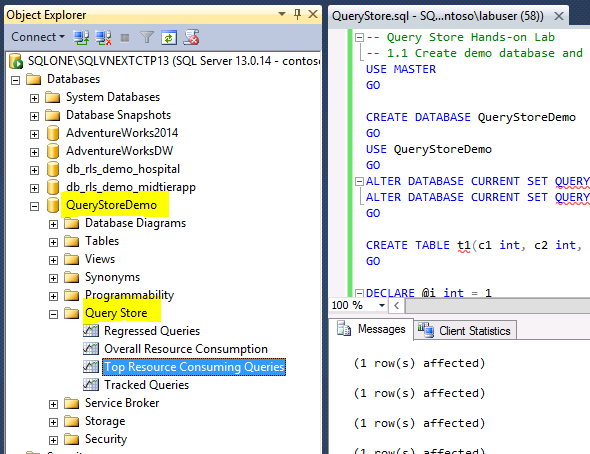
|  |
| --- |
| Explore the Query Store user interface in SSMS |

In this lab, we will explore the Query Store tools in SSMS that help you conduct diagnostics and performance tuning. Let’s get started!

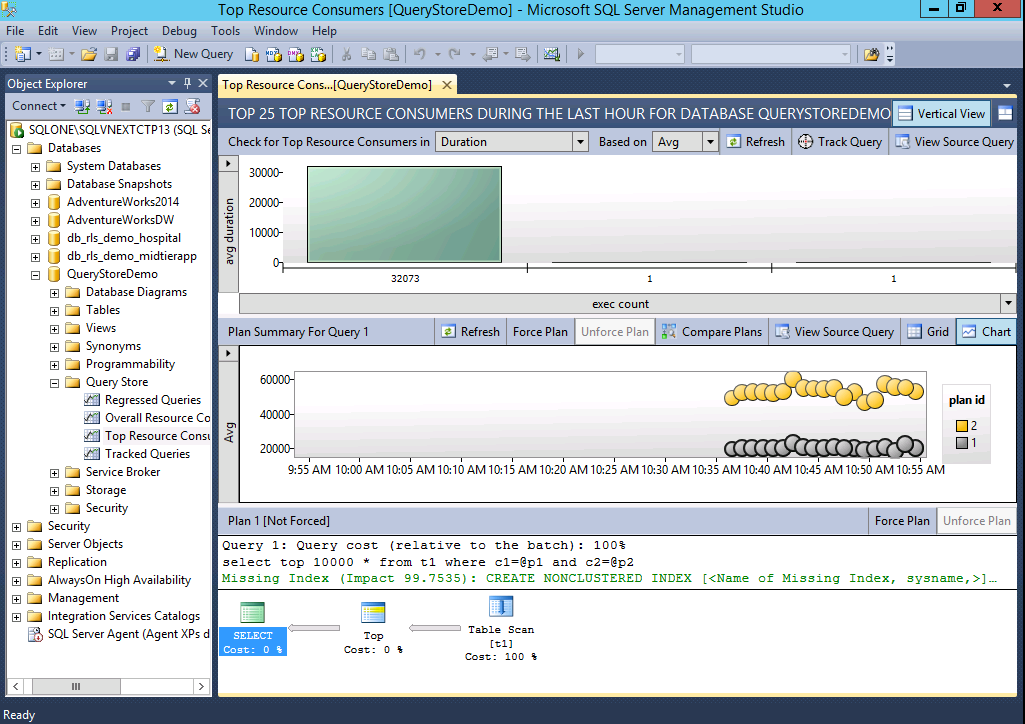
### Explore the Query Store UI in SQL Server Management Studio

Query Store data can be accessed through TSQL from the Query Store system views, but many important uses of the data can also be viewed through the rich toolset exposed in SSMS. We will explore the tools in SSMS first. At the end of this lab, we’ll look at using TSQL and how you can leverage system views in querying this data.

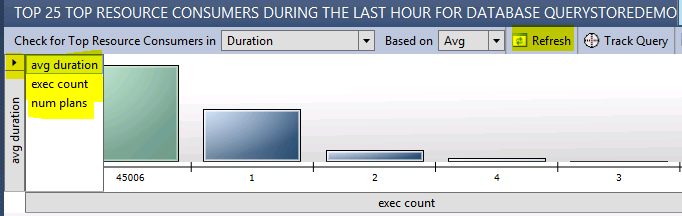
1. Minimize the executing QueryStoreSimpleDemo program, but keep it running.
2. In the object explorer, refresh the database list and expand the new database we just created—QueryStoreDemo. Expand the Query Store node under that database.

