Microsoft SQL Server 2016 CTP 3.1 In-Memory OLTP

Contents

Overview

Note: Estimated time to complete lab is 40–45 minutes.

In-Memory OLTP provides row-based, in-memory data access and modification capabilities, used mostly for transaction-processing workloads. This technology uses lock- and latch-free architecture that enables linear scaling. In-Memory OLTP has memory-optimized data structures and provides native compilation, creating more efficient data access and querying capabilities. This technology is integrated into Microsoft SQL Server 2016, which enables lower total cost of ownership since developers and DBAs can use the same T-SQL, client stack, tooling, backups, and AlwaysOn features. Furthermore, the same database can have both on-disk and in-memory features.

SQL Server 2016 In-Memory OLTP can dramatically improve throughput and latency on transactional processing workloads and can provide significant performance improvements. SQL Server 2016 improvements to In-Memory OLTP enable scaling to larger databases and higher throughput in order to support bigger workloads. In addition, a number of limitations concerning tables and stored procedures have been removed to make it easier to migrate your applications to and leverage the benefits of In-Memory OLTP.

This hands-on lab will familiarize you with SQL Server 2016 In-Memory OLTP and help you see the performance impact this technology can have. It will also highlight some of the improvements in SQL Server 2016 in this area. In particular, you will learn how to:

1. Migrate a traditional workload to In-Memory OLTP and verify performance gain.
2. Inspect the In-Memory OLTP objects, including memory-optimized tables, memory-optimized table types, natively compiled stored procedures, etc.
3. Understand stored procedures that are defined in SQL Server to be natively compiled.
4. Better query surface area for UNION, UNION ALL, OUTER JOIN, etc.

At the end of this lab, you will have worked through some of the most common scenarios involved with In-Memory OLTP and learned about some of the most significant improvements to this technology in SQL Server 2016.

Setting up the environment

To set up the environment, perform the steps below:

1. Log on to the virtual machine environment using the following account information:

   User: **Labuser**
   Password: **Pass@word12**

2. Use the script file located at **C:\SQL Server 2016 CPT3.1 HOLs\In-MemoryOLTP\In-MemOLTP.sql** for this lab.
3. Open **Microsoft SQL Server Management 2016 Studio** on the lab machine.
4. Access the **AdventureWorks2016CTP3** database for this lab.

Note: It will be easier to use SQL Server 2016 Management Studio (SSMS) with a higher screen resolution. If you have a monitor that supports a screen resolution larger than 1024 x 768, change the resolution to as high as 1920 x 1080 for the lab.

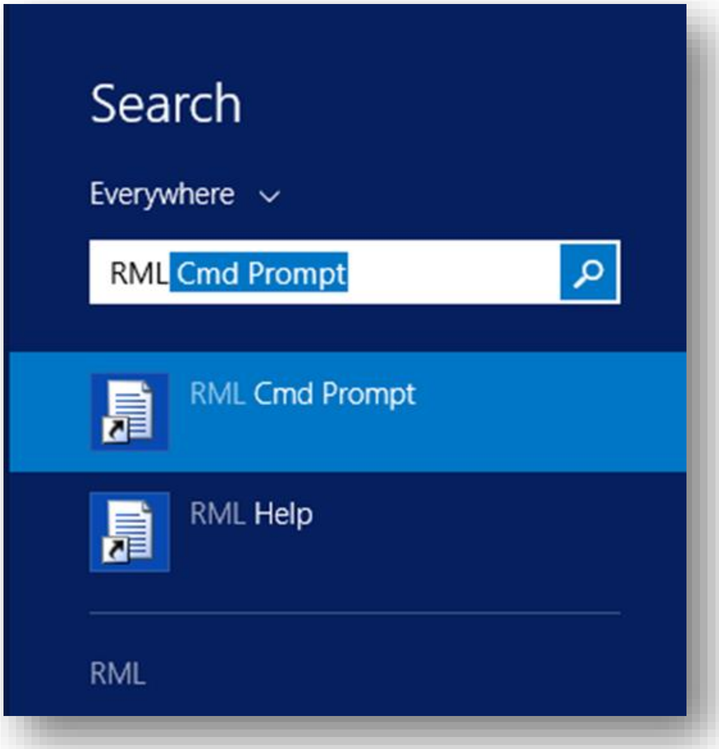
Configure In-Memory OLTP sample workload

Before testing the In-Memory OLTP workload, you first need to test the sample workload for memory-optimized tables and disk-based tables. In this section, you will use the handy ostress.exe utility to execute the two stored procedures at stressful levels. You can compare how long it takes the two stress runs to complete.

Ostress is a command-line tool that was developed by the Microsoft CSS SQL Server support team. This tool can be used to execute queries or run stored procedures in parallel. You can configure the number of threads to run a given T-SQL statement in parallel, and you can specify how many times the statement should be executed on this thread. Ostress will spin up the threads and execute the statement on all threads in parallel. After execution finishes for all threads, ostress will report the total time taken for all threads to complete.

Below are instructions for running the sample workload using ostress utility that illustrates performance benefits, as well as instructions for inspecting In-Memory OLTP objects in the database.

1. To open the RML Cmd Prompt, click **Start**. Type **RML CMD Prompt** in the Search Option. (You can find this under **All Apps** -> **RML Utilities for SQL Server**.)



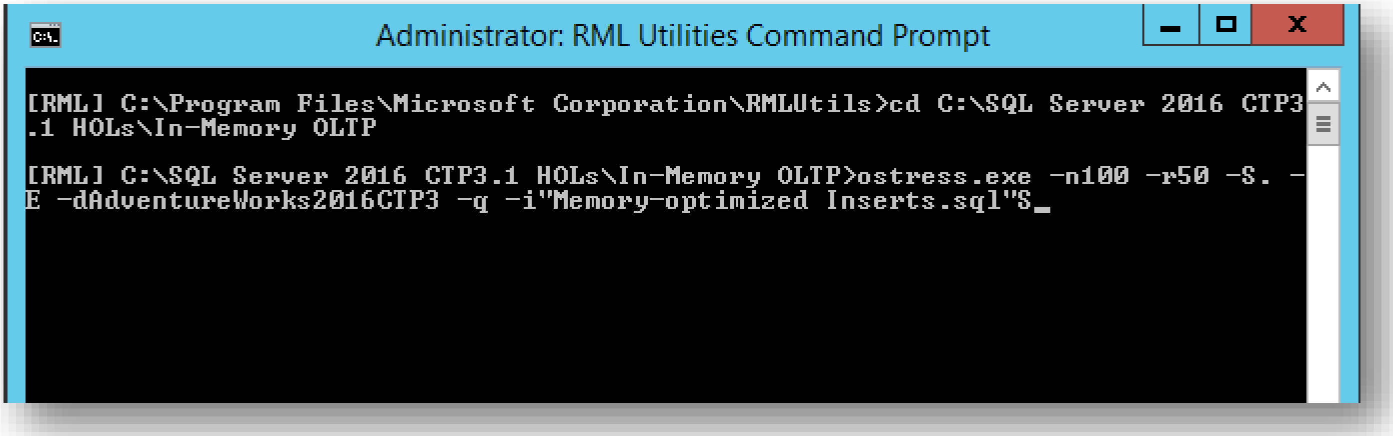
Note: Ostress is run from the command-line prompt. It is most convenient to run the tool from the "RML Cmd Prompt", which is installed as part of the RML Utilities. You can install the RML utilities from the following location: http://blogs.msdn.com/b/psssql/archive/2013/10/29/cumulative

2. From RML Cmd Prompt, navigate to the **C:\SQL Server 2016 CTP3.1 HOLs\In-Memory OLTP** folder that has the required SQL sample scripts for In-Memory OLTP. To navigate to the required folder, use the code below:

```
cd C:\SQL Server 2016 CTP3.1 HOLs\In-Memory OLTP
```

3. Now run the following commands in the RML Cmd Prompt window.

```
ostress.exe -n100 -r50 -S. -E -dAdventureWorks2016CTP3 -q -i"Memory-optimized
Inserts.sql"
```

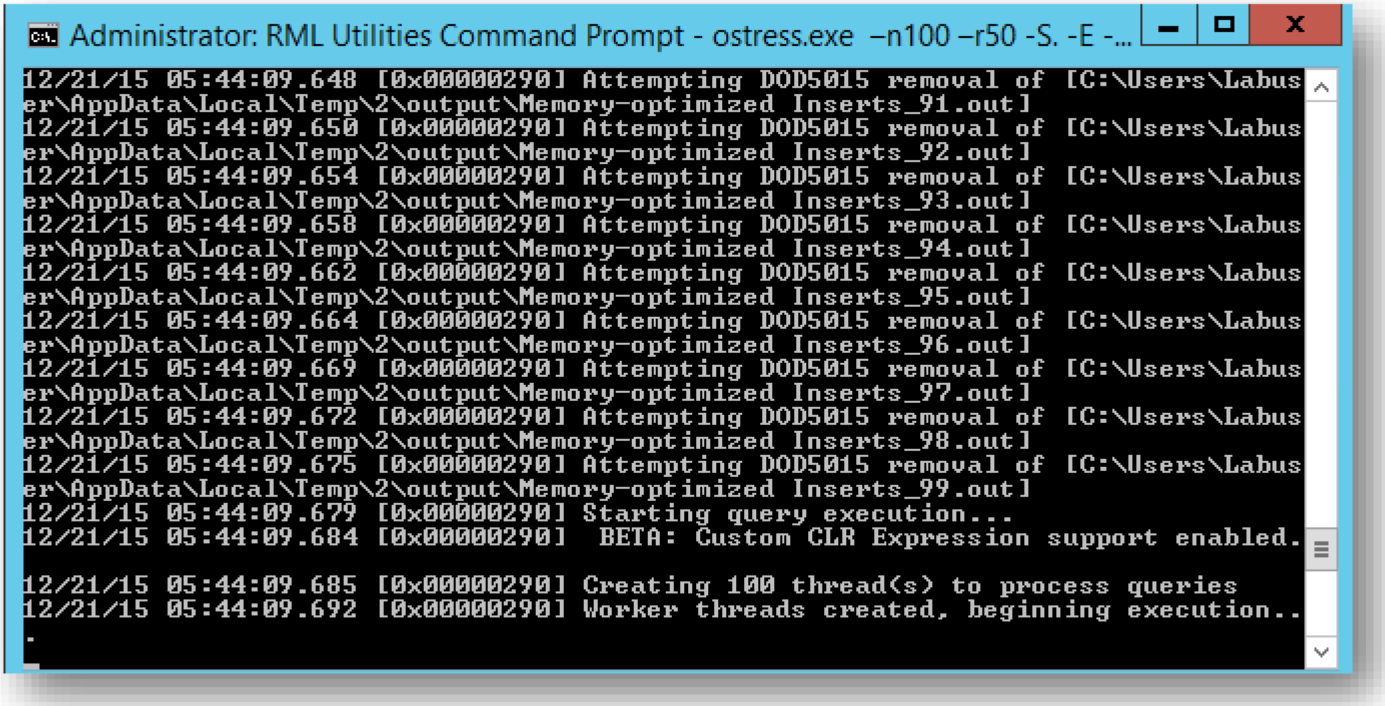


```
[RML] C:\Program Files\Microsoft Corporation\RMLUtils>cd C:\SQL Server 2016 CTP3
.1 HOLs\In-Memory OLTP

[RML] C:\SQL Server 2016 CTP3.1 HOLs\In-Memory OLTP>ostress.exe -n100 -r50 -S. -
E -dAdventureWorks2016CTP3 -q -i"Memory-optimized Inserts.sql"S_
```

