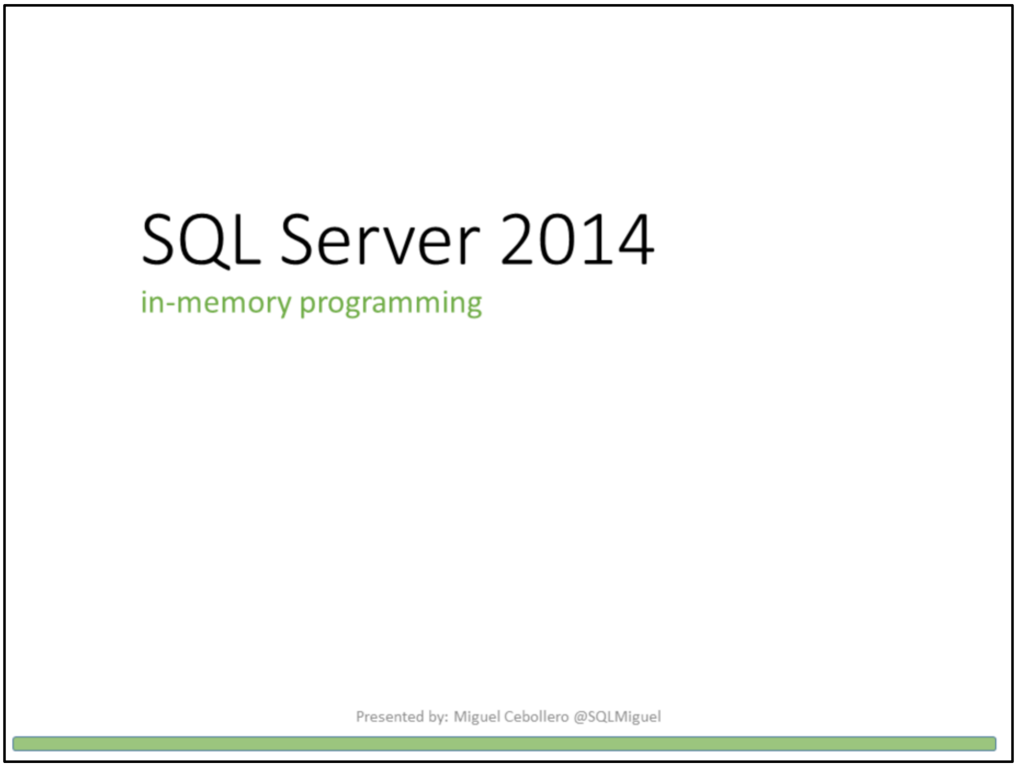
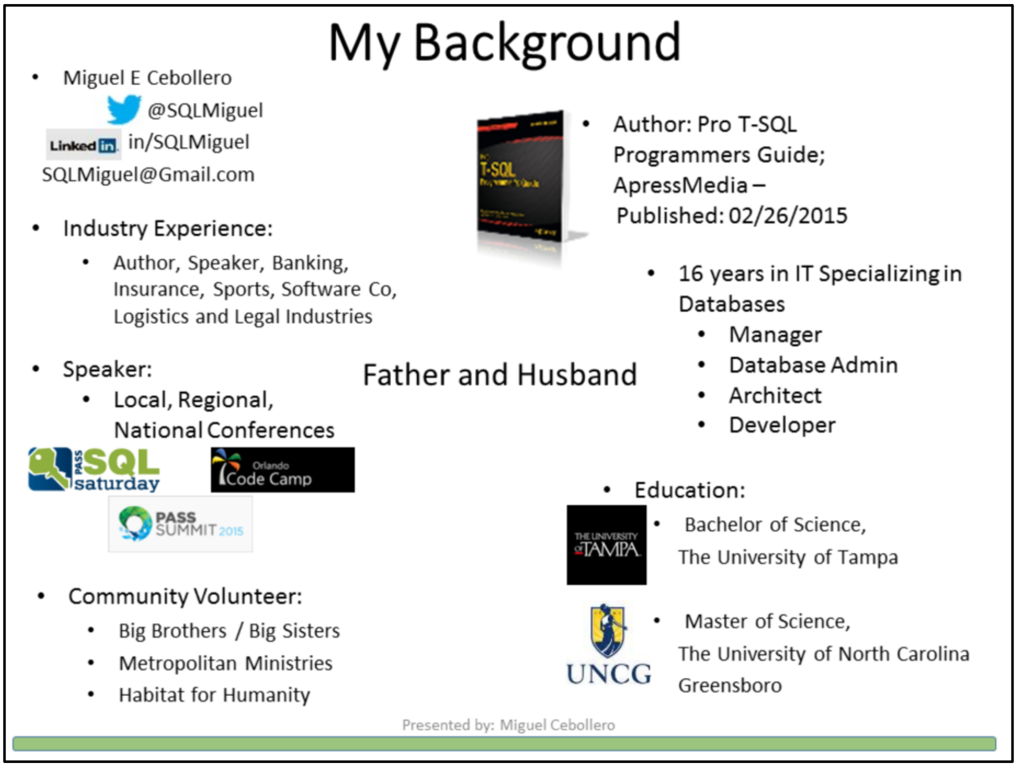