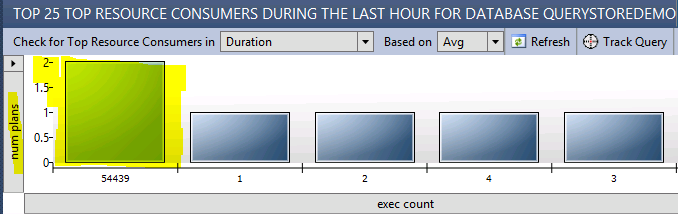


1. Double click on Top Resource Consuming Queries. There are three panes available. The top one allows you to view the top resource consumers based on various criteria. Click the refresh button in the top pane to refresh the view. Select the query that has the biggest impact on resources in the top pane.



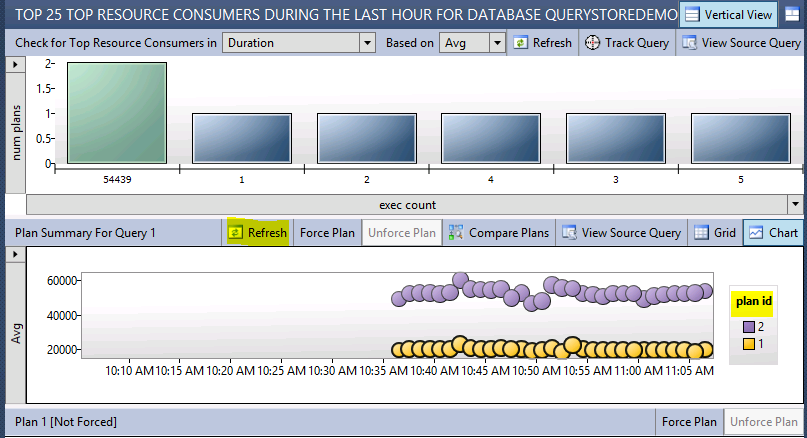
1. Notice that you can hover your mouse over a query in the top pane and see more details about the query ID, average duration, number of execution counts, and number of plans generated. Click refresh. Click on the left button labeled “avg duration” and change to “num plans.”





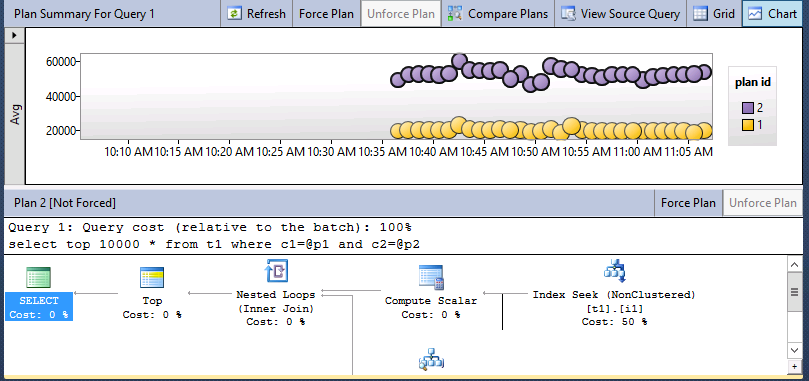
Note that this shows you that while several queries are running in the workload, one query has generated 2 plans. Click on that query that has 2 plans.

1. Look at the middle pane (Plan Summary for Query 1). You should see that two plans are running (with plan ids of 1 and 2) and that SQL Server is constantly alternating between them. Each dot represents an interval of aggregation, based on the interval length, which was set at 1 minute when we created the database. Click refresh in the middle pane to see how this changes over time.



The Query ID and Plan ID are the Query Store internal identifiers.

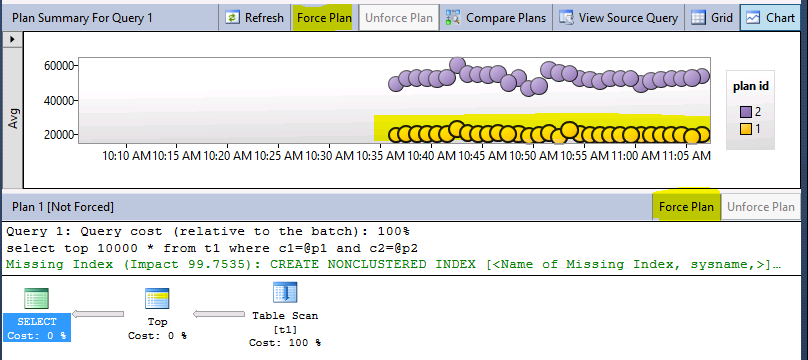
1. In the middle pane, click on the plan on top. It is generating longer average execution times (the values on the left axis represent time in milliseconds). When you click on either plan, you see the execution plan in the bottom pane.



