# Prerequisites

* SQL Server 2014 CTP2 installation
* For performance testing, a server with similar specifications as your production environment. For guidelines on hardware for In-Memory OLTP, see the following blog post:

<http://blogs.technet.com/b/dataplatforminsider/archive/2013/08/01/hardware-considerations-for-in-memory-oltp-in-sql-server-2014.aspx>

# Installing the AdventureWorks In-Memory OLTP sample

1. Download the archive for the full backup of the AdventureWorks2012 database:
   1. Open the following link in your favorite brower: <http://msftdbprodsamples.codeplex.com/downloads/get/417885>
   2. When prompted to save the file, select a local folder on your to store the file, for example ‘c:\temp’
   3. This will download the file ‘AdventureWorks2012-Full Database Backup.zip’ to your local folder
2. Open the downloaded .zip archive ‘AdventureWorks2012-Full Database Backup.zip’ in Windows Explorer
3. Copy the .bak file from the .zip archive to a local folder, for example ‘c:\temp’
4. Restore the database backup using T-SQL or Management Studio
   1. Identify the target folder and filename for the data file, for example 'h:\DATA\AdventureWorks2012\_Data.mdf'
   2. Identify the target folder and filename for the log file, for example 'i:\DATA\AdventureWorks2012\_log.ldf'

The log file should be places on a different drive than the data file, and should have low latency for write operations, for example SSD

Example T-SQL script:

RESTORE DATABASE [AdventureWorks2012]

FROM DISK = N'C:\temp\AdventureWorks2012-Full Database Backup.bak' WITH FILE = 1,

MOVE N'AdventureWorks2012\_Data' TO N'h:\DATA\AdventureWorks2012\_Data.mdf',

MOVE N'AdventureWorks2012\_Log' TO N'i:\DATA\AdventureWorks2012\_log.ldf'

GO

1. Change the database owner to a login on your server, by running the following command in the query window of Management Studio:

ALTER AUTHORIZATION ON DATABASE::AdventureWorks2012 TO [*<NewLogin>*]

1. Update the value for the variable ‘checkpoint\_files\_location’ in the included script 'aw\_inmem\_sample.sql', to point to the target location for the In-Memory OLTP checkpoint files. The checkpoint files should be placed on a drive with fast sequential IO.
   1. Be sure to include the backslash ‘\’ as part of the path name
   2. Example:

:setvar checkpoint\_files\_location "h:\Databases\DATA\"

1. Execute the sample script, in one of two ways:
   1. Using the sqlcmd command-line utility. For example, for example by running the following command from the command-line prompt in the folder containing the script: ‘sqlcmd –S . –E –i aw\_inmem\_sample.sql’
   2. Using Management Studio:
      1. Open the script ‘aw\_inmem\_sample.sql’ in a query window
      2. Connect to the target server that
      3. Enable SQLCMD Mode, by clicking on ‘Query -> SQLCMD Mode’
      4. Click the button ‘Execute’ to run the script

# Structure of the sample

The sample creates new tables for products and sales orders, based on the existing tables in AdventureWorks. The schema of the new tables is similar to the existing tables, with a few differences. The most notable are: the new tables do not have foreign key or check constraints, and do not have computed columns.

Each table has an '\_inmem' and an '\_ondisk' variant with equivalent schemas, to allow a fair performance comparison.

Two new stored procedures, Sales.usp\_InsertSalesOrder and Sales.usp\_UpdateSalesOrderShipInfo, can be used to perform insert and update operations.

Note that the new memory-optimized tables used in the workload for performance comparison are fully durable and fully logged.

The new schema 'Demo' contains helper stored procedures to drive a demo workload. Demo.usp\_DemoInsertSalesOrders can be used to drive inserts and updates, and Demo.usp\_DemoReset is used to reset the demo by removing the inserted rows from the sales order tables.

Concretely, the In-Memory OLTP sample adds the following objects to AdventureWorks2012:

**Tables**

Sales.SalesOrderHeader\_inmem

Sales.SalesOrderHeader\_ondisk

Sales.SalesOrderDetail\_inmem

Sales.SalesOrderDetail\_ondisk

Sales.SpecialOffer\_inmem

Sales.SpecialOffer\_ondisk

Sales.ShoppingCartItem\_inmem

Production.Product\_inmem

Production.Product\_ondisk

Demo.DemoSalesOrderDetailSeed

**Stored Procedures**

Sales.usp\_InsertSalesOrder\_inmem

Sales.usp\_InsertSalesOrder\_ondisk

Sales.usp\_UpdateSalesOrderShipInfo\_inmem

Sales.usp\_UpdateSalesOrderShipInfo\_ondisk

Sales.usp\_UpdateSalesOrderShipInfo\_native

Demo.usp\_DemoInsertSalesOrders

Demo.usp\_DemoReset

Demo.usp\_DemoInitSeed

These tables and stored procedures showcase the T-SQL features that are supported with memory-optimized tables and indexes, and natively compiled stored procedures: the procedures Sales.usp\_InsertSalesOrder\_inmem and Sales.usp\_UpdateSalesOrderShipInfo\_native are natively compiled.

To see the performance benefit of using memory-optimized tables it is important to test at scale, executing inserts and updates concurrently. The ostress tool, described below, can be used to drive a concurrent workload.

# Performance Measurements using Ostress

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; 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 time taken for all threads to finish execution.

## Installing ostress

Ostress is installed as part of the RML Utilities; there is no standalone installation for ostress.

Installation steps:

1. Download and run the installation package for the RML utilities from the following location: <http://download.microsoft.com/download/0/a/4/0a41538e-2d57-40ff-ae85-ec4459f7cdaa/RMLSetup_AMD64.msi>
2. If there is a dialog box saying certain files are in use, click ‘Continue’

## Running ostress

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.

