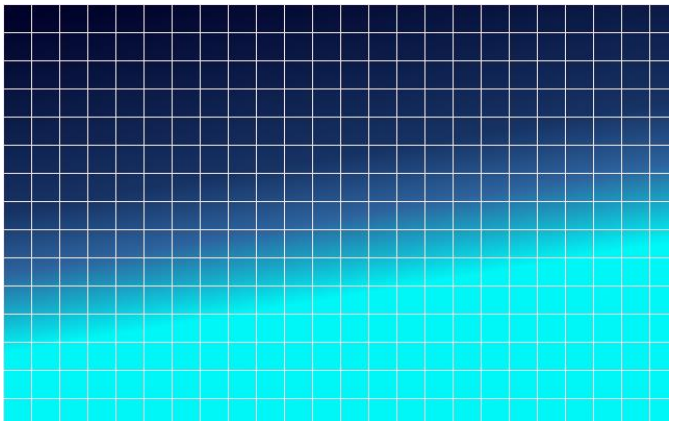
In-Memory OLTP Simulator



In-Memory OLTP Simulator: The Experiment

Technical Article

Version: 1.1

Writer: Artemakis Artemiou (Microsoft Data Platform MVP).

Technical Reviewer: Andreas Nestorides

Published: March 2016

Applies to: In-Memory OLTP Simulator, SQL Server 2014 (or later)

Summary:
In-Memory OLTP Simulator is a software tool that allows the user to easily simulate virtually any workload against the powerful In-Memory OLTP Engine of Microsoft SQL Server. This paper presents a series of simulations for a specific scenario against different workload sizes. The paper discusses the findings of these simulations by describing how the entire process was designed using In-Memory OLTP Simulator, as well as by observing and analyzing the performance trends after executing the different versions of the scenario against SQL Server's In-Memory OLTP Engine.

Copyright

Contents

1. Introduction

1.1 The In-Memory OLTP Engine in SQL Server

In-Memory OLTP, also known as 'Hekaton' and 'In-Memory Optimization', is Microsoft's latest in-memory processing technology. In-Memory OLTP is seamlessly integrated into SQL Server's Database Engine and it is optimized for Online Transaction Processing (OLTP) operations.

In-Memory OLTP introduces new data structures for optimizing the performance of OLTP workloads. These data structures are called 'Memory-Optimized' tables and along with 'Natively-Compiled Stored Procedures' and other related constructs, offer a significant performance boost when it comes to processing large amounts of information, especially in data environments with high levels of concurrency.

In-Memory OLTP performs better to certain types of workload. These types are categorized and commended by Microsoft as per the below table:

Workload Type	Examples	Main Benefits of In-Memory OLTP
High Data Insert Rate	• Smart Metering • System Telemetry	• Eliminate contention • Minimize I/O logging
Read Performance and Scale	• Social Network Browsing	• Eliminate contention • Efficient data retrieval • Minimize code execution time • CPU efficiency for scale
Compute Heavy Data Processing	• Manufacturing supply chains or retailers	• Eliminate contention • Minimize code execution time • Efficient data processing
Low Latency	• Online Gaming Platforms • Capital Markets	• Eliminate contention • Minimize code execution time • Efficient data retrieval
Session State Management	• Managing sessions (i.e. user requests, etc.) for heavily-visited websites	• Eliminate contention • Efficient data retrieval • Optional I/O reduction/removal

Table 1.1: Best-Suited Workload Types for In-Memory OLTP.

1.2 Scope of this Experiment

The scope of this experiment is to present the process of simulating a heavy OLTP workload using the software tool 'In-Memory OLTP Simulator', running it against SQL Server's In-Memory OLTP Engine, checking possible performance gains, and observing potential trends on performance as the workload

size increases.

1.3 Hardware Used

The hardware used for this experiment was a home desktop PC with the following characteristics:

Item	Value
Processor	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 3392 Mhz, 4 Core(s), 8 Logical Processor(s)
System Type	x64-based
Total Physical Memory (RAM)	16.0 GB DDR3
Total Virtual Memory	18.3 GB
Disk Drive	1TB 32MB Cache SATA 7200 rpm
OS Name	Microsoft Windows 8.1 Enterprise x64

Table 1.2: Hardware Used for the Experiment.

1.4 SQL Server Version Used

The simulations in this paper were executed against the In-Memory OLTP Engine in SQL Server 2016 CTP 3.3.

The In-Memory OLTP Engine (In-Memory Optimization) originally shipped with SQL Server 2014 so this paper's examples/simulations can be applied to SQL Server 2014 or later.

Note: If you want to execute the paper's examples in SQL Server 2014, you need to make sure that all indexes on character columns must use a *_BIN2 collation (i.e. Latin1_General_100_BIN2) or else you will get the below error message:

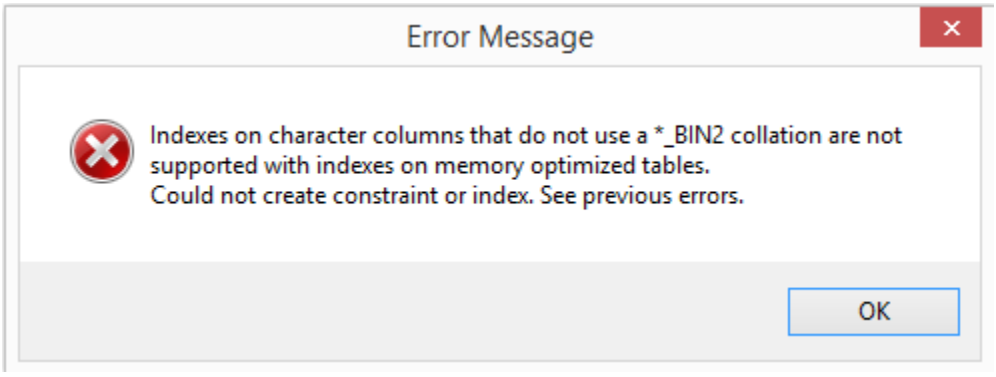


Figure 1.1: Error Message When you Try to Set an Index on a Character Column in SQL Server 2014 Without Using a *_BIN2 Collation.

The above limitation does not exist in SQL Server 2016 or later.

2. In-Memory OLTP Simulator

2.1 What is In-Memory OLTP Simulator

In-Memory OLTP Simulator is a software tool that enables the IT Professional and Developer to easily test the powerful In-Memory OLTP Engine in SQL Server (2014 or later) with different simulations and against different workloads with the use of 'simulation scenarios'. A simulation scenario features three execution modes which are: (i) Disk-Based, (ii) Memory-Optimized, and (iii) Memory-Optimized with Natively-Compiled Stored Procedure. Each mode uses its own tables and related data structures. Statistics are provided live with the execution of each mode and after a full simulation cycle is completed, a set of rich analytics is provided for further analysis. Additionally, In-Memory OLTP Simulator provides stability mechanisms via real-time monitoring of the database server that hosts its database, and if certain thresholds are reached (i.e. CPU or RAM utilization) then the simulation is automatically stopped.

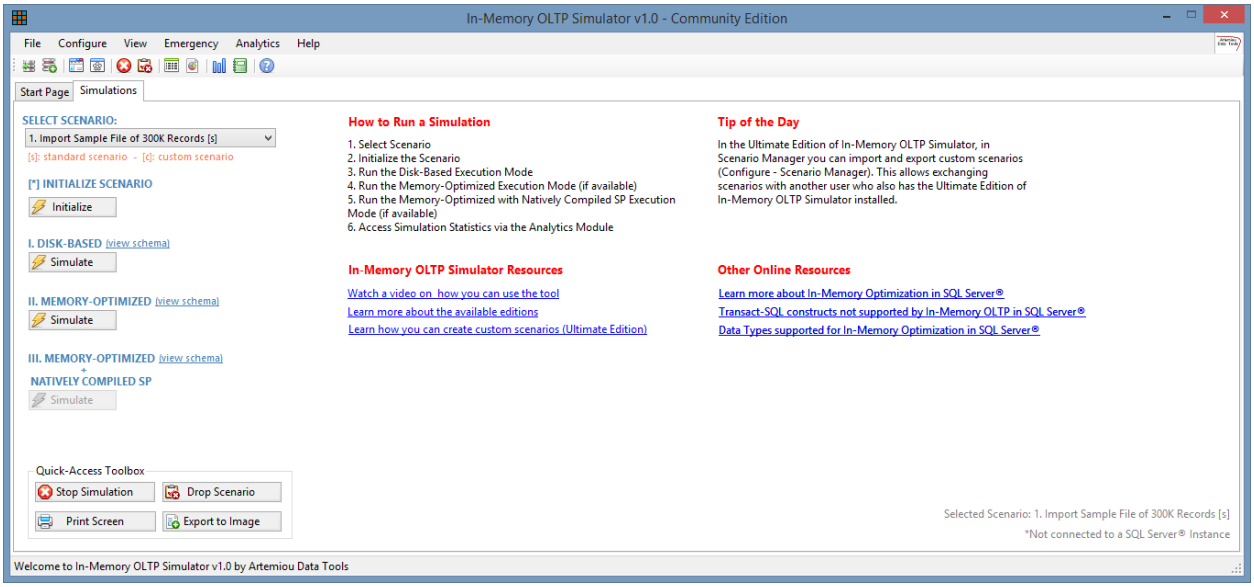


Figure 2.1: The Simulation Page in In-Memory OLTP Simulator.

In the Ultimate Edition of the tool, among other, the user can load data from different sources (i.e. from Production DBMS instances using Linked Servers) into the Simulator's database and simulate real-life workloads with the use of custom scenarios as well as get in-depth analytics on performance when utilizing Memory-Optimized Tables and Natively-Compiled Stored Procedures in SQL Server. In addition to using In-Memory OLTP Simulator for benchmarking and performance analysis, the user can also use it for resource-sizing as it provides real-time statistics on resource usage (i.e. CPU and RAM) while running the simulations.

In-Memory OLTP Simulator was developed by Senior SQL Server Architect and Microsoft Data Platform MVP Artemakis Artemiou within the context of his initiative named 'Artemiou Data Tools'.

2.2 Available Editions

In-Memory OLTP Simulator is available in two editions: (i) The Community Edition, and (ii) The Ultimate Edition. The Community Edition is free and provides basic functionality, whereas in addition to the basic functionality, the Ultimate Edition provides more advanced features, having as a highlight the ability to create and process custom scenarios.

The below table compares the features of the two available editions of In-Memory OLTP Simulator.

Feature	Ultimate Edition	Community Edition
Run Standard Scenarios	Yes	Yes
View Current Usage Statistics	Yes	Yes
Change Number of Records for Standard Scenarios	Yes	Yes
View Active Scenario Definition	Yes	Yes
View Current Resource Usage	Yes	Yes
Stop Simulation	Yes	Yes
Drop Scenario	Yes	Yes
Drop All Scenarios	Yes	Yes
Take Simulator Database Offline	Yes	Yes
Bring Simulator Database Online	Yes	Yes
Simulation Statistics	Yes	Yes
Export Statistics to Text File	Yes	Yes
Export Graphs to Image File	Yes	Yes
Print Statistics	Yes	Yes
View Schema of Tables	Yes	Yes
View Result of Simulation	Yes	Yes
Set Emergency Thresholds	Yes	Yes
Real-Time Resource Monitoring	Yes	Yes
Generate and Use Multiple Simulation Databases	**Yes**	No
Create, Modify and Run Custom Scenarios	**Yes**	No
Export Custom Scenarios	**Yes**	No
Import Custom Scenarios	**Yes**	No
Executive Report	**Yes**	No
Print Report	**Yes**	No
Export Report	**Yes**	No
Send Report in Text Format by Email	**Yes**	No

Table 2.1: Feature Comparison of In-Memory OLTP Simulator's Editions.

2.3 Simulation Engine

If a stopwatch is used during a simulation run using In-Memory OLTP Simulator, you will notice that the times do not exactly match when compared to the times the Simulator calculates. However, this is natural as the way In-Memory OLTP Simulator calculates the simulation time for each execution mode is very specific in order to be as accurate as possible. This is explained below.

First of all, as you can see in the Simulation Page (see Figure 2.2), the scenario initialization step is a separate task. The initialization process creates the required data structures as instructed by the selected scenario's definition and prepares the data to be processed. As these tasks are global for all execution modes, their execution time is irrelevant as In-Memory OLTP Simulator targets at comparing the actual execution times of each mode only.