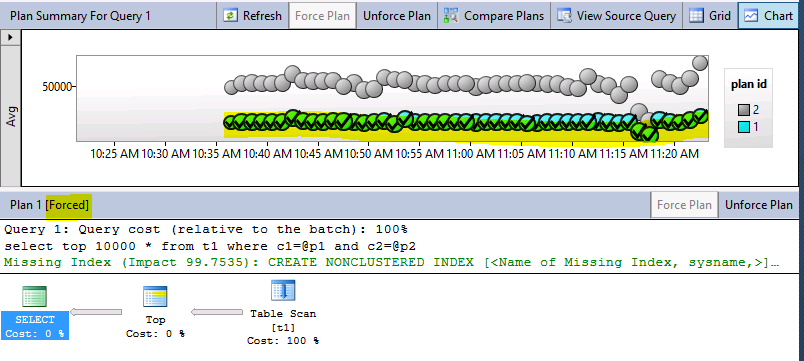
### Force a query to use a certain execution plan

If you look at both the average execution times and the query execution plans for this query, you can see that one plan (the one on top) is taking much longer to run. We can use Query Store to force the query to always run with the better plan.

1. Select the bottom plan in the middle pane to select it. Notice also that hovering your mouse over the plan displays more information about it.



1. Click on Force Plan (in either the middle or bottom pane). And click Yes on the confirmation.



1. The plan now contains check marks in each interval for the forced plan. Every time this query runs now, it will use the forced plan.
2. If you want to later allow SQL Server to choose between plans, you will need to click on the “Unforce Plan” button to the right of the disabled “Force Plan” button.
3. Refresh the pane again to verify.

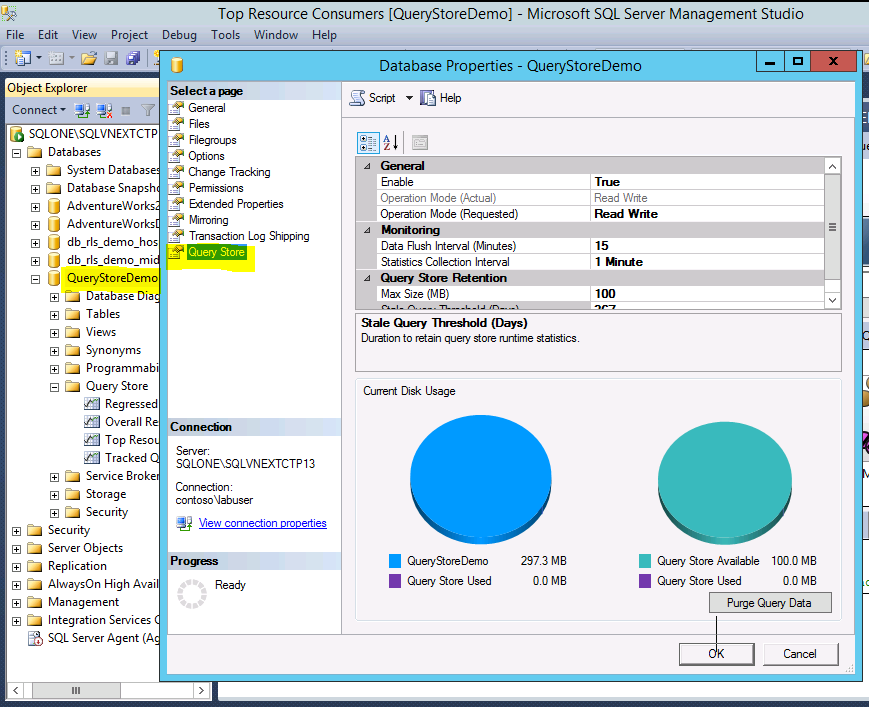
|  |
| --- |
| Explore Query Store database settings |

In this lab we will look at how to adjust various settings at the database level for Query Store.

### Explore database-level settings

Query Store has several settings you can adjust, depending on your needs.

1. In Object Explorer, right click on the database name (QueryStoreDemo) and select Properties. Notice that you can see a properties page for Query Store.



1. Notice that you can adjust several settings here, including:

* Enabling/Disabling Query Store
* Operation Mode (Read Write v. Read only)
* Data Flush Interval
* Statistics Collection Interval
* Max Size in megabytes for Query Store retention, and
* Stale Query Threshold in days.