4. For a sample test, we will first insert 100,000 sales orders into memory-optimized tables using the above command in the RML Cmd Prompt. The main options to consider for running ostress with this sample are:

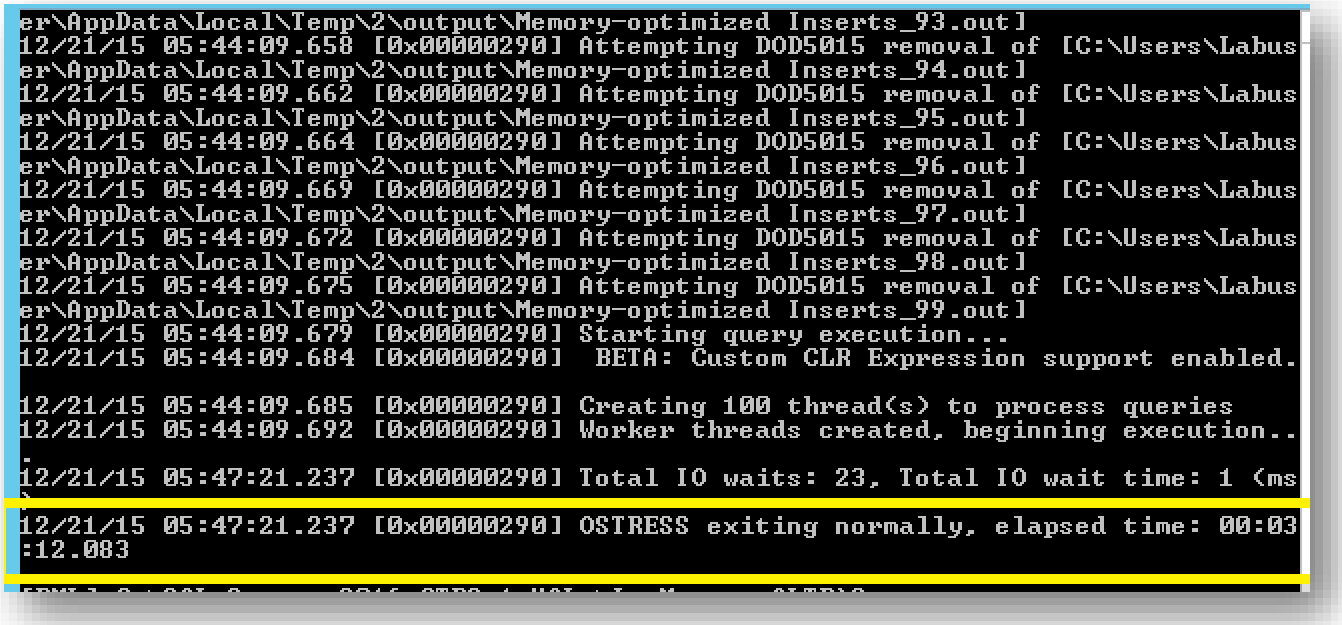
   - -s name of SQL Server instance with which to connect.
   - -e use Windows authentication to connect (default).
   - -d name of the database (for this example AdventureWorks2016CTP3).
   - -q the T-SQL statement to be executed. Here we are using the Memory-optimized Inserts.sql. This SQL script contains the main stored procedure Sales.usp_InsertSalesOrder_inmem. The script constructs a table-valued parameter (TVP) with sample data, and calls the procedure to insert a sales order with five line items. This SQL script is available at **C:\SQL Server 2016 CTP3.1 HOLs\In-Memory OLTP**, where you can view the code of the Memory-optimized Inserts.sql.
   - -n number of connections processing each input file/query. Here we are running 100 connections concurrently (-n100).
   - -r the number of iterations for each connection to execute each input file/query. In this example, each connection runs the T-SQL script five times (-r50).

5. If everything works as expected, your command window will look similar to the following. Error messages are not expected.


Administrator: RML Utilities Command Prompt - ostress.exe –n100 –r50 -S. -E -...

```
12/21/15 05:44:09.648 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_91.out]
12/21/15 05:44:09.650 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_92.out]
12/21/15 05:44:09.654 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_93.out]
12/21/15 05:44:09.658 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_94.out]
12/21/15 05:44:09.662 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_95.out]
12/21/15 05:44:09.664 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_96.out]
12/21/15 05:44:09.669 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_97.out]
12/21/15 05:44:09.672 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_98.out]
12/21/15 05:44:09.675 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_99.out]
12/21/15 05:44:09.679 [0x00000290] Starting query execution...
12/21/15 05:44:09.684 [0x00000290]  BETA: Custom CLR Expression support enabled.

12/21/15 05:44:09.685 [0x00000290] Creating 100 thread(s) to process queries
12/21/15 05:44:09.692 [0x00000290] Worker threads created, beginning execution..
.
```

6. When ostress.exe completes, it writes the run duration as its final line of output in the RML Cmd window.



```
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_93.out]
12/21/15 05:44:09.658 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_94.out]
12/21/15 05:44:09.662 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_95.out]
12/21/15 05:44:09.664 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_96.out]
12/21/15 05:44:09.669 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_97.out]
12/21/15 05:44:09.672 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_98.out]
12/21/15 05:44:09.675 [0x00000290] Attempting DOD5015 removal of [C:\Users\Labus
er\AppData\Local\Temp\2\output\Memory-optimized Inserts_99.out]
12/21/15 05:44:09.679 [0x00000290] Starting query execution...
12/21/15 05:44:09.684 [0x00000290]  BETA: Custom CLR Expression support enabled.

12/21/15 05:44:09.685 [0x00000290] Creating 100 thread(s) to process queries
12/21/15 05:44:09.692 [0x00000290] Worker threads created, beginning execution..
.
12/21/15 05:47:21.237 [0x00000290] Total IO waits: 23, Total IO wait time: 1 (ms
)
12/21/15 05:47:21.237 [0x00000290] OSTRESS exiting normally, elapsed time: 00:03
:12.083
```

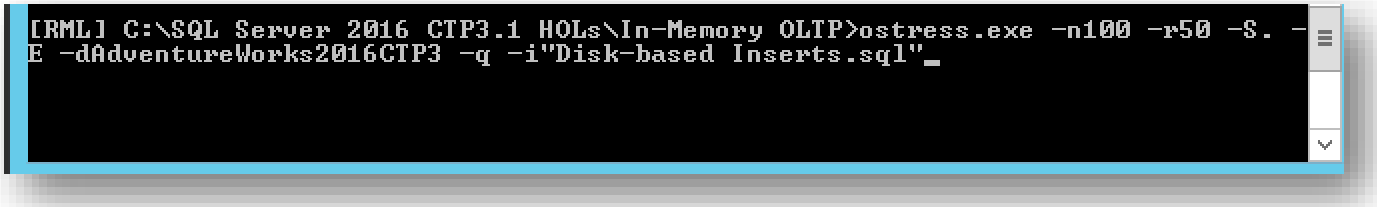
Note: In our example, it took three minutes and 12 seconds for the workload on memory-optimized tables.

Compare performance with disk-based tables

Now we will compare the runtime for the workload on memory-optimized tables versus disk-based tables.

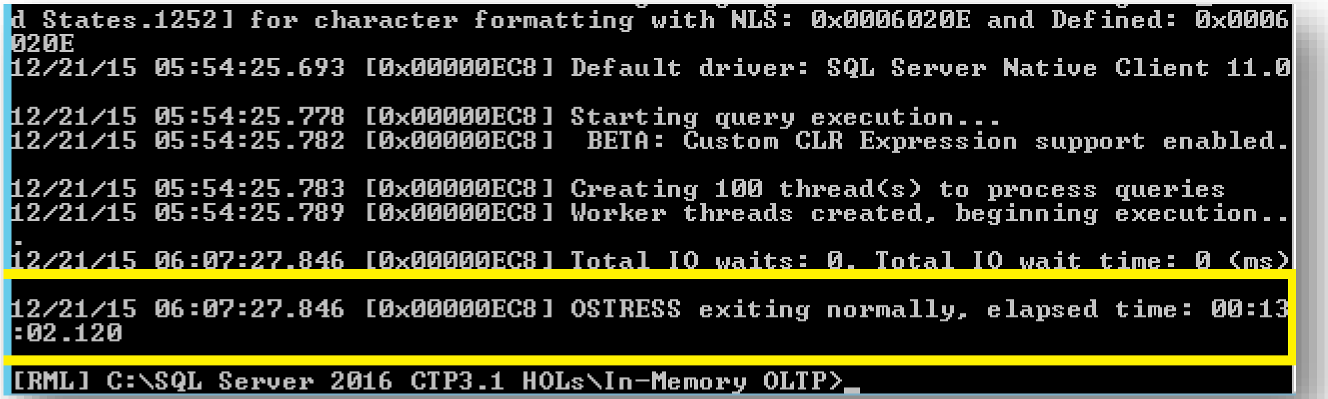
7. To insert the same number of 100,000 sales orders in a disk-based table, copy and paste the following command in the RML Cmd Prompt.

```
ostress.exe -n100 -r50 -S. -E -dAdventureWorks2016CTP3 -q -i"Disk-based Inserts.sql"
```



```
[RML] C:\SQL Server 2016 CTP3.1 HOLs\In-Memory OLTP>ostress.exe -n100 -r50 -S. -
E -dAdventureWorks2016CTP3 -q -i"Disk-based Inserts.sql"
```

8. When ostress.exe completes, it writes the run duration as its final line of output in the RML Cmd window.



```
d States.1252] for character formatting with NLS: 0x0006020E and Defined: 0x0006
020E
12/21/15 05:54:25.693 [0x00000EC8] Default driver: SQL Server Native Client 11.0

12/21/15 05:54:25.778 [0x00000EC8] Starting query execution...
12/21/15 05:54:25.782 [0x00000EC8]  BETA: Custom CLR Expression support enabled.

12/21/15 05:54:25.783 [0x00000EC8] Creating 100 thread(s) to process queries
12/21/15 05:54:25.789 [0x00000EC8] Worker threads created, beginning execution..
.
12/21/15 06:07:27.846 [0x00000EC8] Total IO waits: 0, Total IO wait time: 0 (ms)

12/21/15 06:07:27.846 [0x00000EC8] OSTRESS exiting normally, elapsed time: 00:13
:02.120

[RML] C:\SQL Server 2016 CTP3.1 HOLs\In-Memory OLTP>
```

Note: For the disk-based table in our example, it took 13 minutes and two seconds for the workload (your time may vary).