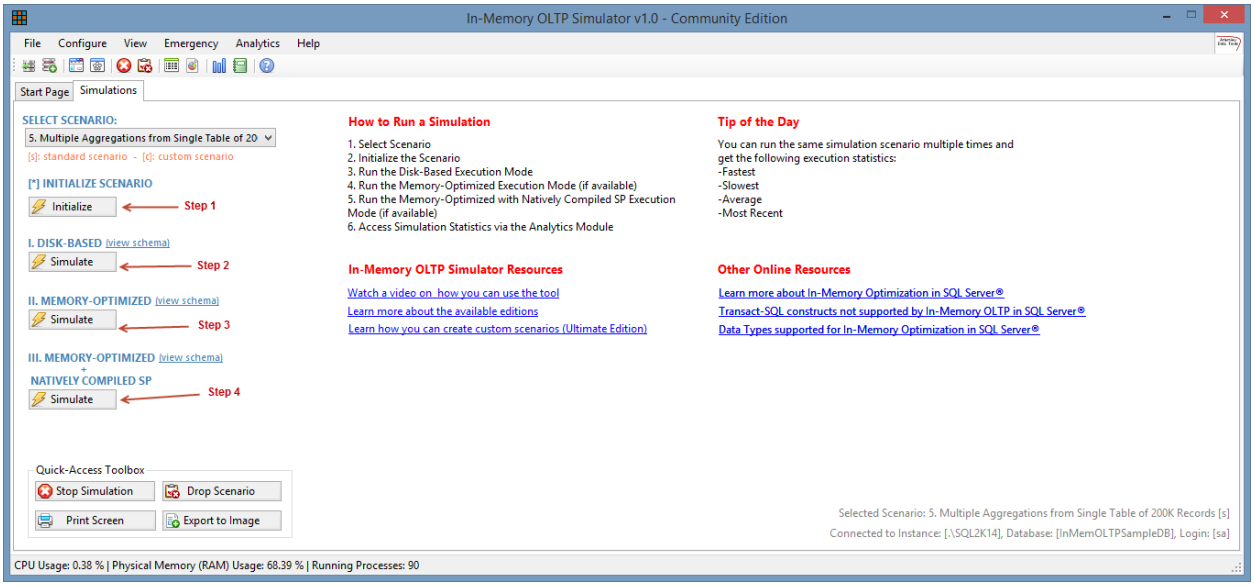


Figure 2.2: The Simulation Page in In-Memory OLTP Simulator - A four-Step Process.

Another important aspect that needs to be taken into consideration is caching/buffering. Every time a scenario mode is executed, In-Memory OLTP Simulator clears the buffers and plan cache in order for SQL statements not to be reused from the cache, neither data to be fetched from the data cache. Even though this task runs in the background during each scenario mode's execution, it is not taken into consideration in the mode's execution time. This is because the task is used for preparing the modes for the most accurate possible execution time measurement. The clearing of buffers is considered as a built-in process and you do not need to take any additional actions.

Someone might argue that it could be more realistic not to clear the cache. However, as the type of workload patterns that may benefit the most by using In-Memory Optimization can be found in fast-changing data environments with large volumes of data, it was considered that it would be more realistic to clear the cache each time a mode of a scenario is executed in order to measure and compare

the pure computation time for each mode.

The diagram below illustrates all the above:

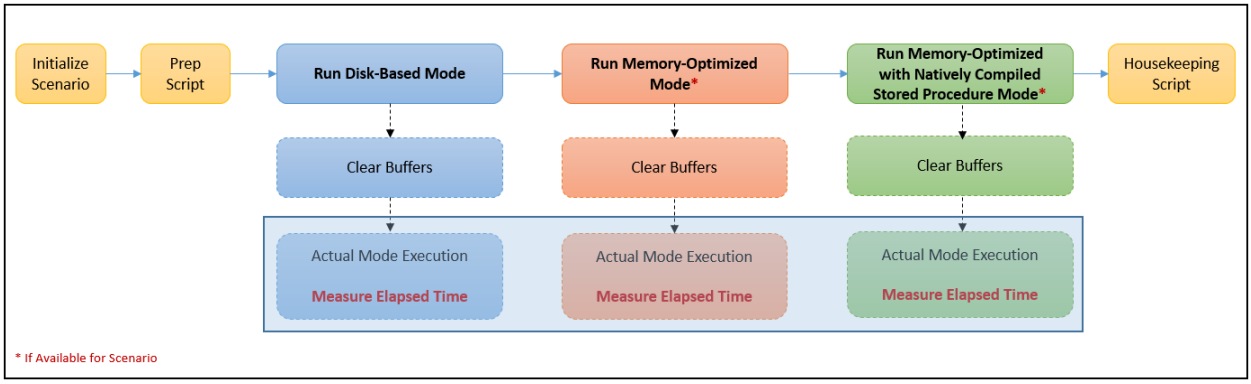


Figure 2.3: The Simulation Engine of In-Memory OLTP Simulator.

Important: Every time you use In-Memory OLTP Simulator, you need to take into consideration the overall state of the Test Database Server on which you run the simulations. If during a simulation there is other significant disk activity, then this will affect the simulation results. Even though you cannot fully eliminate this factor, if there is another process running at the time performing heavy I/O, then it is recommended to wait for it to complete prior to start running simulations using In-Memory OLTP Simulator.

Note: If you execute the scenario in SQL Server Management Studio (SSMS) you might occasionally notice a slightly better performance for all of its modes of execution. This could happen because In-Memory OLTP Simulator is a .NET application which uses certain drivers for connecting to a SQL Server Database Engine and this might introduce some overheads, even though all steps have been taken to minimize any overhead that the abovementioned case may cause.

2.4 System Stability Features

It is strongly recommended never to host In-Memory OLTP Simulator's database on Production SQL Server instances or Production servers. As the nature of the tool is to benchmark extreme simulations, you would risk the stability of your Production environment if you used it to host In-Memory OLTP Simulator's database there. You should use a Test server instead and in case you want to simulate Production workload you can copy the target data (i.e. via a linked server, SSIS, etc.) to the tables that will be created in Simulator's database based on the definition of the custom scenario (custom scenarios can only be created in the Ultimate Edition of In-Memory OLTP Simulator). Note that during the design of a custom scenario, in the "Data Population" section you can set the process that will either fetch data from other data sources or just generate sample data.

If you do not wish to use two different machines for running In-Memory OLTP Simulator (Database Server and Client PC), you can run In-Memory OLTP Simulator directly on the dedicated database server (not Production) that will be used for hosting the simulator's database(s).

In addition to the above considerations, as the nature of In-Memory Optimization involves allocating large amounts of RAM for storing multiple versions of the Memory-Optimized data, this means that you also need to take into consideration possible hardware limitations of your server when it is used for in-memory optimization.

However, as In-Memory OLTP Simulator is an exploratory third-party tool for showcasing the computational benefits of In-Memory Optimization for certain types of workloads, it offers two types of mechanisms for ensuring that you can cope with possible issues in the case where the memory amount that needs to be allocated for a scenario is larger than the available on your database server. These mechanisms are: (i) **Emergency Thresholds**, and (ii) **Emergency Actions** for manual intervention by the user.

2.4.1 Emergency Thresholds

In-Memory OLTP Simulator allows you to set the following two "emergency thresholds": one for the maximum allowed memory utilization percentage and another for the maximum allowed CPU utilization percentage. You can set these values using the following dialog:

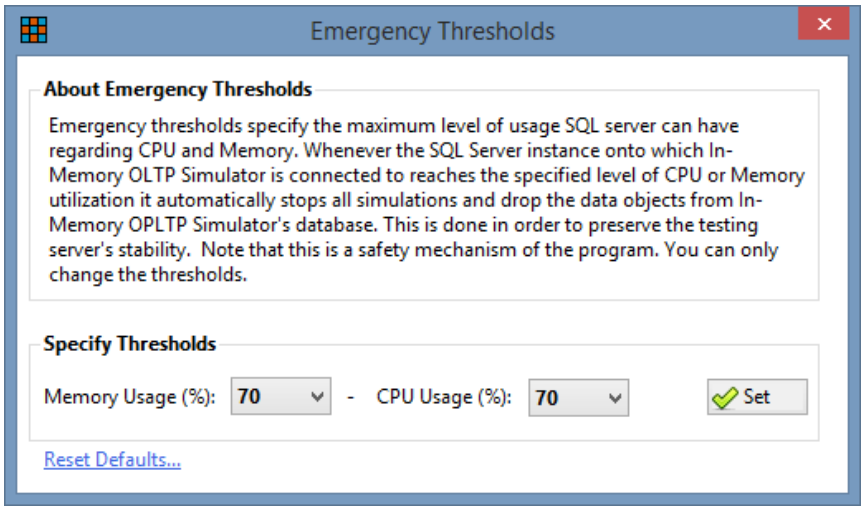


Figure 2.4: Emergency Thresholds Dialog.

You can launch the Emergency Thresholds Dialog with any of the following methods:

- By navigating to "**Configure - Emergency Thresholds**"
- By clicking the following quick access button: 
- By using the key combination: **Ctrl + Shift + E**

During a scenario's initialization and execution, the Simulator's engine constantly checks the memory and CPU utilization of the database server that hosts the Simulator's database. If one of those metrics reaches the maximum allowed percentage, then the engine of In-Memory OLTP Simulator

automatically cancels the scenario's execution and drops the scenario from the Simulator's database. This is done in order to ensure stability of the SQL Server instance where the In-Memory OLTP Simulator database is hosted on.

The default emergency thresholds are set to **70%** for both **Memory** and **CPU** utilization.

The allowed value range for both Memory and CPU utilization percentage is: **50% - 90%**

2.4.2. Emergency Actions

The available manual emergency actions are:

- Stop Simulation
- Drop Scenario
- Drop All Scenarios
- Take Simulator Database Offline
- Bring Simulator Database Online

All emergency actions can be accessed from the “Emergency” menu or by using the key shortcuts as below:

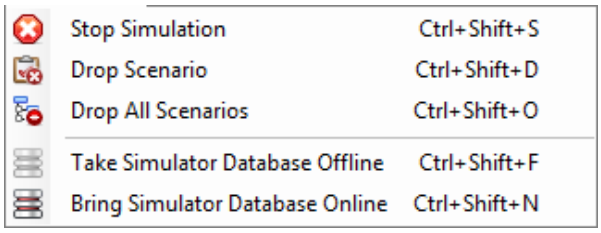


Figure 2.5: The “Emergency Actions” Menu.

The “**Stop Simulation**” action in the Emergency menu enables you to stop the simulation that is currently running. Even though the emergency thresholds are being constantly monitored by the engine of In-Memory OLTP Simulator and certain preventive actions are being taken automatically in order to prevent a possible system instability, with the “Stop Simulation” action you can manually stop the execution of the simulation scenario for any reason.

The “**Drop Scenario**” action can only be used when a simulation is not in progress. The “Drop Scenario” action drops the scenario’s data structures (i.e. tables, stored procedures, etc.) from the simulator’s database in order to allow the server’s Operating System to claim back resources previously allocated to these data structures.

The “**Drop All Scenarios**” action can only be used when a simulation is not in progress. This action drops all scenarios’ data structures (i.e. tables, stored procedures, etc.) from the simulator’s database in order

to allow the server's Operating System to claim back any resources previously allocated to the data structures of all In-Memory OLTP Simulator's scenarios that were initialized at least once.

The "**Take Simulator Database Offline**" action takes In-Memory OLTP Simulator's database offline in order to free up any resources which might have been allocated to the data structures of any In-Memory OLTP Simulator's scenario.

Note: After the "**Take Simulator Database Offline**" action is performed, in order to be able to use In-Memory OLTP Simulator again you will need to bring back the database online either from within the In-Memory OLTP Simulator or from within SQL Server Management Studio.

If for any reason you had to take the Simulator's database offline (i.e. for releasing resources back to the Operating System), you can try bringing it back online using the "**Bring Simulator Database Online**" function, in order to be able to use again the Simulator.

Note: If for any reason it is not possible to bring the Simulator's database back online, you can try bringing it online using SQL Server Management Studio. If In-Memory OLTP Simulator's database is successfully brought back online from within Simulator, then the connection to the database is restored as well. In the opposite case, after bringing the database back online using SQL Server Management Studio, you will need to reconnect to the database via "Configure – Connect to Database".