1. There are also graphics that show you how much the current disk usage is, as well as the ability to purge query data.

|  |
| --- |
| Explore Query Store data in TSQL |

All data that Query Store stores is available through the following system views:

* sys.query\_store\_query\_text
* sys.query\_store\_query
* sys.query\_store\_plan
* sys.query\_context\_settings
* sys.query\_store\_runtime\_stats\_interval
* sys.query\_store\_runtime\_stats
* sys.database\_query\_store\_options

You can use these views in your own queries to get insight about query execution and other matters for a period of time when Query Store was active. In this part of the lab, we’re going to look at some common uses for these custom queries.

Execute each query. They are each in the SQL script file for this lab. Feel free to modify the TSQL to experiment with the results.

### Find last *n* queries that were executed on the database

SELECT TOP 10 qt.query\_sql\_text, q.query\_id, qt.query\_text\_id, p.plan\_id, rs.last\_execution\_time

FROM sys.query\_store\_query\_text qt

JOIN sys.query\_store\_query q ON qt.query\_text\_id = q.query\_text\_id

JOIN sys.query\_store\_plan p ON q.query\_id = p.query\_id JOIN sys.query\_store\_runtime\_stats rs ON p.plan\_id = rs.plan\_id

ORDER BY rs.last\_execution\_time DESC

### Find the count of executions for each query

SELECT q.query\_id, qt.query\_text\_id, qt.query\_sql\_text,

SUM(rs.count\_executions) AS total\_execution\_count

FROM sys.query\_store\_query\_text qt

JOIN sys.query\_store\_query q ON qt.query\_text\_id = q.query\_text\_id

JOIN sys.query\_store\_plan p ON q.query\_id = p.query\_id JOIN sys.query\_store\_runtime\_stats rs ON p.plan\_id = rs.plan\_id

GROUP BY q.query\_id, qt.query\_text\_id, qt.query\_sql\_text

ORDER BY total\_execution\_count DESC

### Find *n* queries with the longest execution time in the last hour

SELECT TOP 10 qt.query\_sql\_text, q.query\_id, qt.query\_text\_id, p.plan\_id,

getutcdate() as CurrentUTCTime, rs.last\_execution\_time, rs.avg\_duration

FROM sys.query\_store\_query\_text qt

JOIN sys.query\_store\_query q ON qt.query\_text\_id = q.query\_text\_id

JOIN sys.query\_store\_plan p ON q.query\_id = p.query\_id JOIN sys.query\_store\_runtime\_stats rs ON p.plan\_id = rs.plan\_id

WHERE rs.last\_execution\_time > dateadd(hour, -1, getutcdate())

ORDER BY rs.avg\_duration DESC

### Find *n* queries that had the biggest average physical IO reads in the last 24 hours, with row counts and execution counts

SELECT TOP 10 qt.query\_sql\_text, q.query\_id, qt.query\_text\_id, p.plan\_id,

rs.runtime\_stats\_id, rsi.start\_time, rsi.end\_time, rs.avg\_physical\_io\_reads,

rs.avg\_rowcount, rs.count\_executions

FROM sys.query\_store\_query\_text qt

JOIN sys.query\_store\_query q ON qt.query\_text\_id = q.query\_text\_id

JOIN sys.query\_store\_plan p ON q.query\_id = p.query\_id JOIN sys.query\_store\_runtime\_stats rs ON p.plan\_id = rs.plan\_id

JOIN sys.query\_store\_runtime\_stats\_interval rsi ON rsi.runtime\_stats\_interval\_id = rs.runtime\_stats\_interval\_id

WHERE rsi.start\_time >= dateadd(hour, -24, getutcdate())

ORDER BY rs.avg\_physical\_io\_reads DESC

### Find queries with multiple plans

WITH Query\_MultPlans

AS

(

SELECT COUNT(\*) AS cnt, q.query\_id

FROM sys.query\_store\_query\_text qt

JOIN sys.query\_store\_query q

ON qt.query\_text\_id = q.query\_text\_id

JOIN sys.query\_store\_plan p

ON p.query\_id = q.query\_id

GROUP BY q.query\_id

HAVING COUNT(DISTINCT plan\_id) > 1

)