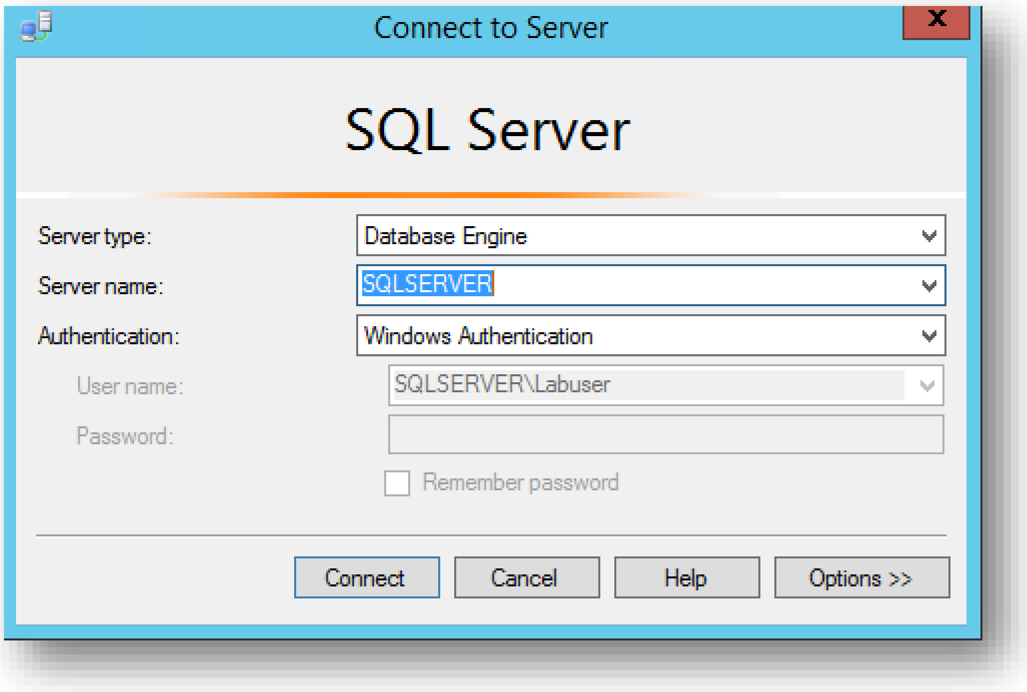
Important: Notice that in the above comparison, in-memory tests on memory-optimized versus disk-based tables have shown four to five times improved performance for this simplistic workload, with ostress running on an Azure Virtual Machine in the same Azure region as the database.

9. Close the RML cmd Prompt window.

Inspecting In-Memory OLTP objects

You can now inspect memory-optimized tables through Object Explorer in SQL Server 2016 Management Studio, or you can query the catalog views.

1. Open SQL Server 2016 Management Studio, and connect to the **SQLSERVER** database engine instance (shown below).



Connect to Server

SQL Server

Server type: Database Engine

Server name: SQLSERVER

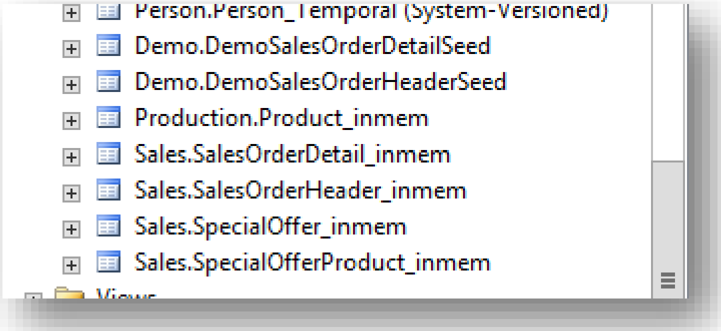
Authentication: Windows Authentication

User name: SQLSERVER\Labuser

Password:

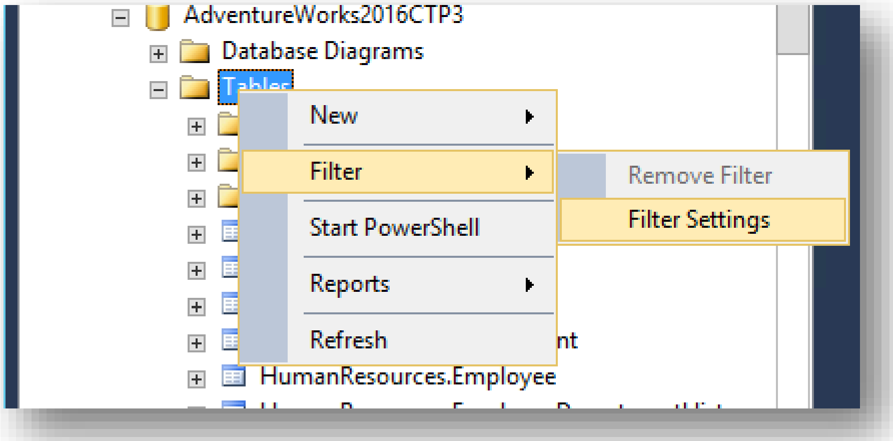
□ Remember password

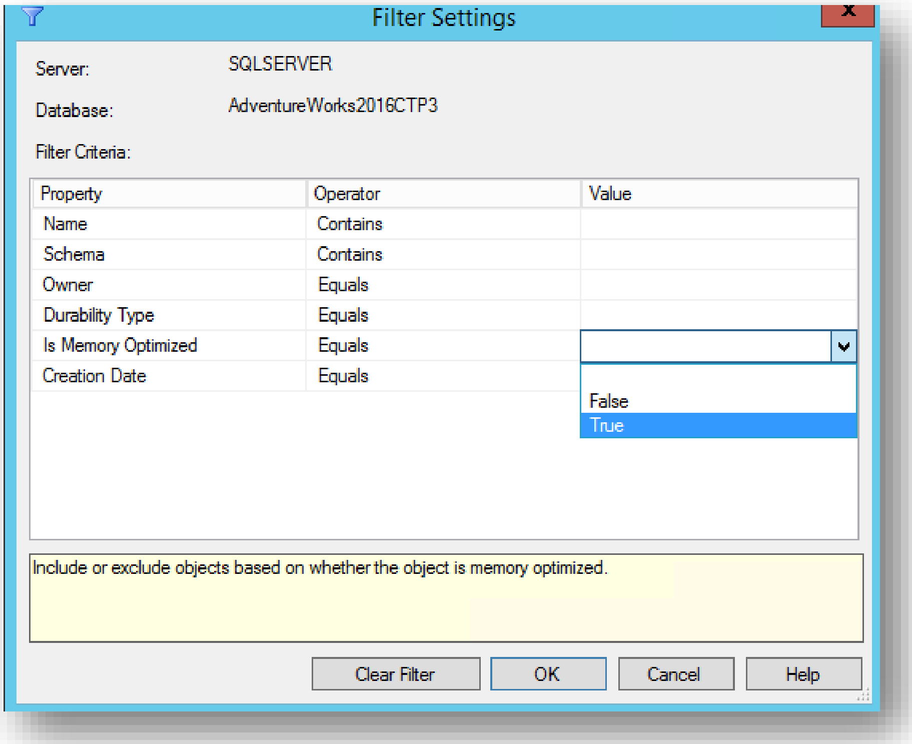
2. Click **AdventureWorks2016CTP3 database**, and then click **Tables**. This database contains the following memory-optimized tables:
   - Products.Product_inmem
   - Sales.SalesOrderHeader_inmem
   - Sales.SalesOrderDetail_inmem
   - Sales.SpecialOffer_inmem
   - Sales.SpecialOfferProduct_inmem
   - Demo.DemoSalesOrderHeaderSeed
   - Demo.DemoSalesOrderDetailSeed



Person.Person_Temporal (System-Versioned)
Demo.DemoSalesOrderDetailSeed
Demo.DemoSalesOrderHeaderSeed
Production.Product_inmem
Sales.SalesOrderDetail_inmem
Sales.SalesOrderHeader_inmem
Sales.SpecialOffer_inmem
Sales.SpecialOfferProduct_inmem
Views

3. You can inspect memory-optimized tables through **Object Explorer in SSMS** by right-clicking **Tables** -> **Filter** -> **Filter Settings** -> **Is Memory Optimized equals True**.





Filter Settings

Server: SQLSERVER

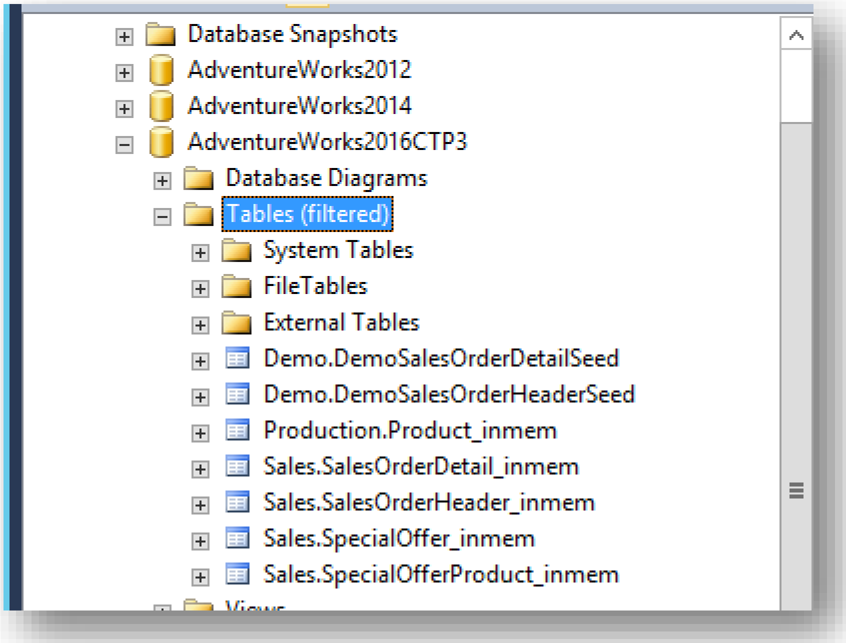
Database: AdventureWorks2016CTP3

Filter Criteria:

Property	Operator	Value
Name	Contains	
Schema	Contains	
Owner	Equals	
Durability Type	Equals	
Is Memory Optimized	Equals	
Creation Date	Equals	

Include or exclude objects based on whether the object is memory optimized.