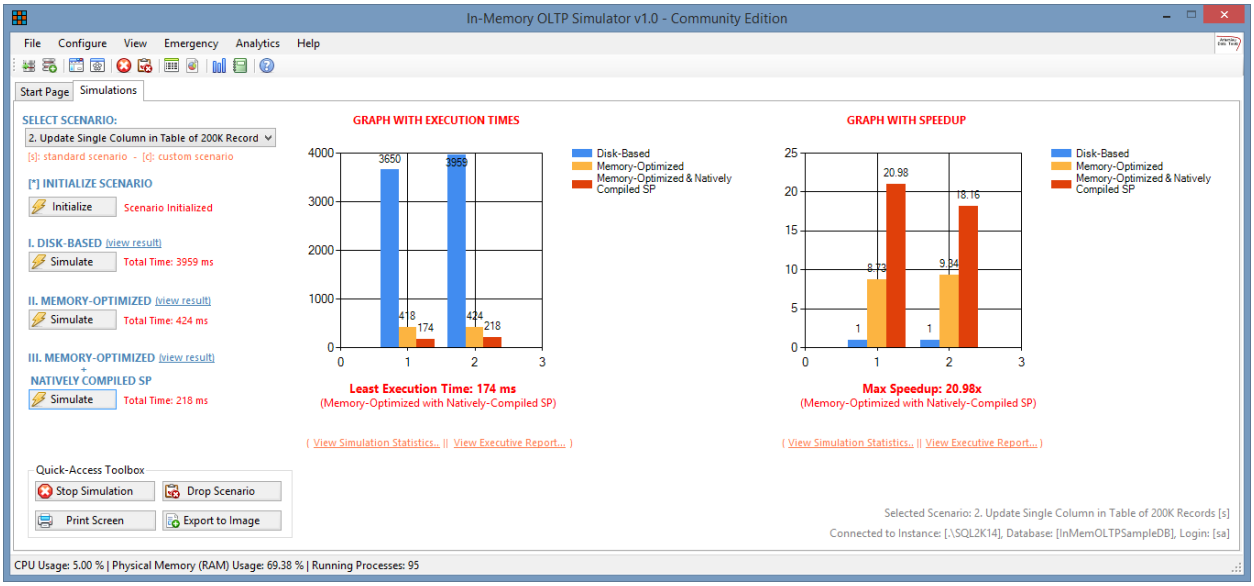


Figure 2.6: The Simulation Page.

3. The Scenario

3.1 Business Requirement

The scenario features a fictitious retail sales company who celebrates its 15 years of operations. Within the context of the celebrations, the Management has decided to update the discount percentage for all its customers by 10% as well as generate a report with all the customers that after the update have a discount percentage of 20% or more in order to offer them free shipping for their next purchase.

Based, on the above, the IT department of the company needs to update the customers table in order to reflect the Management's decision as well as generate the report.

As the purpose of this paper is to show the trends on performance over different workload volumes, the main scenario is broken down to 6 individual scenarios. All the scenarios follow the same business requirement and thus have the same logic. The only difference between the scenarios is the total number of records (customers) that need to be processed. The number of records per scenario is provided in the below table.

Scenario #	Total Number of Records to be Processed
Scenario 1	100 thousand records
Scenario 2	300 thousand records
Scenario 3	500 thousand records
Scenario 4	1 million records
Scenario 5	3 million records
Scenario 6	5 million records

Table 3.1: Workload Scenarios – Number of Records.

3.2 Source Tables

The source tables are located on the original SQL instance from which In-Memory OLTP Simulator will retrieve them during the scenarios' execution for data preparation purposes. The data will be retrieved during the execution of the 'Data Population' step for each scenario and will populate local tables that will be utilized by the three modes of execution in each scenario (Disk-Based, Memory-Optimized, and Memory-Optimized with Natively-Compiled Stored Procedure). The data will be retrieved using a linked server.

The below screenshot illustrates the source table which contains the original data.

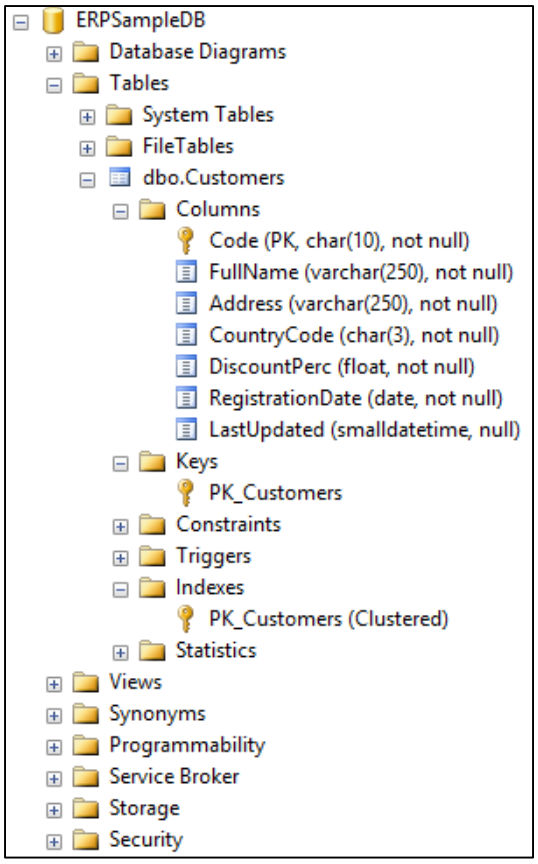


Figure 3.1: Source Table.

3.3 Scenario Analysis

Prior to start designing the scenario using In-Memory OLTP Simulator, a proper analysis must be performed in order to conclude on certain aspects of the custom scenario such as: tables to be created, data population process, index selection, etc.

Note that for the scenarios presented in this paper, all three In-Memory OLTP Simulator execution modes will be used.

Workload

As shown in Table 3.1, the experiment presented in this paper presents the simulation of six scenarios using In-Memory OLTP Simulator. The only difference between the scenarios is the number of records to be processed having as a maximum number 5 million records. The actual business logic for all the scenarios is based on the below pseudocode:

```
UPDATE Customers_Table
SET Discount Percentage = [DiscountPerc]+10.0
   ,[LastUpdated]= Get Current Timestamp

INSERT INTO Report_Table
SELECT ALL Customers FROM Customers_Table WITH discountPerc >= 20
```

Listing 3.1: Pseudocode for Implementing the Business Logic.

Tables

There is only one table in the original source ('Customers' table) so it is quite clear that we will create three respective tables, one for each mode, for each scenario. So for example, we can create the tables: **d_Customers** (for Disk-Based mode), **m_Customers** (for Memory-Optimized mode) and **n_Customers** (for Memory-Optimized with Natively-Compiled Stored Procedure mode). These three tables must be **durable**, meaning that their data will be permanent.

Similarly three more **durable** tables will be created for hosting the report results. These tables will be: **d_Report** (Disk-Based mode), **m_Report** (Memory-Optimized mode), and **n_Report** (Memory-Optimized with Natively-Compiled Stored Procedure mode).

Index Selection

As we can see in the source table in Figure 3.1, the primary key is the customer **code** so we will set the same key in the three corresponding customer tables in our scenario (d_Customers, m_Customers and n_Customers).

For d_Customers a clustered index will be automatically set for the 'code' column as it will be set as the primary key. For m_customers and n_customers we will set 'code' as primary key and create two Nonclustered Hash indexes (for each workload size, the proper 'bucket count' value needs to be used).

The above index selections cover the case of the primary key. However, in the pseudocode we can see that for generating the report, the column 'DiscountPerc' is also used. The question is if we need to add an index or not for this column too. As the need (or not) for an index is always related to the workload type, several options were tested for 'DiscountPerc'. By studying the execution times for the different options as well as the actual execution plans, it was decided that it is better not to use an index for 'DiscountPerc' in any of the three tables because the cost for maintaining the index (i.e. during the update operation) is higher than any benefits the index seeks have to contribute to the overall process.

At this point, it must be noted that the design for each execution mode assumes the best possible setup in order for the simulation to be 100% realistic. To this end, the most optimal design is used towards this goal.

Data Population

The data population script must populate d_Customers, m_Customers and n_Customers tables with the original data found in the source using a linked server. This is the initial data that will be later updated by the three execution modes, thus applying the business logic. As described in different sections in this paper, this process is not timed because its purpose is to set the initial/raw data for all active execution modes.

3.4 Custom Scenario Design Using In-Memory OLTP Simulator

Now that the business requirement is clear and the scenario analysis is completed, the next step is to create the custom scenarios in In-Memory OLTP Simulator in order to run the simulation and get the execution statistics, along with the answer to the business requirement (note that in order to be able to create custom scenarios in In-Memory OLTP Simulator, you need the Ultimate Edition of the tool).

A custom scenario in In-Memory OLTP Simulator consists of the following entities:

- **Metadata**: Basic information about the scenario like description, which execution modes it supports, etc. This is the page where you select which execution modes will be enabled. Disk-Based execution mode is always enabled as it serves as the baseline.
- **Disk-Based Tables**: The definition for the Disk-Based table(s). These are the traditional RDBMS's tables.
- **Mem-Opt Tables**: The definition for the Memory-Optimized table(s).
- **Mem-Opt Tables for Native SP**: The definition for the Memory-Optimized tables used in the Memory-Optimized with Natively-Compiled Stored Procedure execution mode.
- **Data Population Script**: This is the script that prepares the workload for the scenario. The data can be either sample data (like in the case of the standard scenarios) or data from other sources (i.e. Production) fetched into the Simulator's database from within In-Memory OLTP Simulator with the use of Linked Servers or from outside In-Memory OLTP Simulator using SQL Server technologies like SSIS/Data Export tools.
- **Prep Script**: This script can include preparatory SQL statements that run prior to the execution of each new simulation run. The Prep Script's execution time is not taken into consideration in the benchmarking process as it is out of the scope of In-Memory OLTP Simulator's philosophy.
- **Disk-Based SP**: The stored procedure that contains the scenario's logic for the Disk-Based execution mode (uses the Disk-Based Tables).
- **Mem-Opt SP**: The stored procedure that contains the scenario's logic for the Memory-Optimized execution mode (uses the Mem-Opt Tables).
- **Natively-Compiled SP**: The stored procedure that contains the scenario's logic for the Memory-Optimized with Natively-Compiled Stored Procedure execution mode (uses the Mem-Opt Tables for Native SP).
- **Execution Script**: The calls to the three stored procedures along with the input parameters (where available).
- **Housekeeping Script**: Post-execution script for maintenance purposes that runs in the end of each simulation run.

Note: As described in subsection 3.1 (Business Requirement), the purpose of this paper is to show the trends on performance over different workload volumes for the same In-Memory OLTP Simulator's scenario. Based on this fact and Table 3.1, the same scenario is repeated six times having however each time a different number of records to process. Below you will be presented with the design of the custom scenario(s) in In-Memory OLTP Simulator. For purposes of simplicity and in order to avoid repetition, only the design of Scenario 6 in Table 3.1 (records to process: 5 million) will be analytically presented.

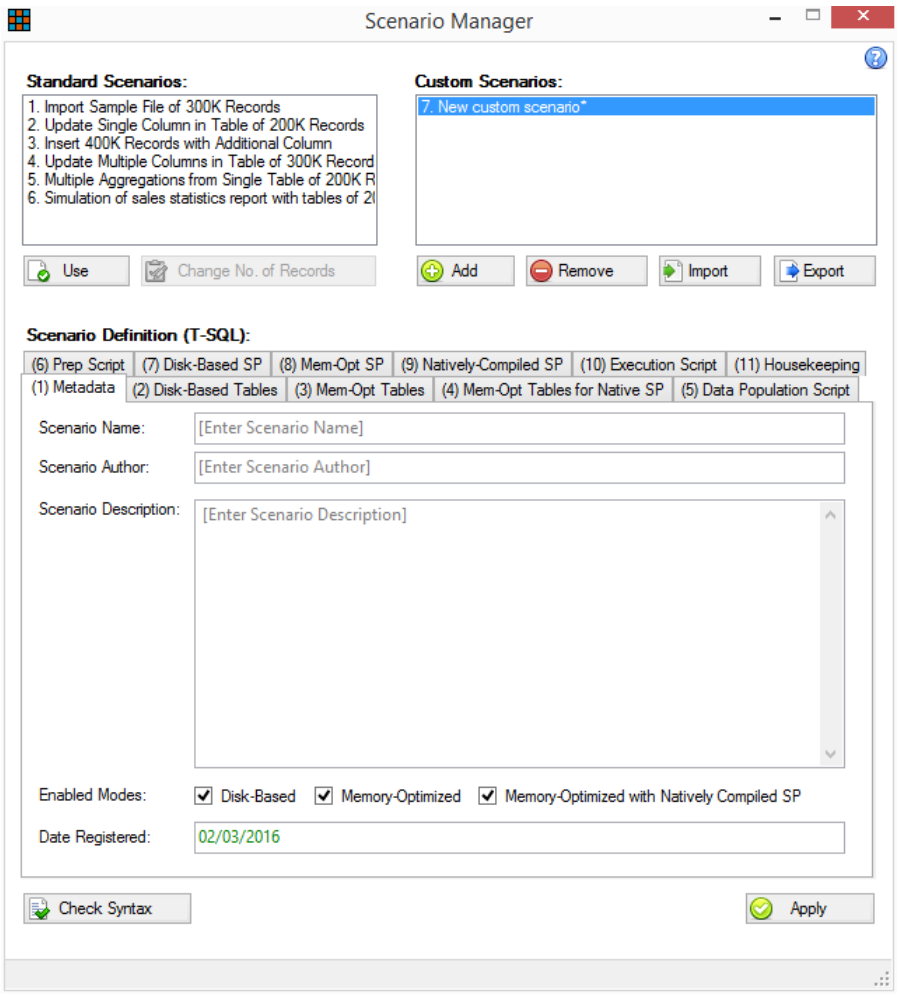


Figure 3.2: View of In-Memory OLTP Simulator's Scenario Manager (Custom scenarios can be created only in the Ultimate Edition).