SELECT q.query\_id, OBJECT\_NAME(object\_id) AS ContainingObject,

query\_sql\_text, plan\_id, p.query\_plan AS plan\_xml,

p.last\_compile\_start\_time, p.last\_execution\_time

FROM Query\_MultPlans qm

JOIN sys.query\_store\_query q

ON qm.query\_id = q.query\_id

JOIN sys.query\_store\_plan p

ON q.query\_id = p.query\_id

JOIN sys.query\_store\_query\_text qt

ON qt.query\_text\_id = q.query\_text\_id

ORDER BY query\_id, plan\_id

### Find all queries that regressed in performance, where execution time was doubled within the last 48 hours

SELECT qt.query\_sql\_text, q.query\_id, qt.query\_text\_id,

p1.plan\_id AS plan1, rsi1.start\_time AS runtime\_stats\_interval\_1, rs1.runtime\_stats\_id AS runtime\_stats\_id\_1, rs1.avg\_duration AS avg\_duration\_1,

p2.plan\_id AS plan2, rsi2.start\_time AS runtime\_stats\_interval\_2, rs2.runtime\_stats\_id AS runtime\_stats\_id\_2, rs2.avg\_duration AS plan2

FROM sys.query\_store\_query\_text qt

JOIN sys.query\_store\_query q ON qt.query\_text\_id = q.query\_text\_id

JOIN sys.query\_store\_plan p1 ON q.query\_id = p1.query\_id

JOIN sys.query\_store\_runtime\_stats rs1 ON p1.plan\_id = rs1.plan\_id

JOIN sys.query\_store\_runtime\_stats\_interval rsi1 ON rsi1.runtime\_stats\_interval\_id = rs1.runtime\_stats\_interval\_id

JOIN sys.query\_store\_plan p2 ON q.query\_id = p2.query\_id

JOIN sys.query\_store\_runtime\_stats rs2 ON p2.plan\_id = rs2.plan\_id

JOIN sys.query\_store\_runtime\_stats\_interval rsi2 ON rsi2.runtime\_stats\_interval\_id = rs2.runtime\_stats\_interval\_id

WHERE rsi1.start\_time > dateadd(hour, -48, getutcdate()) AND rsi2.start\_time > rsi1.start\_time AND

rs2.avg\_duration > 2\*rs1.avg\_duration

You can now close the lab environment.

|  |
| --- |
| Terms of use |

© 2015 Microsoft Corporation. All rights reserved.

By using this Hands-on Lab, you agree to the following terms:

The technology/functionality described in this Hands-on Lab is provided by Microsoft Corporation in a “sandbox” testing environment for purposes of obtaining your feedback and to provide you with a learning experience. You may only use the Hands-on Lab to evaluate such technology features and functionality and provide feedback to Microsoft.  You may not use it for any other purpose. You may not modify, copy, distribute, transmit, display, perform, reproduce, publish, license, create derivative works from, transfer, or sell this Hands-on Lab or any portion thereof.

COPYING OR REPRODUCTION OF THE HANDS-ON LAB (OR ANY PORTION OF IT) TO ANY OTHER SERVER OR LOCATION FOR FURTHER REPRODUCTION OR REDISTRIBUTION IS EXPRESSLY PROHIBITED.

THIS HANDS-ONLAB PROVIDES CERTAIN SOFTWARE TECHNOLOGY/PRODUCT FEATURES AND FUNCTIONALITY, INCLUDING POTENTIAL NEW FEATURES AND CONCEPTS, IN A SIMULATED ENVIRONMENT WITHOUT COMPLEX SET-UP OR INSTALLATION FOR THE PURPOSE DESCRIBED ABOVE.  THE TECHNOLOGY/CONCEPTS REPRESENTED IN THIS HANDS-ON LAB MAY NOT REPRESENT FULL FEATURE FUNCTIONALITY AND MAY NOT WORK THE WAY A FINAL VERSION MAY WORK.  WE ALSO MAY NOT RELEASE A FINAL VERSION OF SUCH FEATURES OR CONCEPTS.  YOUR EXPERIENCE WITH USING SUCH FEATURES AND FUNCITONALITY IN A PHYSICAL ENVIRONMENT MAY ALSO BE DIFFERENT.

**FEEDBACK**.  If you give feedback about the technology features, functionality and/or concepts described in this Hands-on Lab to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose.  You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software or service that includes the feedback.  You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them.  These rights survive this agreement.

MICROSOFT CORPORATION HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE HANDS-ON LAB , INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT.  MICROSOFT DOES NOT MAKE ANY ASSURANCES OR REPRESENTATIONS WITH REGARD TO THE ACCURACY OF THE RESULTS, OUTPUT THAT DERIVES FROM USE OF THE VIRTUAL LAB, OR SUITABILITY OF THE INFORMATION CONTAINED IN THE VIRTUAL LAB FOR ANY PURPOSE.

DISCLAIMER

This lab contains only a portion of new features and enhancements in Microsoft SQL Server 2016 CTP2. Some of the features might change in future releases of the product. In this lab, you will learn about some, but not all, new features.