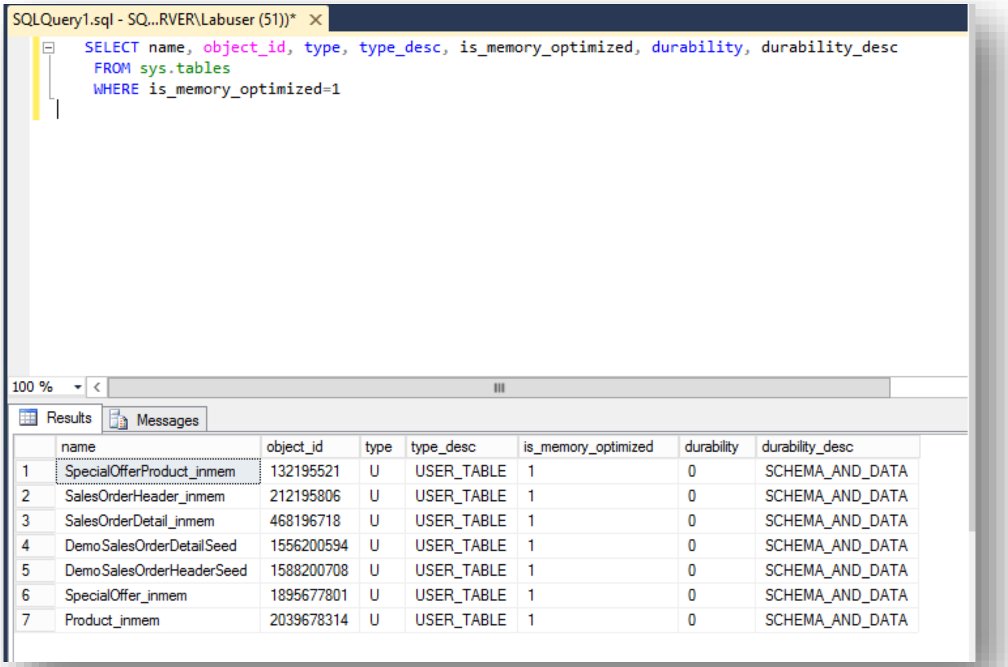
4. After applying the filter, you can see only the memory-optimized table. Remove the filter after verifying this.



```
⊞ Database Snapshots
⊞ AdventureWorks2012
⊞ AdventureWorks2014
⊟ AdventureWorks2016CTP3
    ⊞ Database Diagrams
    ⊟ Tables (filtered)
        ⊞ System Tables
        ⊞ FileTables
        ⊞ External Tables
        ⊞ Demo.DemoSalesOrderDetailSeed
        ⊞ Demo.DemoSalesOrderHeaderSeed
        ⊞ Production.Product_inmem
        ⊞ Sales.SalesOrderDetail_inmem
        ⊞ Sales.SalesOrderHeader_inmem
        ⊞ Sales.SpecialOffer_inmem
        ⊞ Sales.SpecialOfferProduct_inmem
    ⊞ Views
```

5. You can also query the catalog views. Copy and paste the following script:

```
   SELECT name, object_id, type, type_desc, is_memory_optimized, durability, 
durability_desc
    FROM sys.tables
    WHERE is_memory_optimized=1
```

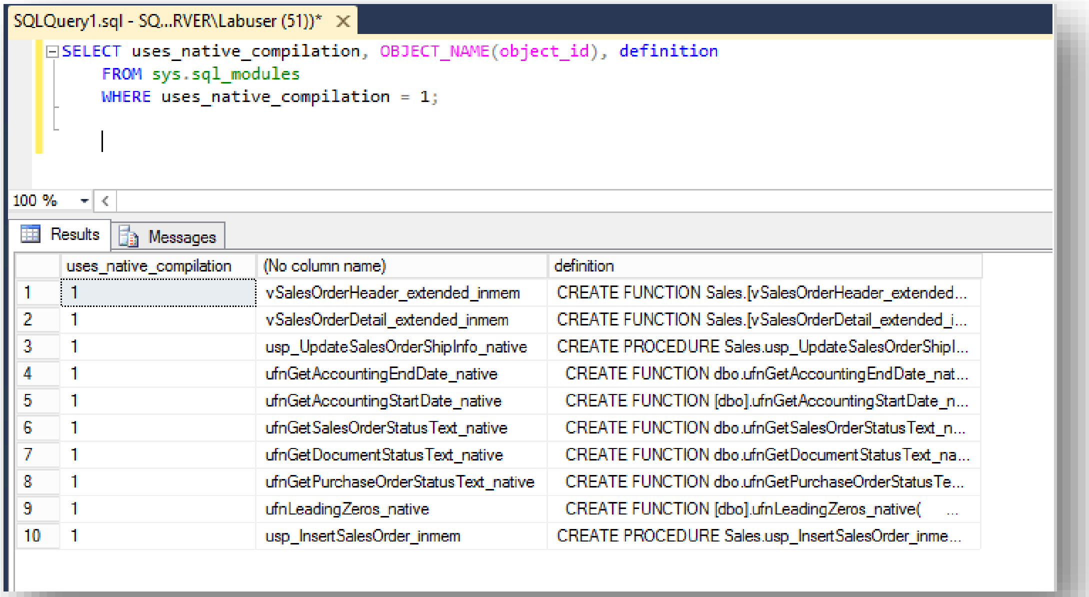


```
SELECT name, object_id, type, type_desc, is_memory_optimized, durability, durability_desc
 FROM sys.tables
 WHERE is_memory_optimized=1
```

	name	object_id	type	type_desc	is_memory_optimized	durability	durability_desc
1	SpecialOfferProduct_inmem	132195521	U	USER_TABLE	1	0	SCHEMA_AND_DATA
2	SalesOrderHeader_inmem	212195806	U	USER_TABLE	1	0	SCHEMA_AND_DATA
3	SalesOrderDetail_inmem	468196718	U	USER_TABLE	1	0	SCHEMA_AND_DATA
4	DemoSalesOrderDetailSeed	1556200594	U	USER_TABLE	1	0	SCHEMA_AND_DATA
5	DemoSalesOrderHeaderSeed	1588200708	U	USER_TABLE	1	0	SCHEMA_AND_DATA
6	SpecialOffer_inmem	1895677801	U	USER_TABLE	1	0	SCHEMA_AND_DATA
7	Product_inmem	2039678314	U	USER_TABLE	1	0	SCHEMA_AND_DATA

6. The natively compiled modules can be inspected through Object Explorer, or you can query the catalog views.

```
SELECT uses_native_compilation, OBJECT_NAME(object_id), definition
    FROM sys.sql_modules
    WHERE uses_native_compilation = 1;
```

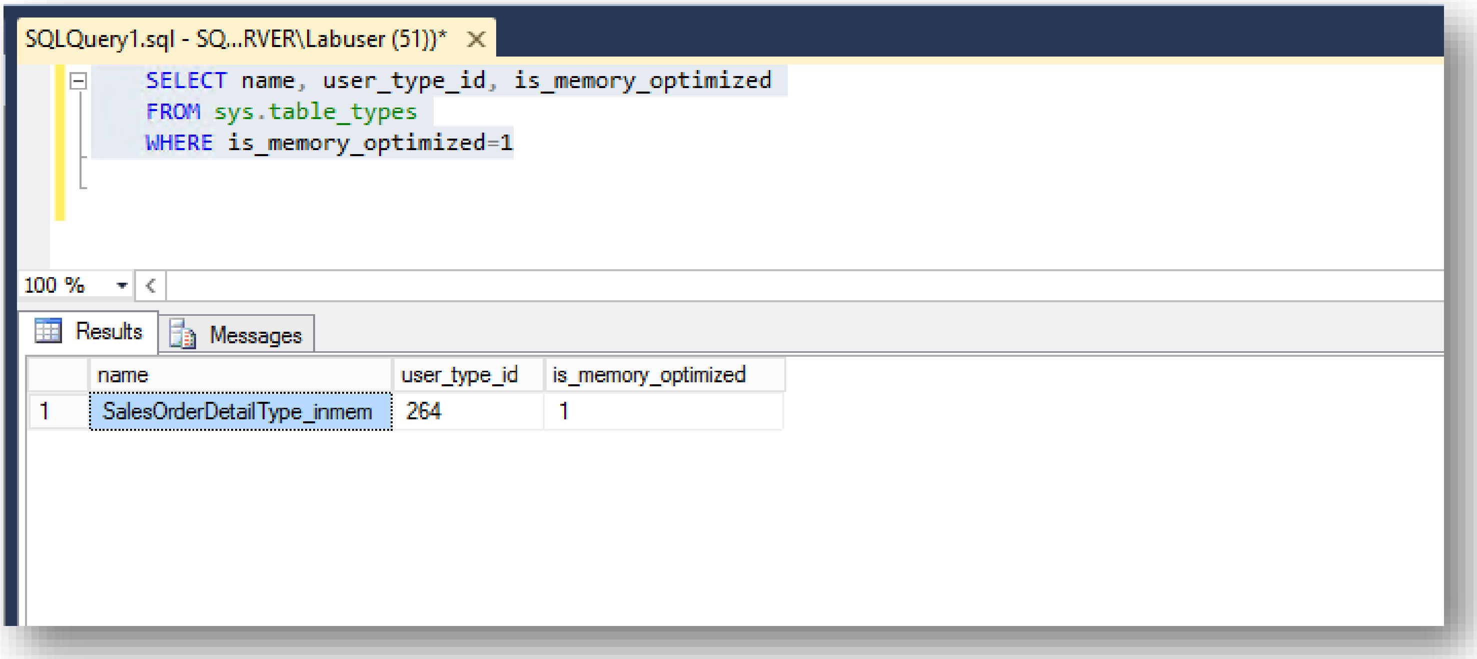


```
SELECT uses_native_compilation, OBJECT_NAME(object_id), definition
    FROM sys.sql_modules
    WHERE uses_native_compilation = 1;
```

	uses_native_compilation	(No column name)	definition
1	1	vSalesOrderHeader_extended_inmem	CREATE FUNCTION Sales.[vSalesOrderHeader_extended...
2	1	vSalesOrderDetail_extended_inmem	CREATE FUNCTION Sales.[vSalesOrderDetail_extended_i...
3	1	usp_UpdateSalesOrderShipInfo_native	CREATE PROCEDURE Sales.usp_UpdateSalesOrderShipI...
4	1	ufnGetAccountingEndDate_native	CREATE FUNCTION dbo.ufnGetAccountingEndDate_nat...
5	1	ufnGetAccountingStartDate_native	CREATE FUNCTION [dbo].ufnGetAccountingStartDate_n...
6	1	ufnGetSalesOrderStatusText_native	CREATE FUNCTION dbo.ufnGetSalesOrderStatusText_n...
7	1	ufnGetDocumentStatusText_native	CREATE FUNCTION dbo.ufnGetDocumentStatusText_na...
8	1	ufnGetPurchaseOrderStatusText_native	CREATE FUNCTION dbo.ufnGetPurchaseOrderStatusTe...
9	1	ufnLeadingZeros_native	CREATE FUNCTION [dbo].ufnLeadingZeros_native(...
10	1	usp_InsertSalesOrder_inmem	CREATE PROCEDURE Sales.usp_InsertSalesOrder_inme...