The setup of the custom scenario is shown below. The name of the scenario is '**Update Customers Table_5M Records**'.

Metadata

Scenario Name: Update Customers Table_5M Records

Scenario Author: Artemakis Artemiou

Scenario Description: This scenario updates two columns in a 'Customers' table with 5 million records as well as inserts into another table a subset of the first based on a range condition.

The scenario features all three execution modes. The memory-optimized versions of the 'Customers' table are durable, meaning that even in the case of server failover or crash, the data will still be available. The same stands for the 'Report' tables.

The same scenario exists for 100K, 300K, 500K, 1M and 3M records.

Enabled Modes:
Disk-Based, Memory-Optimized, Memory-Optimized with Natively-Compiled Stored Procedure

Listing 3.2: Scenario's Metadata.

Disk-Based Tables

```
--Disk-Based: Customers Table
CREATE TABLE [dbo].[d_customers]
(
  [Code] [CHAR](10) NOT NULL PRIMARY KEY,
  [FullName] [VARCHAR](250) NOT NULL,
  [Address] [VARCHAR](250) NOT NULL,
  [CountryCode] [CHAR](3) NOT NULL,
  [DiscountPerc] [FLOAT] NOT NULL,
  [RegistrationDate] [DATE] NOT NULL,
  [LastUpdated] [DATETIME] NULL
);

--Disk-Based: Report Table
CREATE TABLE [dbo].[d_report]
(
  [Code] [CHAR](10) NOT NULL PRIMARY KEY,
  [FullName] [VARCHAR](250) NOT NULL,
  [Address] [VARCHAR](250) NOT NULL,
  [CountryCode] [CHAR](3) NOT NULL,
  [DiscountPerc] [FLOAT] NOT NULL,
  [RegistrationDate] [DATE] NOT NULL,
  [LastUpdated] [DATETIME] NULL
);
```

Listing 3.3: Disk-Based Table Definitions.

Mem-Opt Tables

```
--Memory-Optimized: Customers Table
CREATE TABLE [dbo].[m_customers]
(
  [Code] [CHAR](10) NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 7000000),
  [FullName] [VARCHAR](250) NOT NULL,
  [Address] [VARCHAR](250) NOT NULL,
  [CountryCode] [CHAR](3) NOT NULL,
  [DiscountPerc] [FLOAT] NOT NULL,
  [RegistrationDate] [DATE] NOT NULL,
  [LastUpdated] [DATETIME] NULL
)WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA);

--Memory-Optimized: Report Table
CREATE TABLE [dbo].[m_report]
(
  [Code] [CHAR](10) NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 7000000),
  [FullName] [VARCHAR](250) NOT NULL,
  [Address] [VARCHAR](250) NOT NULL,
  [CountryCode] [CHAR](3) NOT NULL,
  [DiscountPerc] [FLOAT] NOT NULL,
  [RegistrationDate] [DATE] NOT NULL,
  [LastUpdated] [DATETIME] NULL
)WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA);
```

Listing 3.4: Memory-Optimized Table Definitions.

Mem-Opt Tables for Native SP

```
--Memory-Optimized for Natively-Compiled SP: Customers Table
CREATE TABLE [dbo].[n_customers]
(
  [Code] [CHAR](10) NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 7500000),
  [FullName] [VARCHAR](250) NOT NULL,
  [Address] [VARCHAR](250) NOT NULL,
  [CountryCode] [CHAR](3) NOT NULL,
  [DiscountPerc] [FLOAT] NOT NULL,
  [RegistrationDate] [DATE] NOT NULL,
  [LastUpdated] [DATETIME] NULL
)WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA);

--Memory-Optimized for Natively-Compiled SP: Report Table
CREATE TABLE [dbo].[n_report]
(
  [Code] [CHAR](10) NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 7500000),
  [FullName] [VARCHAR](250) NOT NULL,
  [Address] [VARCHAR](250) NOT NULL,
  [CountryCode] [CHAR](3) NOT NULL,
  [DiscountPerc] [FLOAT] NOT NULL,
  [RegistrationDate] [DATE] NOT NULL,
  [LastUpdated] [DATETIME] NULL
)WITH ( MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA);
```

Listing 3.5: Memory-Optimized Table Definitions for Natively-Compiled SP.

Data Population Script

```
--Data Population Script

--Clear all tables
TRUNCATE TABLE [dbo].[d_customers]
DELETE [dbo].[m_customers]
DELETE [dbo].[n_customers]
TRUNCATE TABLE [dbo].[d_report]
DELETE [dbo].[m_report]
DELETE [dbo].[n_report]

--Populate Memory-Optimized Tables
INSERT [dbo].[m_customers] ([Code], [FullName], [Address], [CountryCode], [DiscountPerc],
[RegistrationDate])
SELECT [Code], [FullName], [Address], [CountryCode], [DiscountPerc], [RegistrationDate]
FROM [.\SQL2K16CTP].[ERPSampleDB].[dbo].[Customers]

--Populate Memory-Optimized Tables for Native SP
INSERT [dbo].[n_customers] ([Code], [FullName], [Address], [CountryCode], [DiscountPerc],
[RegistrationDate])
SELECT [Code], [FullName], [Address], [CountryCode], [DiscountPerc], [RegistrationDate]
FROM [dbo].[m_Customers]

--Populate Disk-Based Tables
INSERT [dbo].[d_customers] ([Code], [FullName], [Address], [CountryCode], [DiscountPerc],
[RegistrationDate])
SELECT [Code], [FullName], [Address], [CountryCode], [DiscountPerc], [RegistrationDate]
FROM [dbo].[m_customers]
```

Listing 3.6: Data Population Script.

Prep Script

```
--Clear the Report tables
TRUNCATE TABLE [dbo].[d_report]
DELETE [dbo].[m_report]
DELETE [dbo].[n_report]
```

Listing 3.7: Preparation Script.

Disk-Based SP

```
--Disk-Based Stored Procedure for Scenario Execution - Definition
CREATE PROCEDURE [dbo].[d_Process]
AS
BEGIN

   UPDATE dbo.d_customers
   SET [DiscountPerc]=[DiscountPerc]+10.0
       ,[LastUpdated]=getdate();

   INSERT INTO dbo.d_report ([Code], [FullName], [Address], [CountryCode],
                             [DiscountPerc], [RegistrationDate], [LastUpdated])
   SELECT [Code], [FullName], [Address], [CountryCode],
          [DiscountPerc], [RegistrationDate], [LastUpdated]
   FROM dbo.d_Customers
   WHERE discountPerc>=20;

END;
```

Listing 3.8: Disk-Based Stored Procedure Definition.

Mem-Opt SP

```
--Memory-Optimized Stored Procedure for Scenario Execution - Definition
CREATE PROCEDURE [dbo].[m_Process]
AS
BEGIN

   UPDATE dbo.m_customers
   SET [DiscountPerc]=[DiscountPerc]+10.0
       ,[LastUpdated]=getdate();

   INSERT INTO dbo.m_report ([Code], [FullName], [Address], [CountryCode],
                             [DiscountPerc], [RegistrationDate], [LastUpdated])
   SELECT [Code], [FullName], [Address], [CountryCode],
          [DiscountPerc], [RegistrationDate], [LastUpdated]
   FROM dbo.m_Customers
   WHERE discountPerc>=20;

END;
```

Listing 3.9: Memory-Optimized Stored Procedure Definition.

Natively-Compiled SP

```
--Natively Compiled Stored Procedure for Scenario Execution - Definition
CREATE PROCEDURE [dbo].[n_Process]
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS
BEGIN ATOMIC WITH
  (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
   LANGUAGE = N'us_english')

   UPDATE dbo.n_customers
   SET [DiscountPerc]=[DiscountPerc]+10.0
       ,[LastUpdated]=getdate();

      INSERT INTO dbo.n_report ([Code], [FullName], [Address], [CountryCode],
                                [DiscountPerc], [RegistrationDate], [LastUpdated])
   SELECT [Code], [FullName], [Address], [CountryCode],
          [DiscountPerc], [RegistrationDate], [LastUpdated]
   FROM dbo.n_Customers
   WHERE discountPerc>=20;

END;
```

Listing 3.10: Natively-Compiled Stored Procedure Definition.

Execution Script

```
--Disk-Based SP
EXEC d_process;

--Memory-Optimized SP
EXEC m_process;

--Natively-Compiled SP
EXEC n_process;
```

Listing 3.11: Execution Script.

Housekeeping Script

Not used for the scenarios in this paper.

4. Simulation Statistics

The simulation statistics for all workload volumes are presented below. It is recommended to observe the trends on performance as the workload size increases.

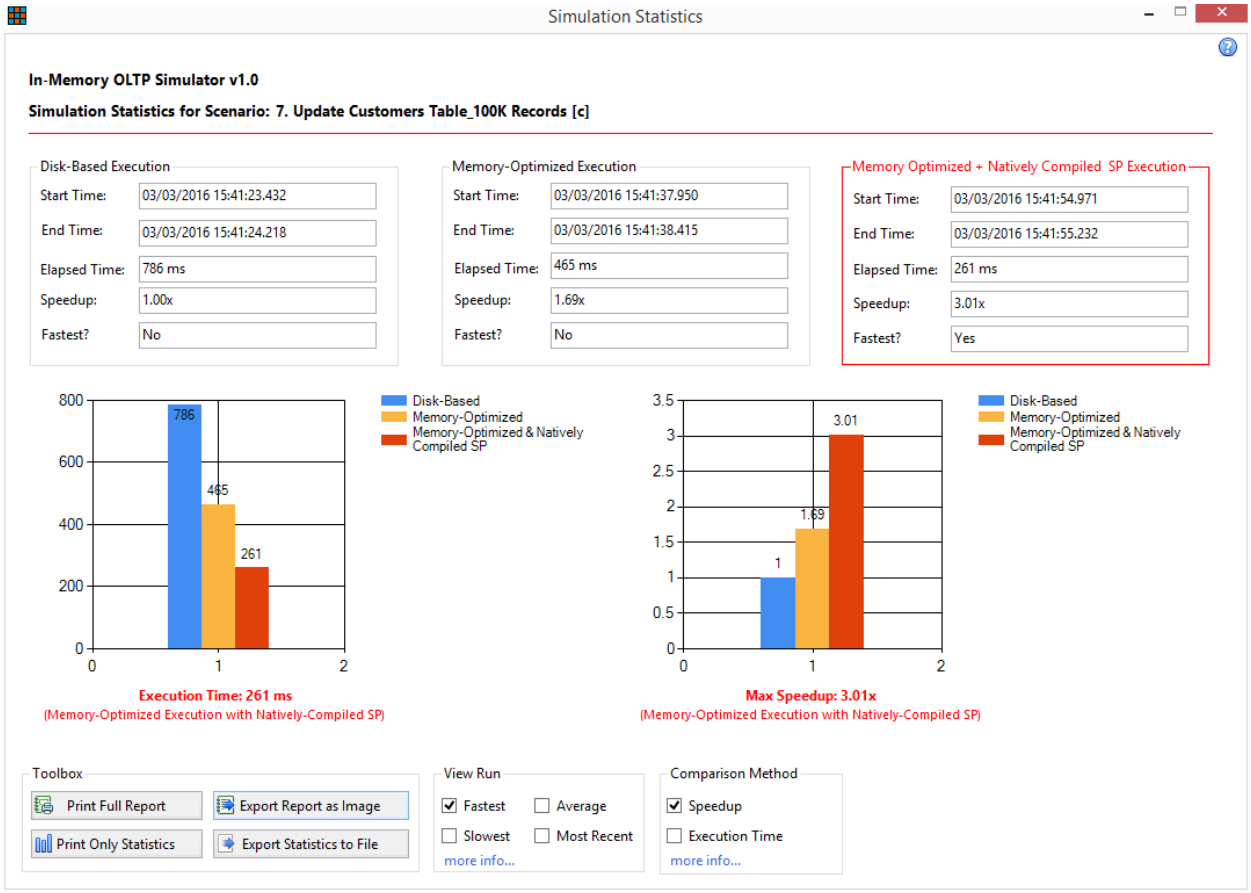


Figure 4.1: Execution Statistics for Customers Table with 100K Records.