To open the RML Cmd Prompt follow these instructions:

In Windows Server 2012 and in Windows 8 and 8.1, open the start menu by clicking the Windows key, and type ‘rml’. Click on “RML Cmd Prompt”, which will be in the list of search results.

The command-line options for ostress can be seen when simply running ostress.exe without any options. The main options to consider for running ostress with this sample are:

-S name of Microsoft SQL Server server to connect to

-E use Windows authentication to connect (default); if you use SQL Server authentication, use the options –U and –P to specify the username and password, respectively

-d name of the database, for this example AdventureWorks2012

-Q the T-SQL statement to be executed

-n number of connections processing each input file/query

The main stored procedure for running the demo workload is Demo.usp\_DemoInsertSalesOrders. This stored procedure inserts a number of orders into the database. The procedure has the following parameters:

|  |  |  |
| --- | --- | --- |
| @use\_inmem | bit | 1 (default) means the orders are inserted into the memory-optimized tables Sales.SalesOrderHeader\_inmem and Sales.SalesOrderDetail\_inmem  0 means the orders are inserted into the disk-based tables Sales.SalesOrderHeader\_ondisk and Sales.SalesOrderDetail\_ondisk |
| @order\_count | int | The number of orders to be inserted. The orders are inserted in a loop, with each insert having its own transaction.  Default = 100000 |
| @include\_update | bit | 0 (default) – perform only insert operations.  1 – each insert operation is followed by an update operation |

The ostress tool is used to execute the procedure Demo.usp\_DemoInsertSalesOrders in parallel, to simulate clients inserting order concurrently.

Resetting the demo is achieved by executing Demo.usp\_DemoReset. This procedure deletes the rows in the memory-optimized tables, truncates the disk-based tables, and executes a database checkpoint. Best practice is to reset the demo after every test run, to avoid running out of memory.

### Samples

The following command connects to the default instance and executes the procedure Demo.usp\_DemoInsertSalesOrders to insert 10,000,000 sales orders in disk-based tables, using 100 threads, each inserting 100,000 rows.

ostress.exe -S. -E -dAdventureWorks2012 -Q"EXEC Demo.usp\_DemoInsertSalesOrders @use\_inmem = 0, @order\_count=100000" –n100

The following will do the same for memory-optimized tables:

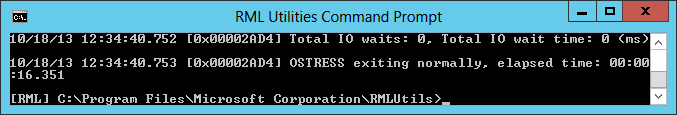
ostress.exe -S. -E -dAdventureWorks2012 -Q"EXEC Demo.usp\_DemoInsertSalesOrders @use\_inmem = 1, @order\_count=100000" –n100

The following resets the demo:

ostress.exe -S. -E -dAdventureWorks2012 -Q"EXEC Demo.usp\_DemoReset"

## Interpreting Results

When ostress finishes execution, it reports the elapsed time for the test run. The following screenshot shows the result of a run inserting a total of 1,000,000 orders, with 100 threads inserting 10,000 orders each. The test was run on a 2-socket mid-range server.



The following command was used to start the test:

ostress.exe -S. -E -dAdventureWorks2012 -Q"EXEC Demo.usp\_DemoInsertSalesOrders @use\_inmem = 1, @order\_count=10000" –n100

To compare performance for disk-based and memory-optimized tables on your system, you can compare the elapse time for the test run using disk-based tables with the elapse time for the test run using memory-optimized tables.

The below table show the results from one test run using the CTP2 build on a mid-range server with two CPUs, which 12 physical (24 logical) cores each. The data files were laid out on one SSD drive, and the log file was on another SSD drive.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Disk-based tables | Memory-optimized tables | Performance difference |
| 10,000,000 order inserts using 100 threads, with 100,000 inserts each | 1:31:11 | 0:01:49 | 50X |

The main factor in the performance difference between memory-optimized tables and disk-based tables in this test is the fact that when using disk-based tables, SQL Server could not fully utilize the CPU (during the run it was at around 5%). The reason was latch contention: concurrent transactions are attempting to write to the same data page; latches are used to ensure only one transaction at a time can write to a page. The In-Memory OLTP engine is latch-free; in fact, there is no concept of pages. Thus, concurrent transactions do not block each other’s inserts, thus enabling SQL Server to fully utilize the CPU – during the run, CPU was constantly at 100%.

Test results will typically vary with hardware, and also the level of concurrency used in the test run. A couple of things to look for if the results are not as expected:

* Number of concurrent transactions. When running the workload on a single thread, performance gain with In-Memory OLTP will likely be less than 2X. Contention is only a big problem if there is a high level of concurrency.
* Low number of cores available to SQL Server. This means there will be a low level of concurrency in the system, as there can only be as many concurrently executing transactions as there are cores available to SQL.
  + Symptom: if the CPU utilization is high when running the workload on disk-based tables, this means there is not a lot of contention, pointing to a lack of concurrency.
* Speed of the log drive. If the log drive cannot keep up with the transaction in the system, the workload becomes bottlenecked on log IO. Although logging is more efficient with In-Memory OLTP, if log IO is a bottleneck, the potential performance gain is limited.
  + Symptom: if the CPU utilization is not close to 100% or is very spiky when running the workload on memory-optimized tables, it is possible there is a log IO bottleneck. This can be confirmed by opening Resource Monitor and looking at the queue length for the log drive.