7. You can also inspect the **memory-optimized table type** through the following query:

```
    SELECT name, user_type_id, is_memory_optimized
    FROM sys.table_types
    WHERE is_memory_optimized=1
```



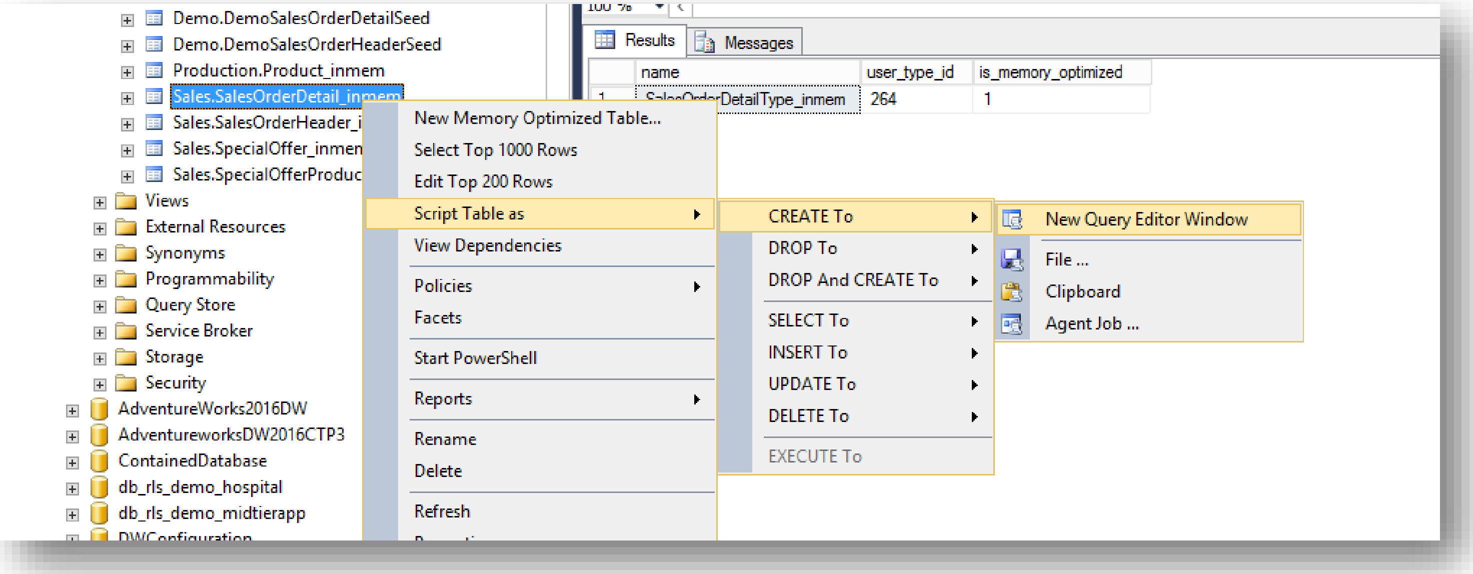
```
SELECT name, user_type_id, is_memory_optimized
FROM sys.table_types
WHERE is_memory_optimized=1
```

	name	user_type_id	is_memory_optimized
1	SalesOrderDetailType_inmem	264	1

Exploring memory-optimized tables

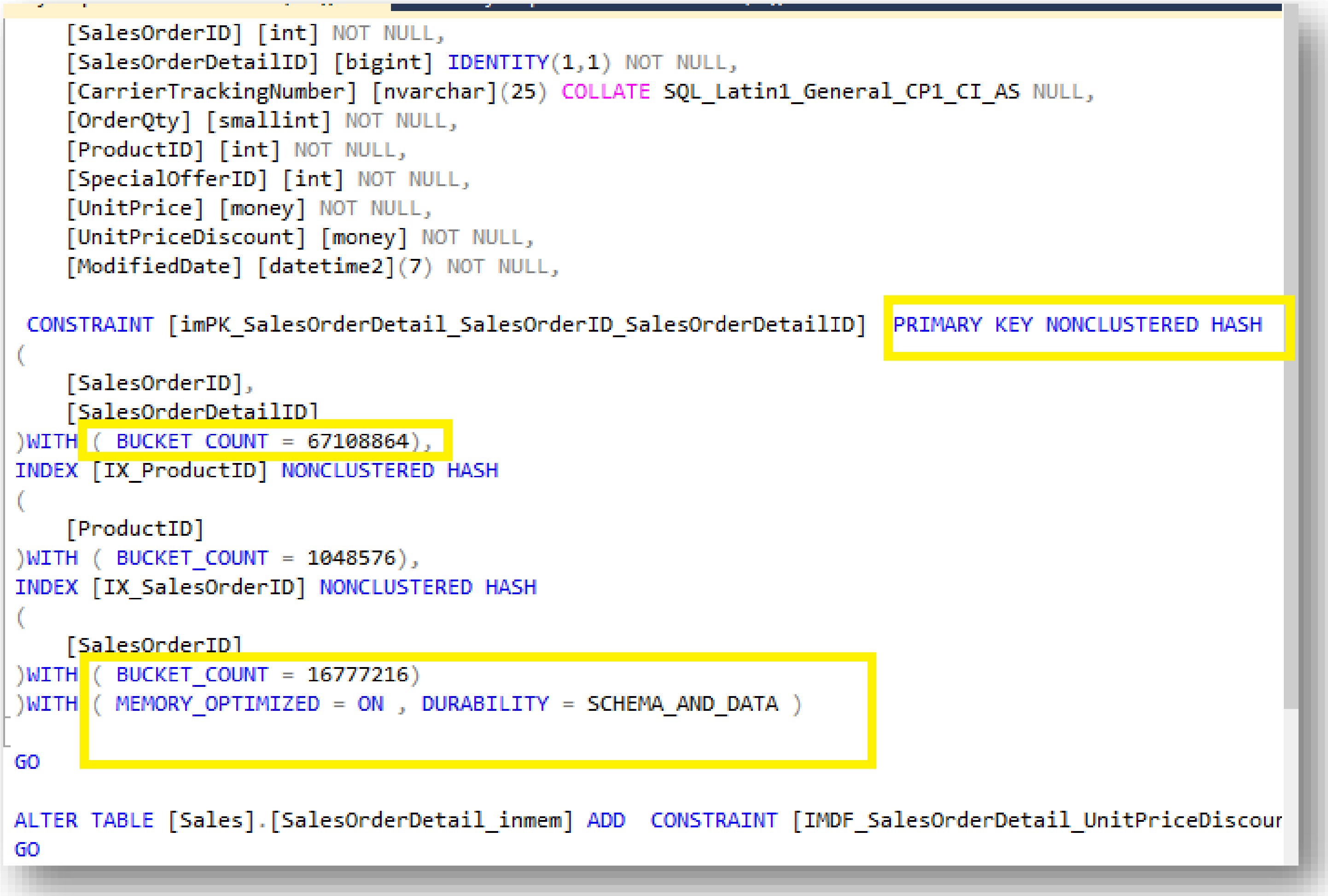
Memory-optimized tables can be created with both nonclustered indexes and nonclustered hash indexes. Hash indexes have an associated bucket count. Hash indexes consume a fixed amount of memory, which is a function of the bucket count.

1. In Object Explorer, expand the table list, scroll to the bottom of the list, and find the Sales.SalesOrderDetail_inmem. Right-click the table, and then select **Script Table as** -> **CREATE To** -> **New Query Window**.



2. Note that three nonclustered hash indexes are created, for which bucket counts are set. Also notice that the end of the CREATE TABLE statement has the settings of MEMORY_OPTIMIZED = ON and DURABILITY = SCHEMA_AND_DATA. This last setting defines the table as being durable, in that it maintains its data. If

it had been set to SCHEMA_ONLY (often used for transient data such as ETL staging), any data in the table would be lost when the server is restarted.



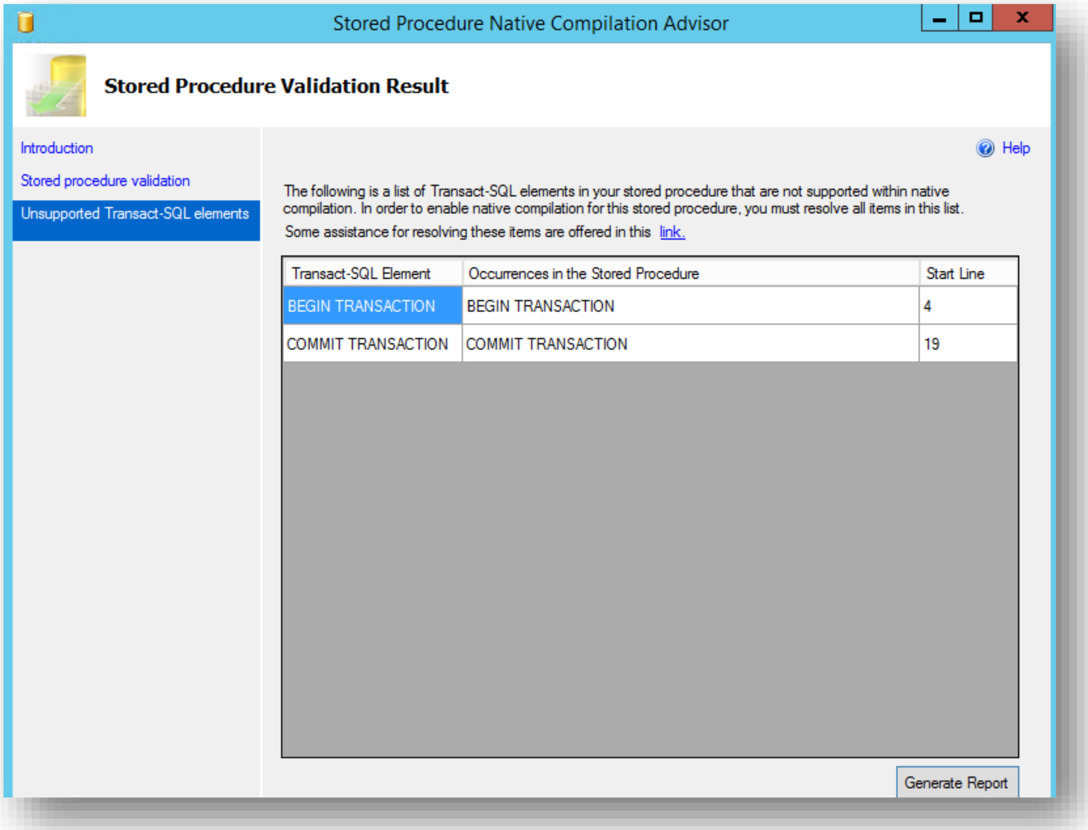
```
    [SalesOrderID] [int] NOT NULL,
    [SalesOrderDetailID] [bigint] IDENTITY(1,1) NOT NULL,
    [CarrierTrackingNumber] [nvarchar](25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
    [OrderQty] [smallint] NOT NULL,
    [ProductID] [int] NOT NULL,
    [SpecialOfferID] [int] NOT NULL,
    [UnitPrice] [money] NOT NULL,
    [UnitPriceDiscount] [money] NOT NULL,
    [ModifiedDate] [datetime2](7) NOT NULL,

 CONSTRAINT [imPK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID] PRIMARY KEY NONCLUSTERED HASH
(
    [SalesOrderID],
    [SalesOrderDetailID]
)WITH ( BUCKET_COUNT = 67108864),
INDEX [IX_ProductID] NONCLUSTERED HASH
(
    [ProductID]
)WITH ( BUCKET_COUNT = 1048576),
INDEX [IX_SalesOrderID] NONCLUSTERED HASH
(
    [SalesOrderID]
)WITH ( BUCKET_COUNT = 16777216)
)WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA )

GO

ALTER TABLE [Sales].[SalesOrderDetail_inmem] ADD  CONSTRAINT [IMDF_SalesOrderDetail_UnitPriceDiscour
GO
```