The above figure shows the simulation statistics for the **100K** customer records workload. As you can see, the Memory-Optimized mode ran **1.69** times faster than the baseline that is the Disk-Based execution mode. The Memory-Optimized with Natively-Compiled Stored Procedure mode ran **3.01** times faster.

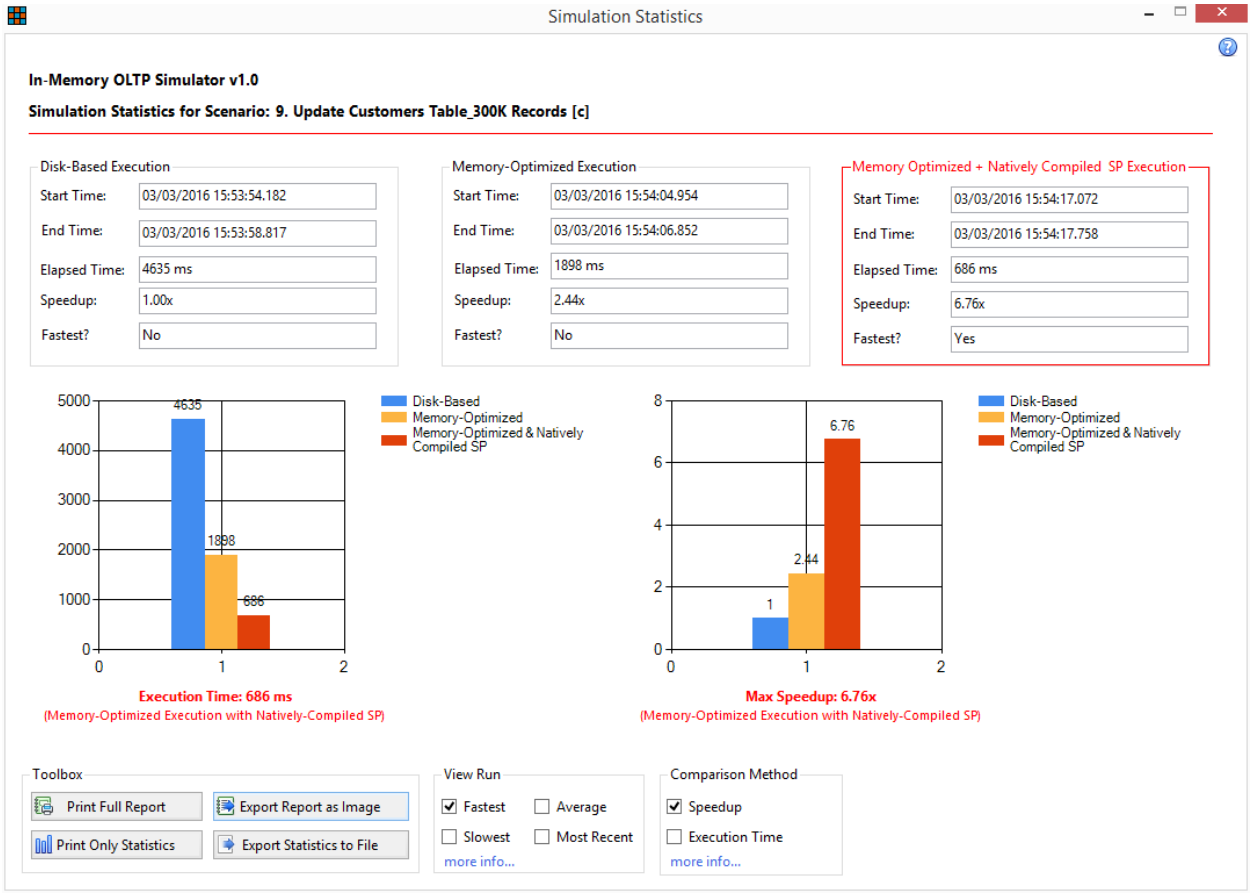


Figure 4.2: Execution Statistics for Customers Table with 300K Records.

The above figure shows the simulation statistics for the **300K** customer records workload. As you can see, the Memory-Optimized mode ran **2.44** times faster than the Disk-Based execution mode. The Memory-Optimized with Natively-Compiled Stored Procedure mode ran **6.76** times faster.

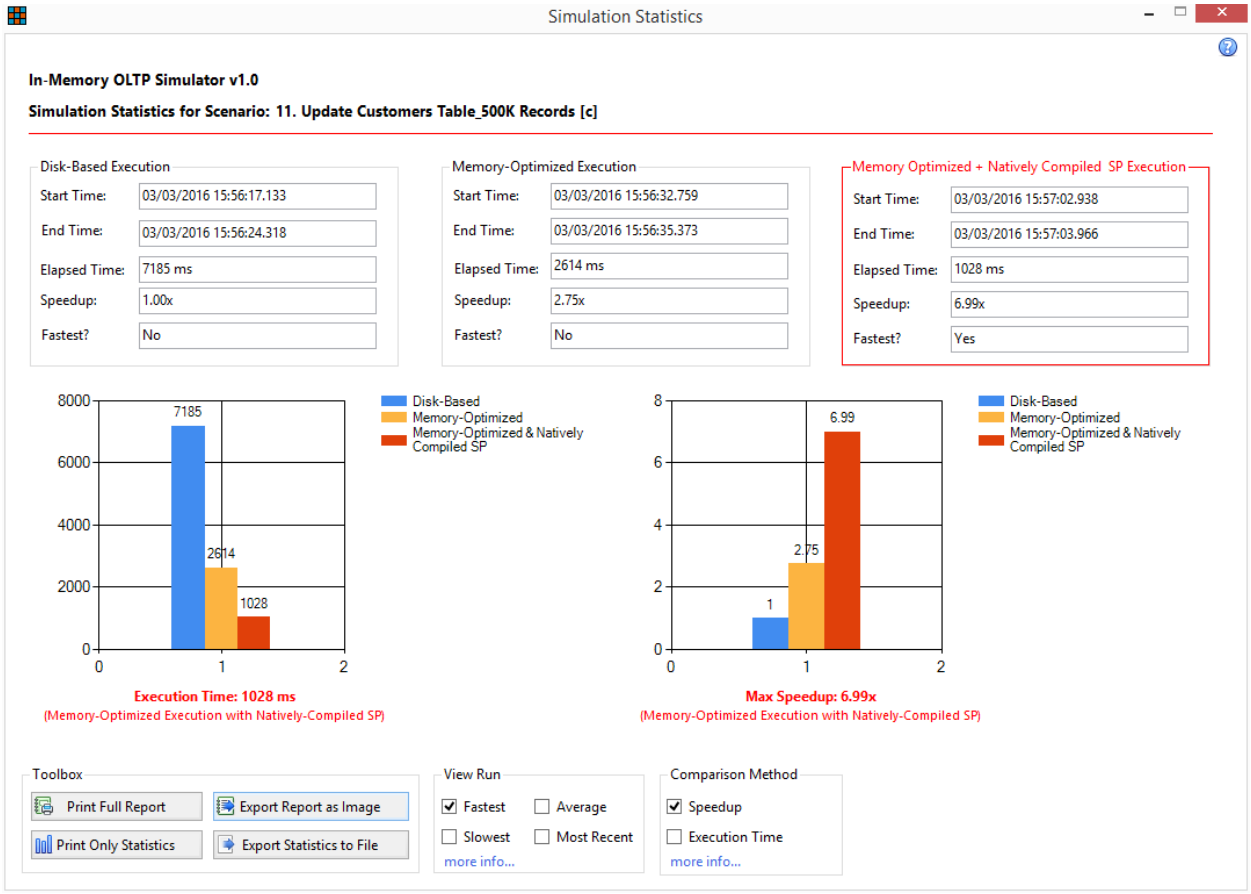


Figure 4.3: Execution Statistics for Customers Table with 500K Records.

The above figure shows the simulation statistics for the **500K** customer records workload. The Memory-Optimized mode ran **2.75** times faster than the Disk-Based execution mode. The Memory-Optimized with Natively-Compiled Stored Procedure mode ran **6.99** times faster.

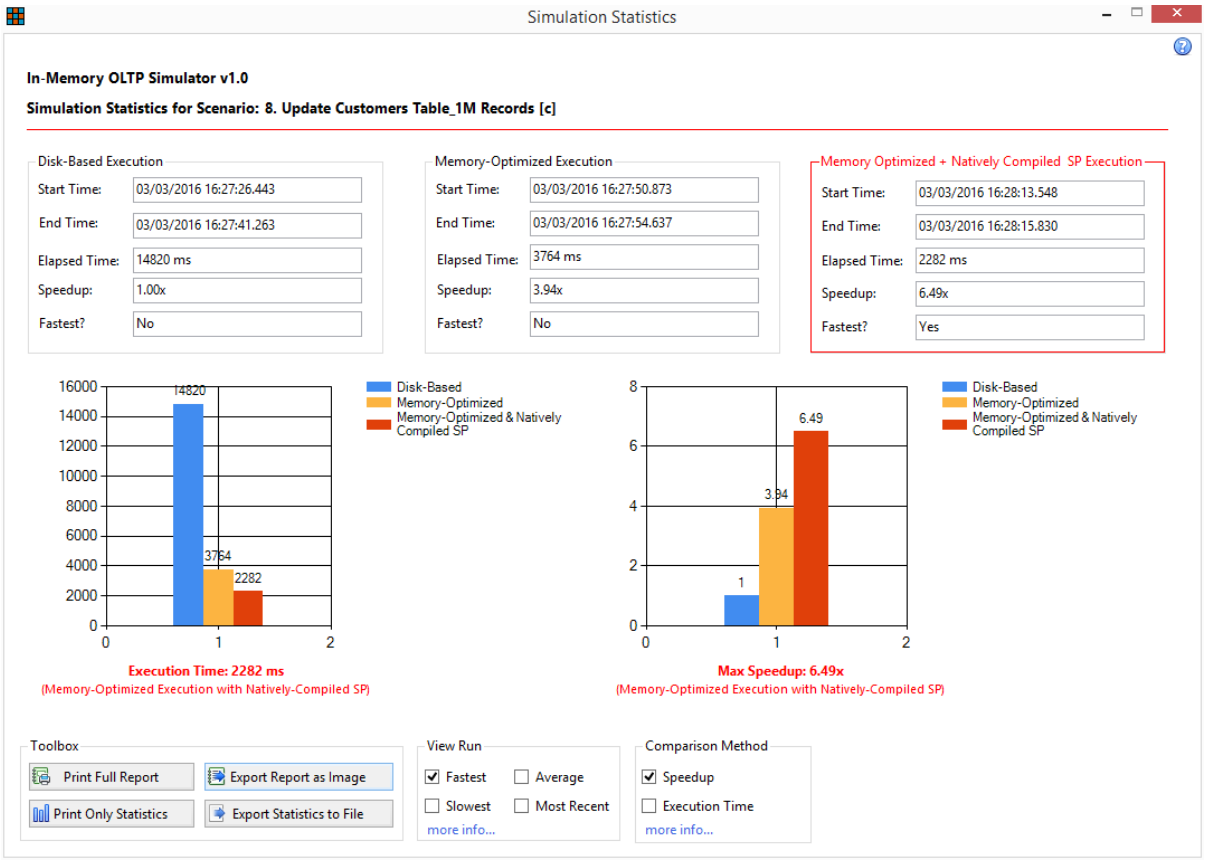


Figure 4.4: Execution Statistics for Customers Table with 1M Records.

The above figure shows the simulation statistics for the **1M** customer records workload. The Memory-Optimized mode ran **3.94** times faster than the Disk-Based execution mode. The Memory-Optimized with Natively-Compiled Stored Procedure mode ran **6.49** times faster.

Another observation that could be done for this run it is that the speedup for the Memory-Optimized Execution with Natively-Compiled SP is a little lower than in the previous run (500K records). If we observe however all runs we can see that this is not the trend. In all the other runs the speedup for Memory-Optimized Execution with Natively-Compiled SP gets higher all the time (in comparison to each previous run) so this might be an isolated incident (i.e. maybe at that time another I/O intensive process ran and affected the simulation's execution in terms of disk latency). You must note that in SQL Server In-Memory Optimization, whenever **durable** tables are used, besides RAM, the data also resides on disk in order to be recovered in the case of a server failover or crash.

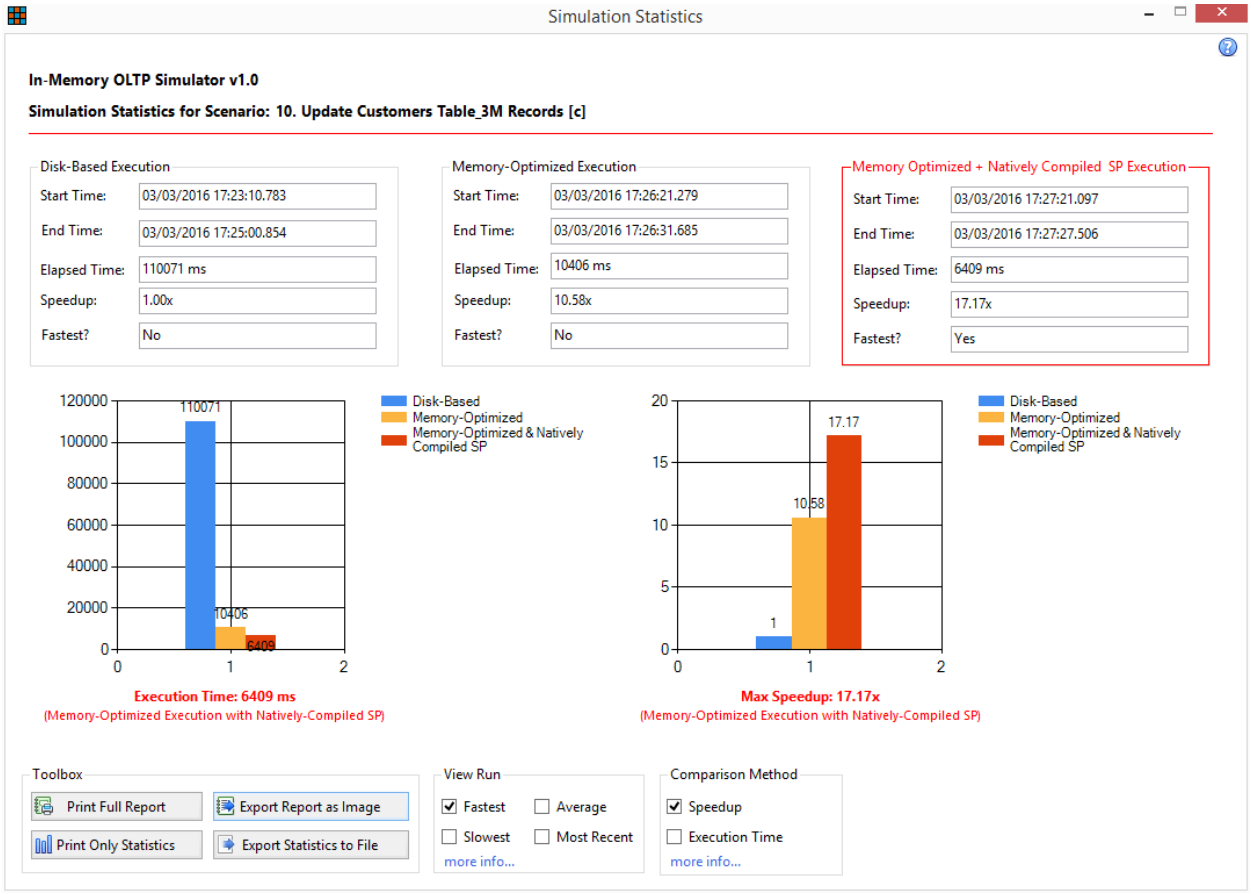


Figure 4.5: Execution Statistics for Customers Table with 3M Records.

The above figure shows the simulation statistics for the **3M** customer records workload. The Memory-Optimized mode ran **10.58** times faster than the Disk-Based execution mode. The Memory-Optimized with Natively-Compiled Stored Procedure mode ran **17.17** times faster.

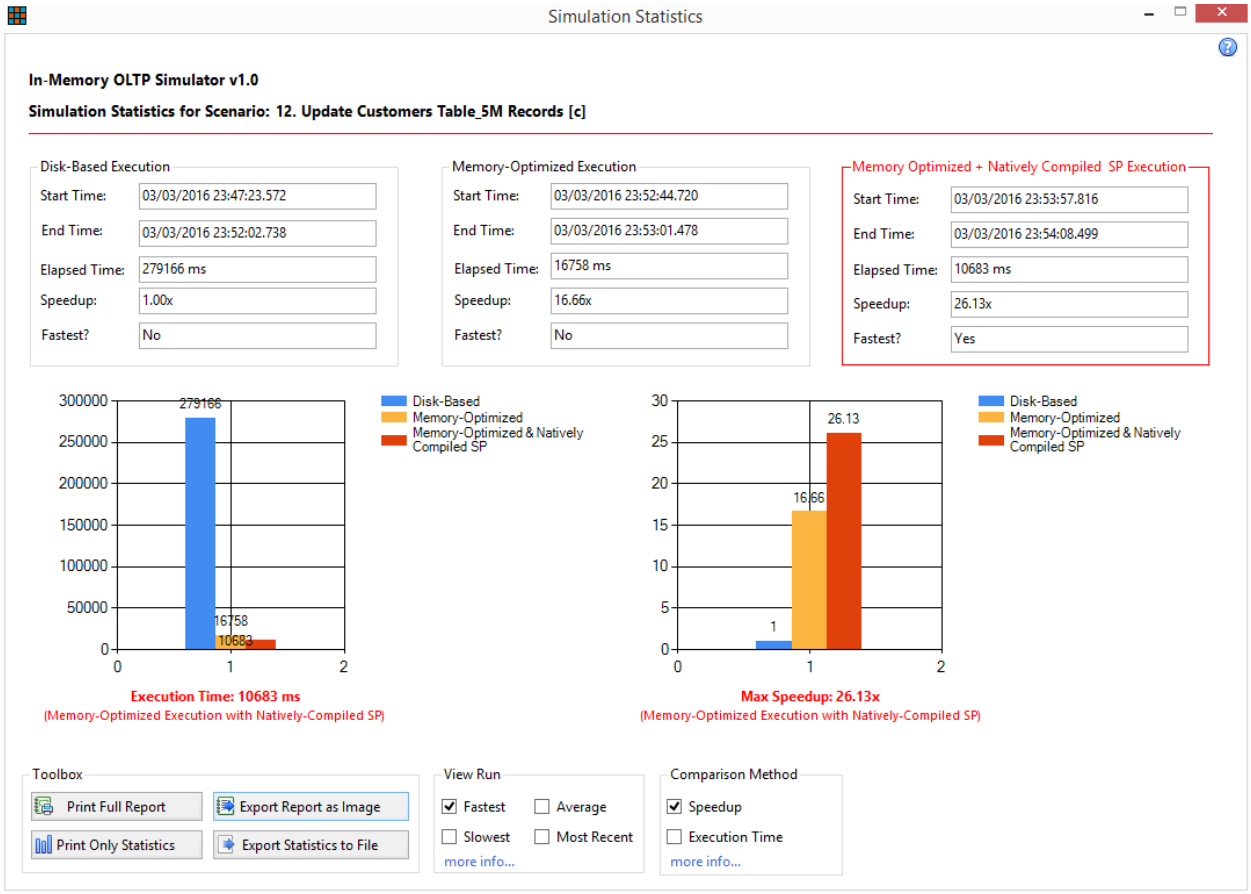


Figure 4.6: Execution Statistics for Customers Table with 5M Records.

The above figure shows the simulation statistics for the **5M** customer records workload. The Memory-Optimized mode ran **16.66** times faster than the Disk-Based execution mode. The Memory-Optimized with Natively-Compiled Stored Procedure mode ran **26.13 times faster**.

In addition to the simulation statistics presented above for all the scenarios, below you can see the processing output for the last scenario, that is the scenario with the Customers table of 5M records. This is provided as a proof that all three modes process the same logic and produce the same output.

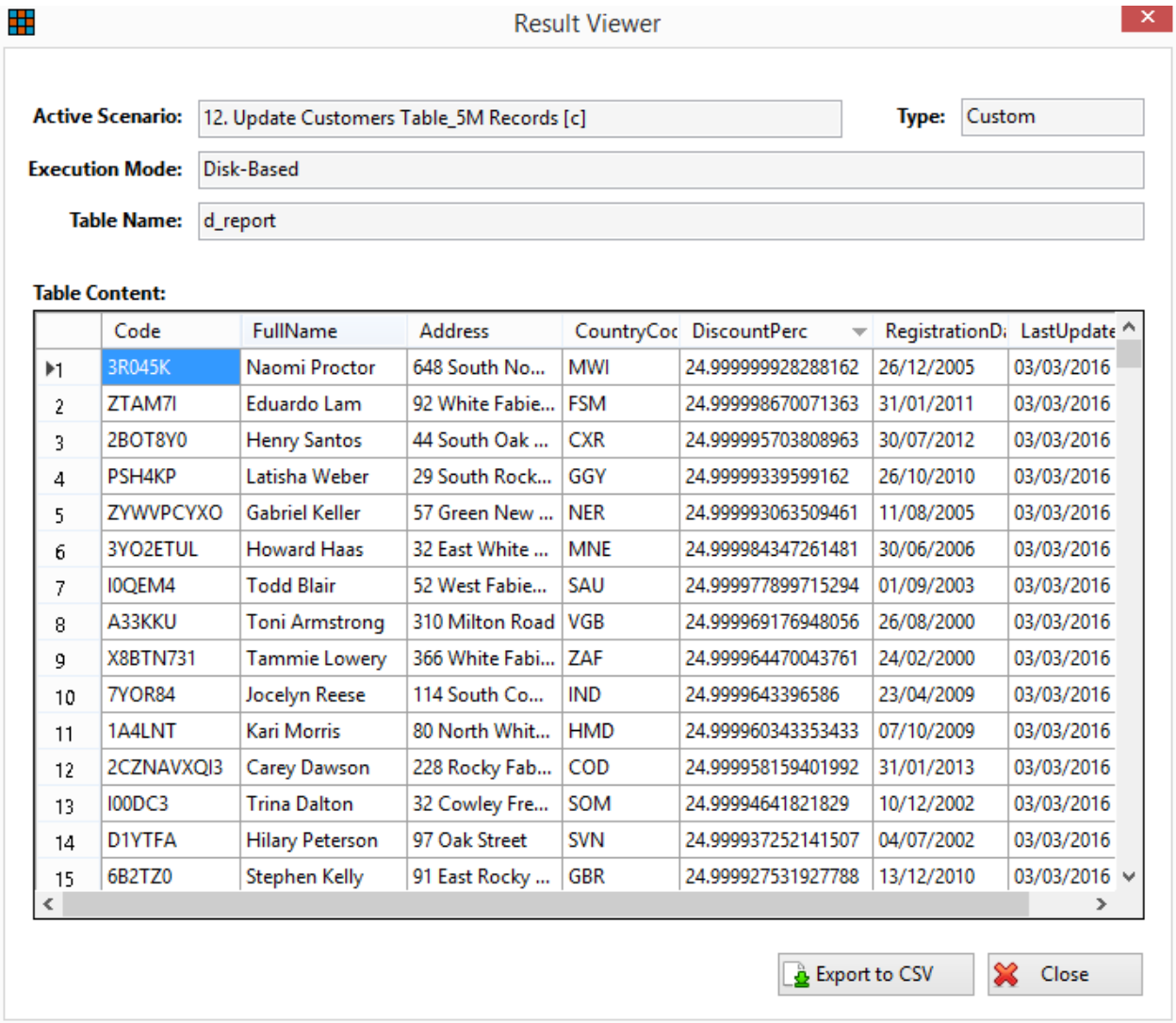


Figure 4.7: Generated Report by Disk-Based Execution Mode (scenario with 5M customer records).