Exploring natively compiled stored procedures better T-SQL coverage

Native compilation allows faster data access and more efficient query execution than interpreted (traditional) Transact-SQL. Natively compiled stored procedures are parsed and compiled when they are loaded to native DLLs. This is in contrast to other stored procedures which are compiled on first run, have an execution plan created and reused, and use an interpreter for execution.

Important: In SQL Server 2014, it was not possible to use some standard T-SQL such as left outer joins, union all, and distinct. SQL Server 2016 now has better coverage of T-SQL query surface area.

1. In Object Explorer, expand **Programmability** -> **Stored Procedures**. Right-click **Sales.usp_UpdateSalesOrderShipInfo_ondisk**, select **Native Compilation Advisor**, and then click **Next** twice.


Stored Procedure Native Compilation Advisor
Introduction
Stored procedure validation
Unsupported Transact-SQL elements
Help
Generate Report

Stored Procedure Validation Result

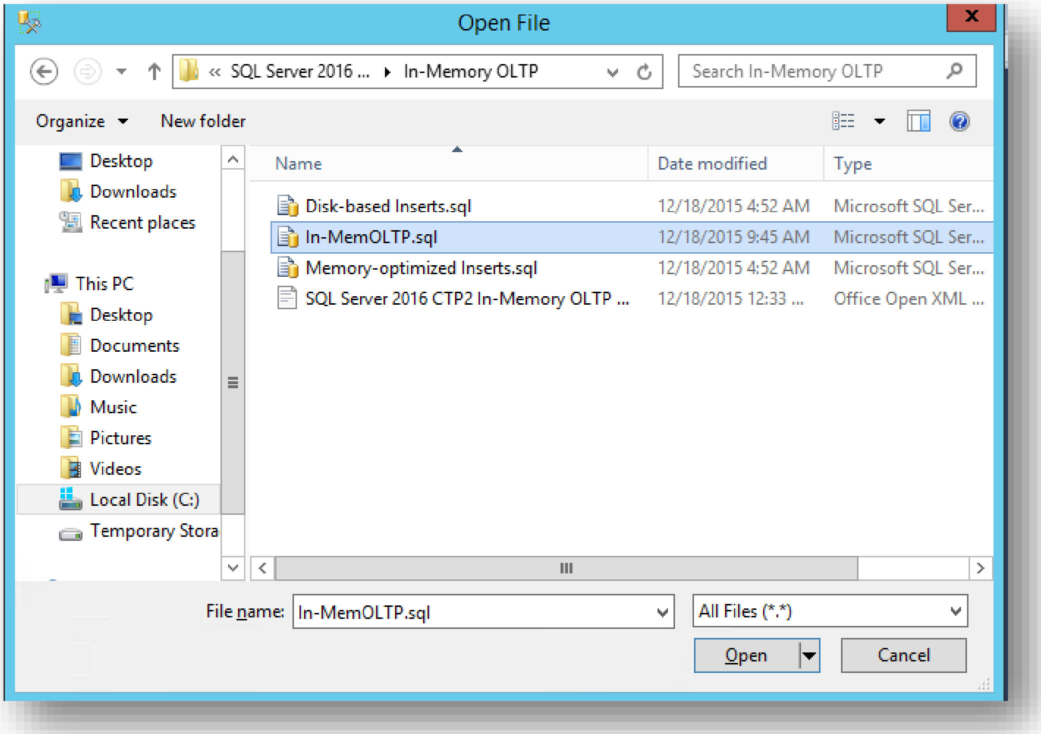
The following is a list of Transact-SQL elements in your stored procedure that are not supported within native compilation. In order to enable native compilation for this stored procedure, you must resolve all items in this list.

Some assistance for resolving these items are offered in this link.

Transact-SQL Element	Occurrences in the Stored Procedure	Start Line
BEGIN TRANSACTION	BEGIN TRANSACTION	4
COMMIT TRANSACTION	COMMIT TRANSACTION	19

This wizard can help identify any T-SQL constructs that are not supported with native compilation.

2. Now in SSMS, click **Ctrl+O**, and then open the sql script in **C:\SQL Server 2016 CTP3.1 HOLs\In-Memory OLTP**.

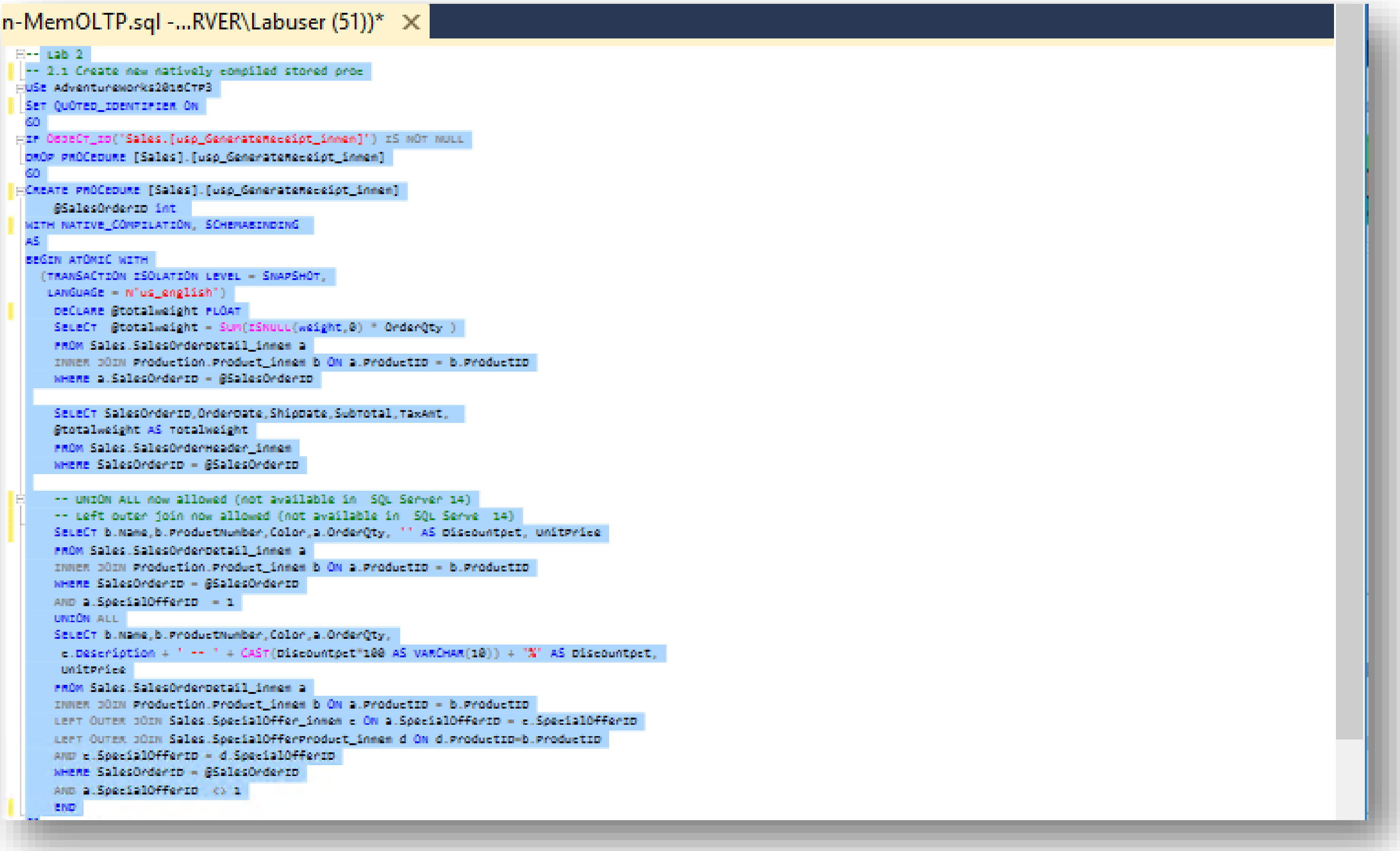


Name	Date modified	Type
Disk-based Inserts.sql	12/18/2015 4:52 AM	Microsoft SQL Ser...
In-MemOLTP.sql	12/18/2015 9:45 AM	Microsoft SQL Ser...
Memory-optimized Inserts.sql	12/18/2015 4:52 AM	Microsoft SQL Ser...
SQL Server 2016 CTP2 In-Memory OLTP ...	12/18/2015 12:33 ...	Office Open XML ...