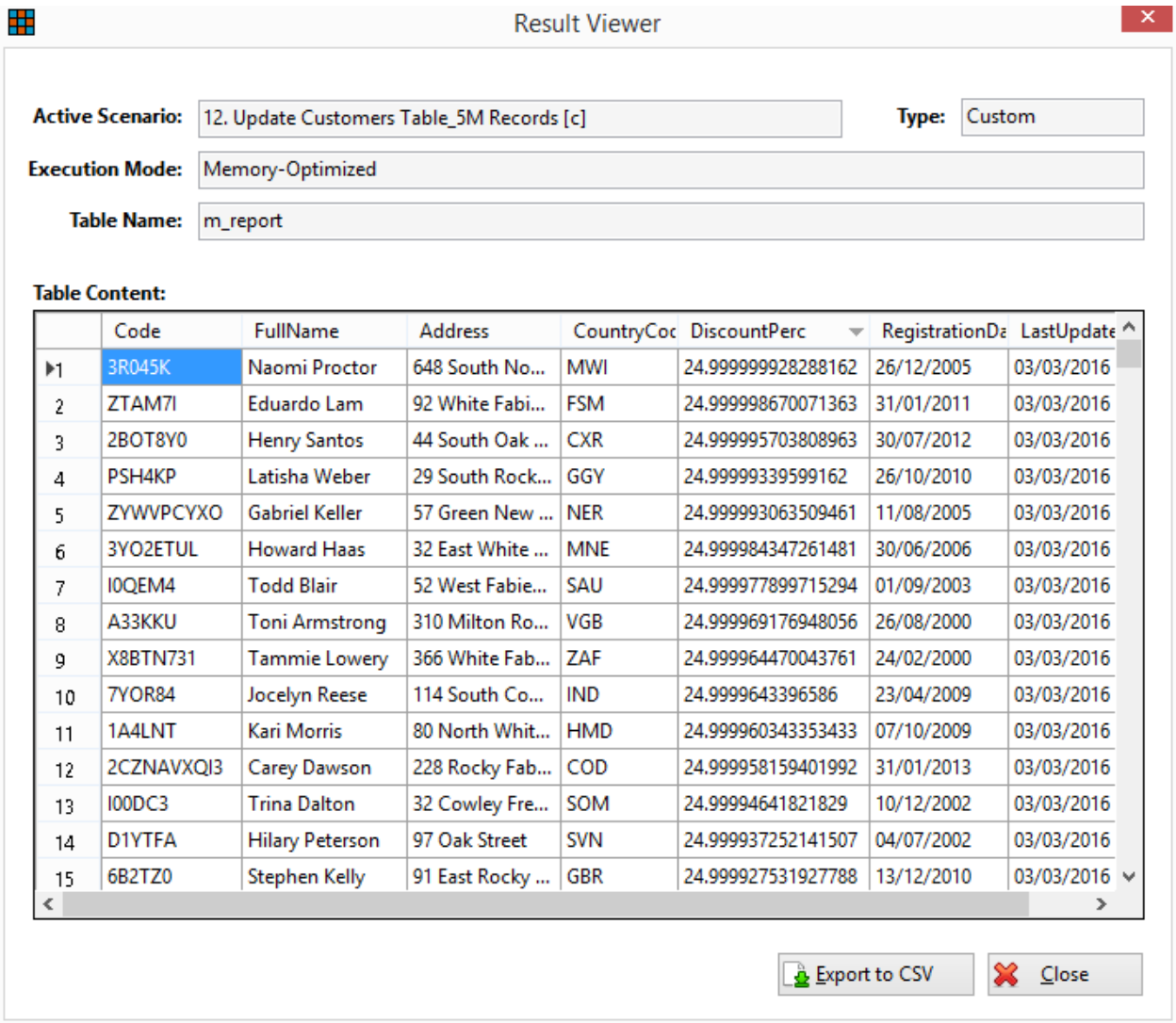


Figure 4.8: Generated Report by Memory-Optimized Execution Mode (scenario with 5M customer records).

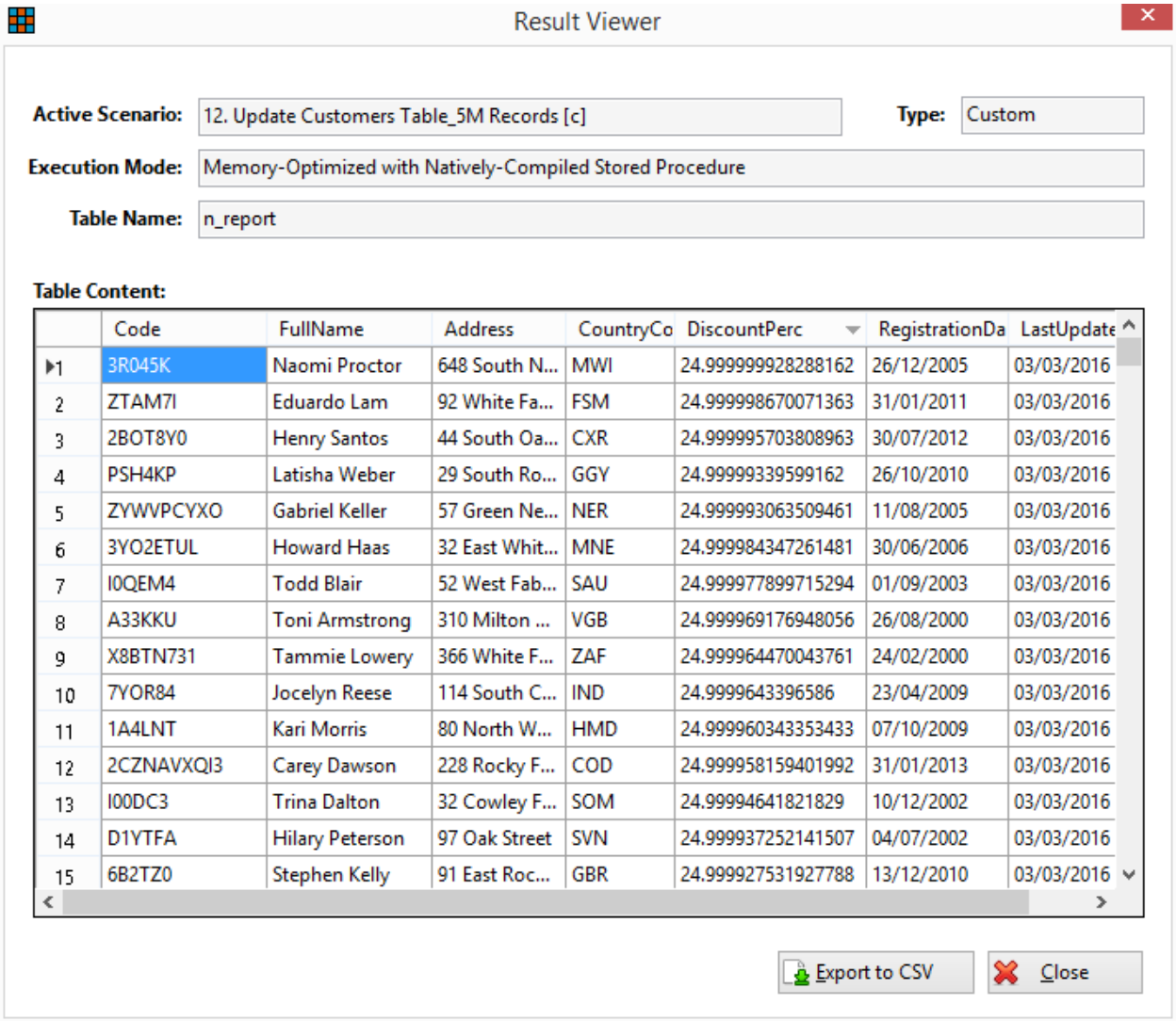


Figure 4.9: Generated Report by Memory-Optimized with Natively-Compiled Stored Procedure Execution Mode (scenario with 5M customer records).

The figure below shows the number of records in the three customer tables used by the corresponding execution modes. As you can see, each mode processed the same number of records, in this case 5M records. The same stands for the rest of the workloads with the corresponding number of records (100K, 300K, 500K, 1M and 3M).

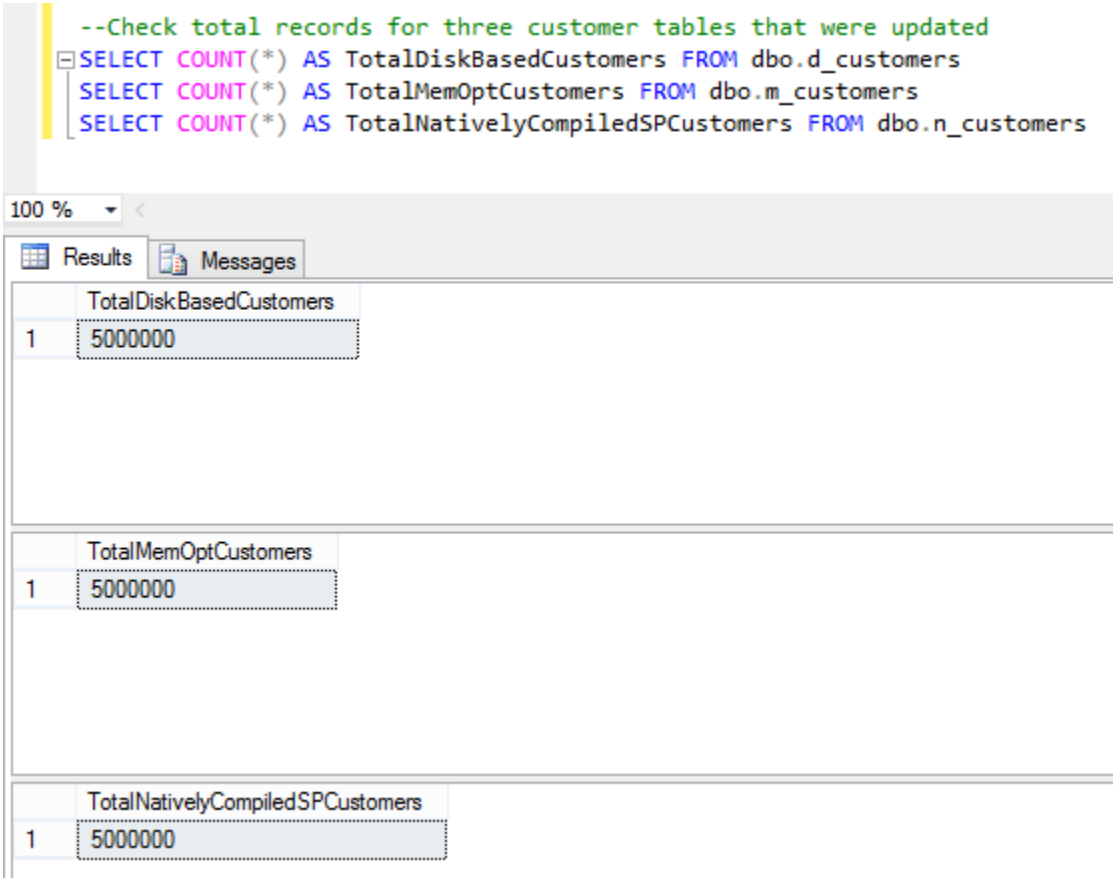


Figure 4.10: Total Number of Records in the 3 Customer Tables for the 5M Records Scenario.

The figure below shows the number of records produced for the requested report. As you can see, each mode produced the same number of records in the report, in this case for the 5M records workload. The same stands for the rest of the workloads with the corresponding number of records (100K, 300K, 500K, 1M and 3M).

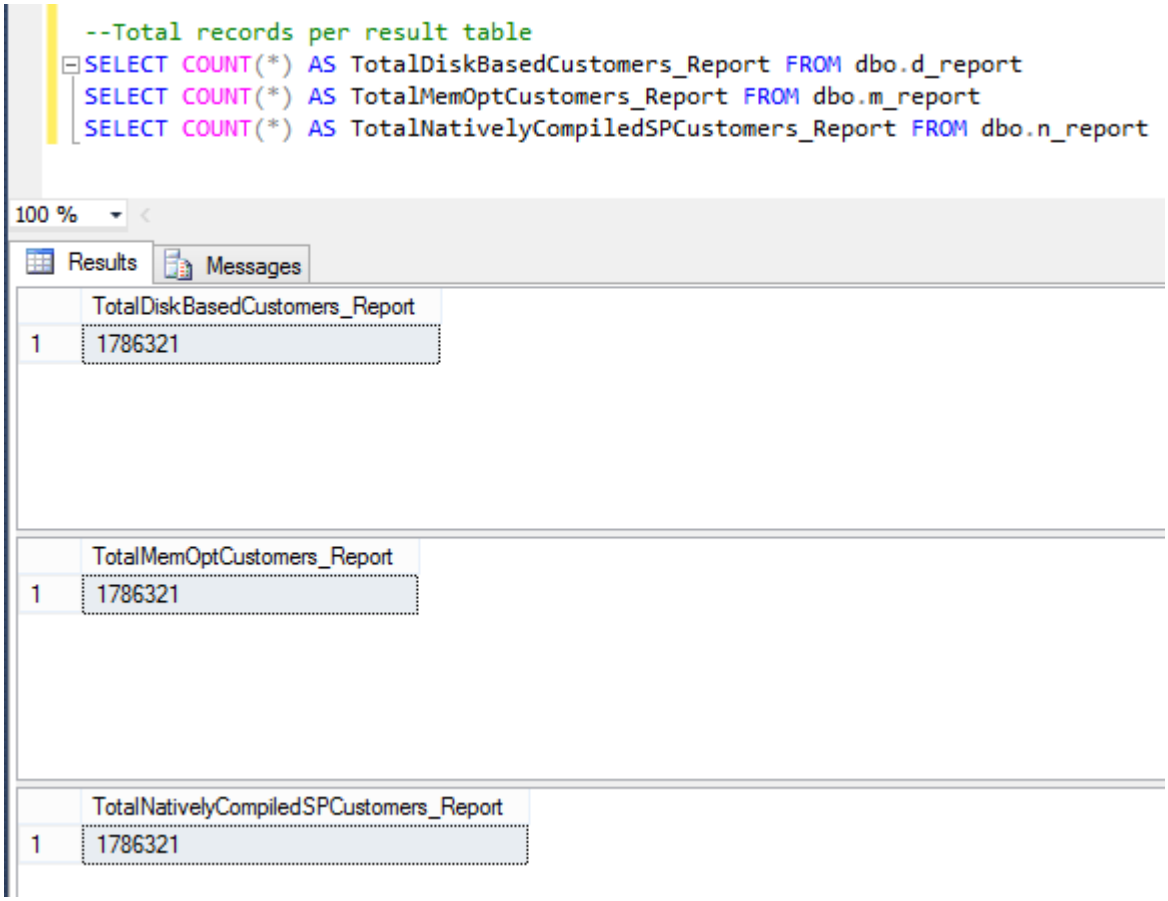


Figure 4.11: Total Number of Records in the 3 Report Tables for the 5M Records Workload.

5. Analysis of Simulation Results

Below you can see the comparative simulation statistics for all three modes per workload size/scenario, in terms of execution times and speedup.

Execution Times (in milliseconds)						
	Workload Size (Number of Records to be Processed)					
Execution Mode	**100K**	**300K**	**500K**	**1M**	**3M**	**5M**
Disk-Based	786	4635	7185	14820	110071	279166
Memory-Optimized	465	1898	2614	3764	10406	16758
Memory-Optimized with Natively-Compiled Stored Procedure	261	686	1028	2282	6409	10683

Table 5.1: Comparative Execution Times per Mode for all Workloads.

From the comparative results in Table 5.1, we can see that as the workload size increases, the performance gap between the baseline (Disk-Based mode) and the two Memory-Optimized/Natively-Compiled Stored Procedure modes increases. The gap becomes much larger in the cases where the workload sizes are 3M and 5M. This is clearly illustrated in the graph presented below (Figure 5.1).

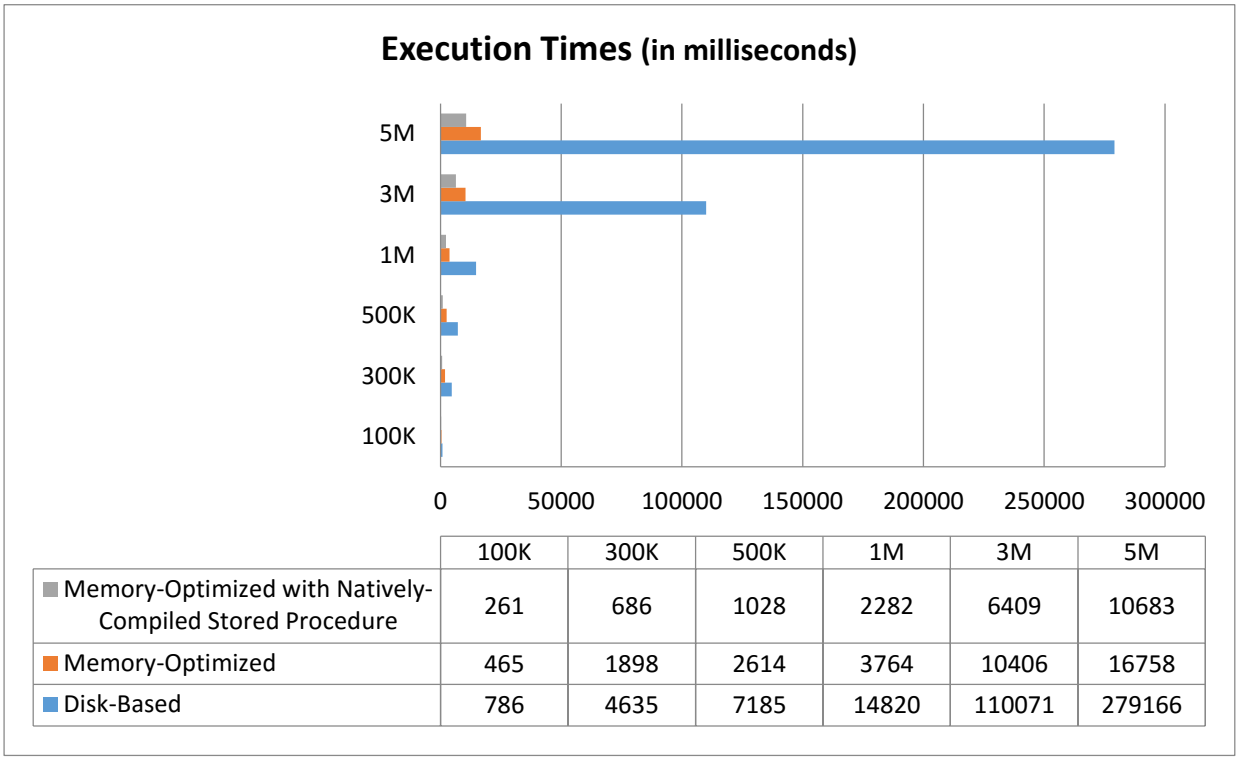


Figure 5.1: Graph Comparing the Execution Times per Mode for all Workloads.

As speedups are calculated based on the execution times, similarly, in Table 5.2 we can see that as the workload size increases, the speedup of the two Memory-Optimized/Natively-Compiled Stored Procedure modes increases. Again, in the case of the two largest workloads (3M and 5M records) the speedups are remarkable.

Speedups (x times faster than baseline)						
	Workload Size (Number of Records to be Processed)					
Execution Mode	**100K**	**300K**	**500K**	**1M**	**3M**	**5M**
Disk-Based	1	1	1	1	1	1
Memory-Optimized	1.69	2.44	2.75	3.94	10.58	16.66
Memory-Optimized with Natively-Compiled Stored Procedure	3.01	6.76	6.99	6.49	17.17	26.13

Table 5.2: Comparative Speedups per Mode for all Workloads.

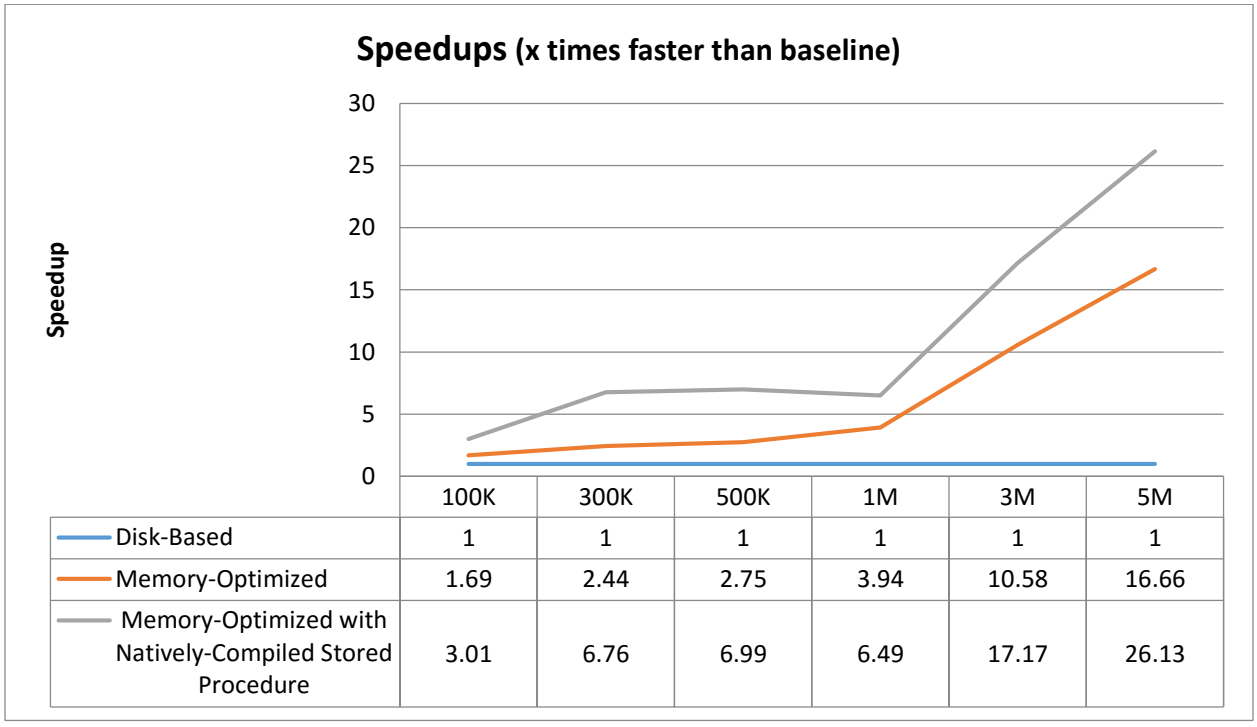


Table 5.2: Graph Comparing the Speedups per Mode for all Workloads.

6. Conclusions

The source of inspiration for the design and development of In-Memory OLTP Simulator, is the powerful SQL Server's In-Memory OLTP Engine and the fact that it can boost the performance of a plethora of database operations for different workload types.

The motivation behind the development of In-Memory OLTP Simulator, was to build a software tool that would make it easy for any SQL Server user to realistically check for possible performance gains for his or her specific workload type and volume when using SQL Server's In-Memory OLTP Engine.

This paper aimed at two things: (i) To show how easy is to simulate your workload using In-Memory OLTP Simulator and process it using SQL Server's In-Memory OLTP Engine, and (ii) To let you get a glimpse of what SQL Server's In-Memory OLTP Engine can offer when it comes to performance boost for real-life database processing scenarios.

In-Memory OLTP Simulator is constantly evolving, more functionality is being added, and it is being transformed into a powerful simulation platform that can help the user set different parameters for his or her workload, run it against the powerful SQL Server In-Memory OLTP Engine, and get meaningful statistics that can help him or her decide which is the optimal way of processing specific workload types and volumes.

The simulations presented in this paper showed significant performance gain when SQL Server's In-Memory OLTP Engine was used for processing the target workloads. For all workload sizes, there was a performance boost when the In-Memory technology was used, in comparison to the baseline that is the Disk-Based mode. Also, as the workload size got increased, the performance gain was remarkably larger as the Disk-Based mode became much slower in contrast to the Memory-Optimized and Memory-Optimized with Natively-Compiled Stored Procedure execution modes.

SQL Server In-Memory Optimization is a powerful new technology which will radically change what we knew about performance in data processing operations. It offers significant performance boost for a large set of workload types which is something that can definitely help towards conducting much more efficient data processing.

The existence of tools like In-Memory OLTP Simulator can help the user investigate and realize the potential performance benefits when using SQL Server In-Memory Optimization for his or her workload.

7. Future Work

With In-Memory OLTP Simulator, the possibilities are endless. You can easily simulate just about any workload against SQL Server's In-Memory OLTP Engine and study the performance trends.

Future work includes adding to In-Memory OLTP Simulator support for Azure SQL Database, table designer functionality, additional standard scenarios, and syntax highlighting as well as other enhancements.

Furthermore, more technical articles like this will be published in the future which will be describing how more complex data processes can be simulated using In-Memory OLTP Simulator and executed against SQL Server's In-Memory OLTP Engine.

Resources

To learn more about In-Memory OLTP Simulator, visit:

- In-Memory OLTP Simulator Official Website
- In-Memory OLTP Simulator Blog
- In-Memory OLTP Simulator Frequently Asked Questions
- In-Memory OLTP Simulator Datasheet
- In-Memory OLTP Simulator Facebook Page
- Artemiou Data Tools Website
- Artemakis Artemiou Official Website
- Artemakis Artemiou Blog
- Artemakis Artemiou on Twitter

To learn more about SQL Server In-Memory OLTP, visit:

- MSDN Library: In-Memory OLTP (In-Memory Optimization)
- Microsoft Whitepaper: In-Memory OLTP – Common Workload Patterns and Migration Considerations
- Microsoft Whitepaper: SQL Server 2016 In-memory OLTP
- Microsoft Whitepaper: SQL Server 2016 In-memory OLTP and Columnstore Feature Comparison
- Simple-Talk Article: Introducing SQL Server In-Memory OLTP
- The SQL Server and .NET Blog Articles:
    - In-Memory OLTP: Comparison of Features/Limitations between SQL Server 2014 and SQL Server 2016
    - In-Memory Optimization in SQL Server: Will my Workload Execute Faster?
- Download and Evaluate SQL Server 2016 (CTP 3.3)

Feedback

Did you find this paper useful? Please submit your feedback on this link with any comments and suggestions.