File name: In-MemOLTP.sql

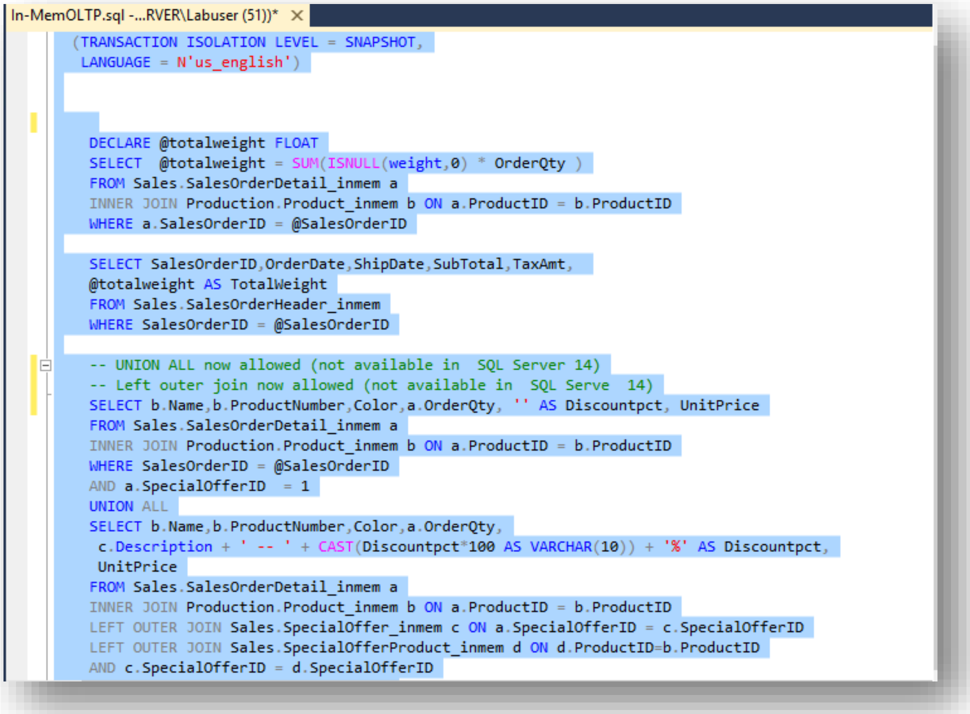
All Files (*.*)

3. Run only the highlighted Lab2 part of the script. Run only this section of the script to create the new natively compiled stored procedure.



```
-- Lab 2
-- 2.1 Create new natively compiled stored proc
USE AdventureWorks2016CTP3
SET QUOTED_IDENTIFIER ON
GO
IF OBJECT_ID('Sales.[usp_GenerateReceipt_inmem]') IS NOT NULL
DROP PROCEDURE [Sales].[usp_GenerateReceipt_inmem]
GO
CREATE PROCEDURE [Sales].[usp_GenerateReceipt_inmem]
    @SalesOrderID int
WITH NATIVE_COMPILATION, SCHEMABINDING
AS
BEGIN ATOMIC WITH
  (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
   LANGUAGE = N'us_english')
    DECLARE @totalweight FLOAT
    SELECT  @totalweight = SUM(ISNULL(weight,0) * OrderQty )
    FROM Sales.SalesOrderDetail_inmem a
    INNER JOIN Production.Product_inmem b ON a.ProductID = b.ProductID
    WHERE a.SalesOrderID = @SalesOrderID

    SELECT SalesOrderID,OrderDate,ShipDate,SubTotal,TaxAmt,
    @totalweight AS Totalweight
    FROM Sales.SalesOrderHeader_inmem
    WHERE SalesOrderID = @SalesOrderID

    -- UNION ALL now allowed (not available in  SQL Server 14)
    -- Left outer join now allowed (not available in  SQL Serve  14)
    SELECT b.Name,b.ProductNumber,Color,a.OrderQty, '' AS DiscountPct, UnitPrice
    FROM Sales.SalesOrderDetail_inmem a
    INNER JOIN Production.Product_inmem b ON a.ProductID = b.ProductID
    WHERE SalesOrderID = @SalesOrderID
    AND a.SpecialOfferID  = 1
    UNION ALL
    SELECT b.Name,b.ProductNumber,Color,a.OrderQty,
     c.Description + ' -- ' + CAST(DiscountPct*100 AS VARCHAR(10)) + '%' AS DiscountPct,
     UnitPrice
    FROM Sales.SalesOrderDetail_inmem a
    INNER JOIN Production.Product_inmem b ON a.ProductID = b.ProductID
    LEFT OUTER JOIN Sales.SpecialOffer_inmem c ON a.SpecialOfferID = c.SpecialOfferID
    LEFT OUTER JOIN Sales.SpecialOfferProduct_inmem d ON d.ProductID=b.ProductID
    AND c.SpecialOfferID = d.SpecialOfferID
    WHERE SalesOrderID = @SalesOrderID
    AND a.SpecialOfferID  <> 1
    END
```

4. Notice the WITH NATIVE_COMPILATION statement. Also note that the SCHEMABINDING is required.



In-MemOLTP.sql -...RVER\Labuser (51))*

```
 (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
  LANGUAGE = N'us_english')



   DECLARE @totalweight FLOAT
   SELECT  @totalweight = SUM(ISNULL(weight,0) * OrderQty )
   FROM Sales.SalesOrderDetail_inmem a
   INNER JOIN Production.Product_inmem b ON a.ProductID = b.ProductID
   WHERE a.SalesOrderID = @SalesOrderID

   SELECT SalesOrderID,OrderDate,ShipDate,SubTotal,TaxAmt,
   @totalweight AS TotalWeight
   FROM Sales.SalesOrderHeader_inmem
   WHERE SalesOrderID = @SalesOrderID

   -- UNION ALL now allowed (not available in  SQL Server 14)
   -- Left outer join now allowed (not available in  SQL Serve  14)
   SELECT b.Name,b.ProductNumber,Color,a.OrderQty, '' AS Discountpct, UnitPrice
   FROM Sales.SalesOrderDetail_inmem a
   INNER JOIN Production.Product_inmem b ON a.ProductID = b.ProductID
   WHERE SalesOrderID = @SalesOrderID
   AND a.SpecialOfferID  = 1
   UNION ALL
   SELECT b.Name,b.ProductNumber,Color,a.OrderQty,
    c.Description + ' -- ' + CAST(Discountpct*100 AS VARCHAR(10)) + '%' AS Discountpct,
    UnitPrice
   FROM Sales.SalesOrderDetail_inmem a
   INNER JOIN Production.Product_inmem b ON a.ProductID = b.ProductID
   LEFT OUTER JOIN Sales.SpecialOffer_inmem c ON a.SpecialOfferID = c.SpecialOfferID
   LEFT OUTER JOIN Sales.SpecialOfferProduct_inmem d ON d.ProductID=b.ProductID
   AND c.SpecialOfferID = d.SpecialOfferID
```

Note: The LEFT OUTER JOIN and UNION ALL that are used in the new query are now allowed in SQL Server 2016.

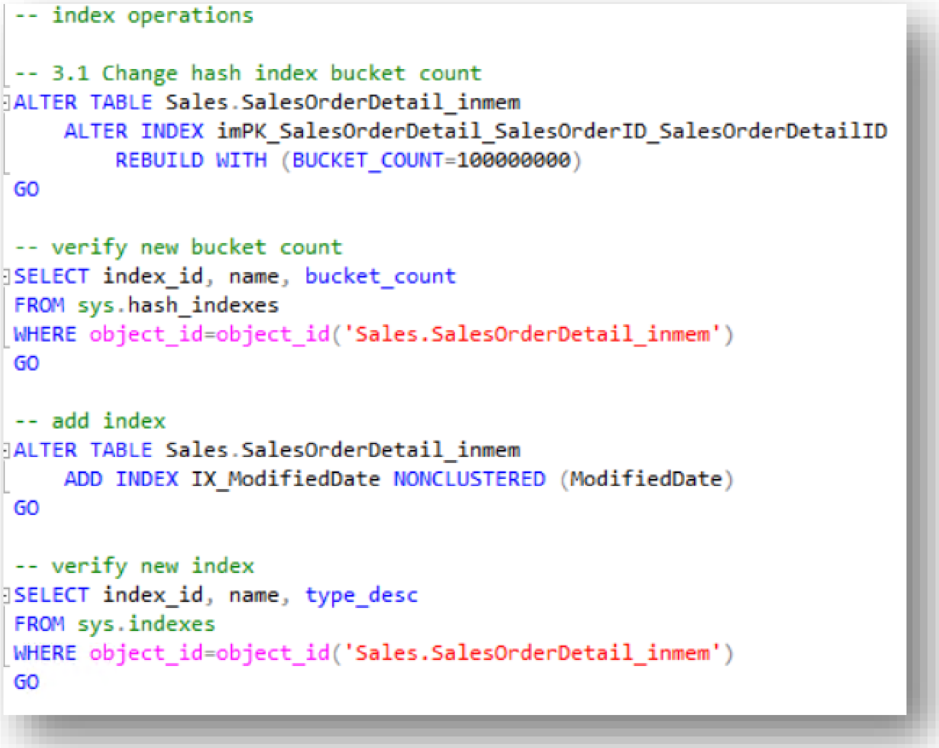
Explore new ALTER capabilities and impacts

In SQL Server 2014, once a memory-optimized table was created, if a change was needed it could only be dropped and recreated. With SQL Server 2016, ALTER TABLE statements on index and schema changes are now supported. However, in SQL Server 2016, ALTER TABLE is an offline operation.

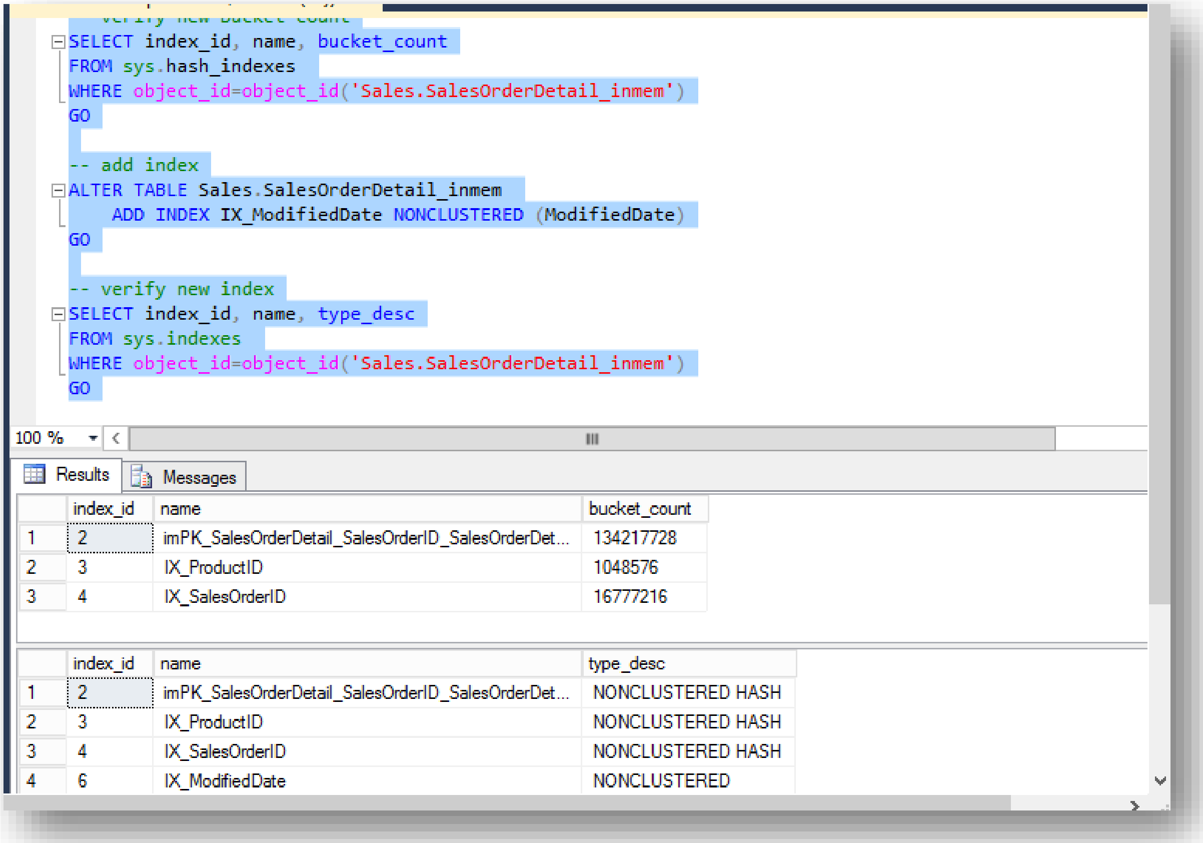
1. In the In-MemOLTP.sql file that you opened in the last part of the lab, there are several scripts that relate to altering indexes. Find the script marked with the comment:

   ```
   -- 3.1 Change hash index bucket count
   ```

2. Execute the four statements that change the bucket count, verify the new bucket count, add a new nonclustered index, and verify the new index.



```sql
-- index operations

-- 3.1 Change hash index bucket count
ALTER TABLE Sales.SalesOrderDetail_inmem
    ALTER INDEX imPK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID
        REBUILD WITH (BUCKET_COUNT=100000000)
GO

-- verify new bucket count
SELECT index_id, name, bucket_count
FROM sys.hash_indexes
WHERE object_id=object_id('Sales.SalesOrderDetail_inmem')
GO

-- add index
ALTER TABLE Sales.SalesOrderDetail_inmem
    ADD INDEX IX_ModifiedDate NONCLUSTERED (ModifiedDate)
GO

-- verify new index
SELECT index_id, name, type_desc
FROM sys.indexes
WHERE object_id=object_id('Sales.SalesOrderDetail_inmem')
GO
```



```sql
SELECT index_id, name, bucket_count
FROM sys.hash_indexes
WHERE object_id=object_id('Sales.SalesOrderDetail_inmem')
GO

-- add index
ALTER TABLE Sales.SalesOrderDetail_inmem
    ADD INDEX IX_ModifiedDate NONCLUSTERED (ModifiedDate)
GO

-- verify new index
SELECT index_id, name, type_desc
FROM sys.indexes
WHERE object_id=object_id('Sales.SalesOrderDetail_inmem')
GO
```

	index_id	name	bucket_count
1	2	imPK_SalesOrderDetail_SalesOrderID_SalesOrderDet...	134217728
2	3	IX_ProductID	1048576
3	4	IX_SalesOrderID	16777216

	index_id	name	type_desc
1	2	imPK_SalesOrderDetail_SalesOrderID_SalesOrderDet...	NONCLUSTERED HASH
2	3	IX_ProductID	NONCLUSTERED HASH
3	4	IX_SalesOrderID	NONCLUSTERED HASH
4	6	IX_ModifiedDate	NONCLUSTERED

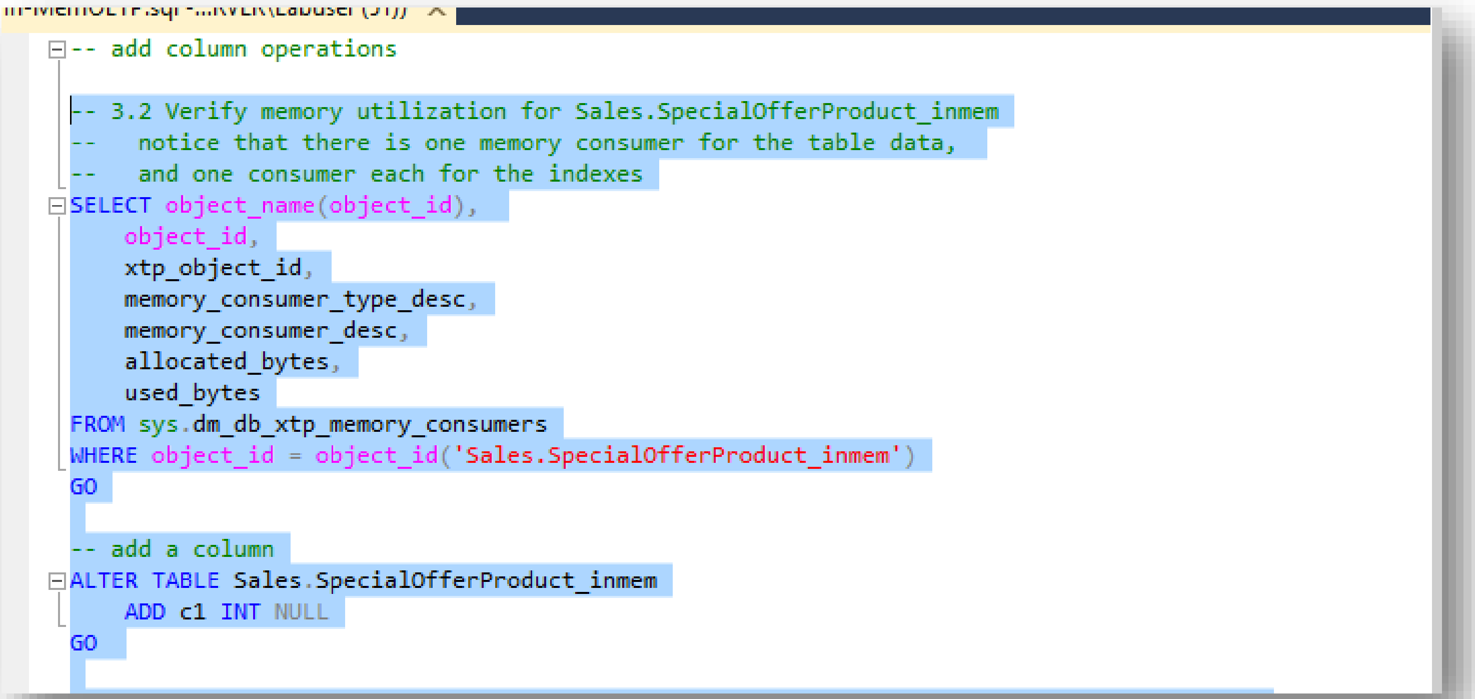
3. Observe the time taken for each of the operations—every ALTER for a memory-optimized table creates a full copy in memory. It is possible that there may be insufficient system memory resources available to run this operation. If you receive this error, just move on to the next step.

Add a column on memory-optimized table and view impact

4. Find the script marked with the comment:

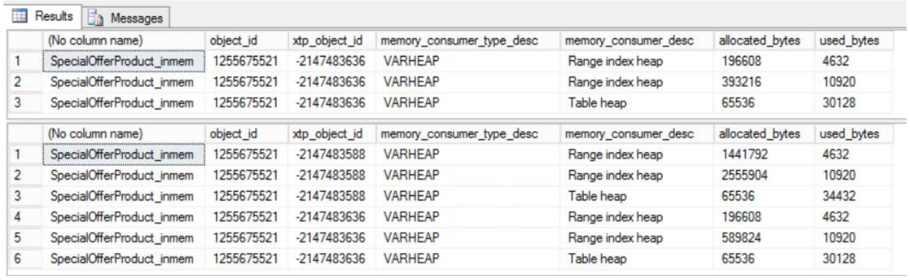
```
-- 3.2 Verify memory utilization for Sales.SpecialOfferProduct_inmem
```

5. Execute the four statements that verify the initial memory utilization for the table. Add a new column, verify the new memory utilization, and finally drop the new column.



```
-- add column operations

-- 3.2 Verify memory utilization for Sales.SpecialOfferProduct_inmem
--   notice that there is one memory consumer for the table data,
--   and one consumer each for the indexes
SELECT object_name(object_id),
    object_id,
    xtp_object_id,
    memory_consumer_type_desc,
    memory_consumer_desc,
    allocated_bytes,
    used_bytes
FROM sys.dm_db_xtp_memory_consumers
WHERE object_id = object_id('Sales.SpecialOfferProduct_inmem')
GO

-- add a column
ALTER TABLE Sales.SpecialOfferProduct_inmem
    ADD c1 INT NULL
GO
```

6. Notice that after the column was added, there are now double the memory consumers for the table data: two for each index. The old version of the table remains temporarily in memory until garbage collection can clean it up.



	(No column name)	object_id	xtp_object_id	memory_consumer_type_desc	memory_consumer_desc	allocated_bytes	used_bytes
1	SpecialOfferProduct_inmem	1255675521	-2147483636	VARHEAP	Range index heap	196608	4632
2	SpecialOfferProduct_inmem	1255675521	-2147483636	VARHEAP	Range index heap	393216	10920
3	SpecialOfferProduct_inmem	1255675521	-2147483636	VARHEAP	Table heap	65536	30128

	(No column name)	object_id	xtp_object_id	memory_consumer_type_desc	memory_consumer_desc	allocated_bytes	used_bytes
1	SpecialOfferProduct_inmem	1255675521	-2147483588	VARHEAP	Range index heap	1441792	4632
2	SpecialOfferProduct_inmem	1255675521	-2147483588	VARHEAP	Range index heap	2555904	10920
3	SpecialOfferProduct_inmem	1255675521	-2147483588	VARHEAP	Table heap	65536	34432
4	SpecialOfferProduct_inmem	1255675521	-2147483636	VARHEAP	Range index heap	196608	4632
5	SpecialOfferProduct_inmem	1255675521	-2147483636	VARHEAP	Range index heap	589824	10920
6	SpecialOfferProduct_inmem	1255675521	-2147483636	VARHEAP	Table heap	65536	30128

7. You can now close the lab environment.

Summary

In-Memory OLTP can significantly improve OLTP database application performance. In-Memory OLTP is a memory-optimized database engine integrated into the SQL Server engine, optimized for OLTP. In SQL Server 2016, several improvements have been made to In-Memory OLTP. The Transact-SQL surface area has been increased to make it easier to migrate database applications. Support for performing ALTER operations for memory-optimized tables and natively compiled stored procedures has been added, to make it easier to maintain applications.

Terms of use

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-ON LAB 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 FUNCTIONALITY 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 provisioned under Microsoft SQL Server 2016. 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.