SQL Server 2014

in-memory programming

Presented by: Miguel Cebollero @SQLMiguel



My Background

- Miguel E Cebollero
  - @SQLMiguel
  - in/SQLMiguel
  - SQLMiguel@Gmail.com
- Industry Experience:
  - Author, Speaker, Banking, Insurance, Sports, Software Co, Logistics and Legal Industries
- Speaker:
  - Local, Regional, National Conferences

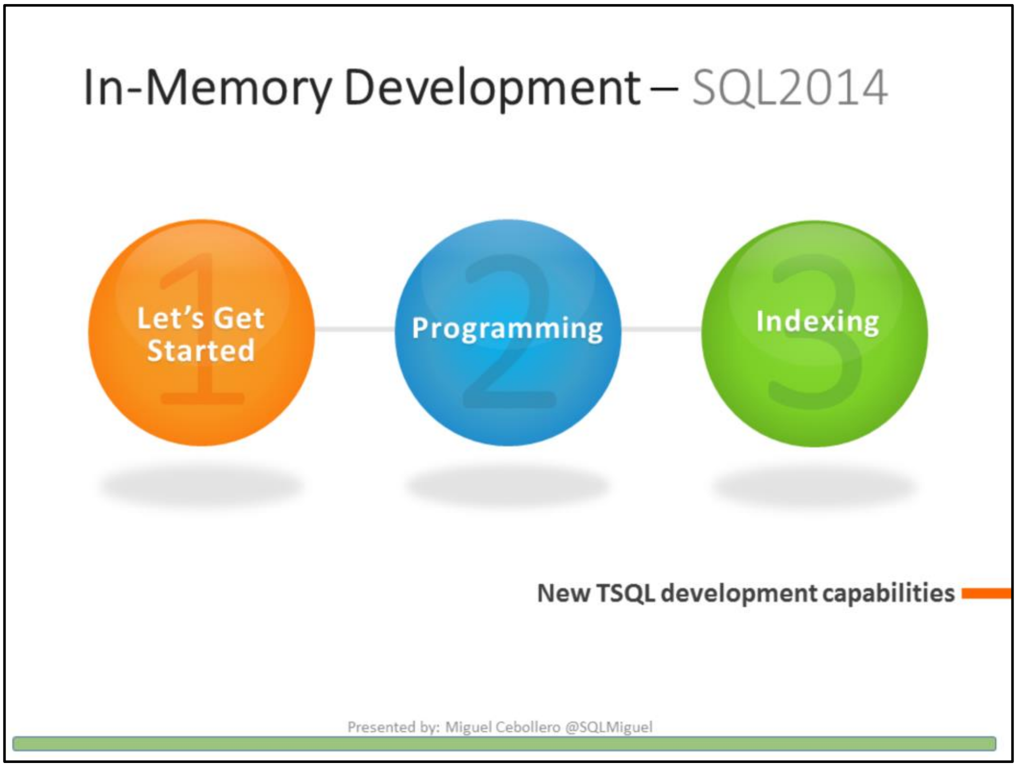
PASS SQL saturday
Orlando Code Camp
PASS SUMMIT 2015

- Community Volunteer:
  - Big Brothers / Big Sisters
  - Metropolitan Ministries
  - Habitat for Humanity

Father and Husband

- Author: Pro T-SQL Programmers Guide; ApressMedia – Published: 02/26/2015
- 16 years in IT Specializing in Databases
  - Manager
  - Database Admin
  - Architect
  - Developer
- Education:
  - Bachelor of Science, The University of Tampa
  - Master of Science, The University of North Carolina Greensboro

Presented by: Miguel Cebollero



In-Memory Development – SQL2014

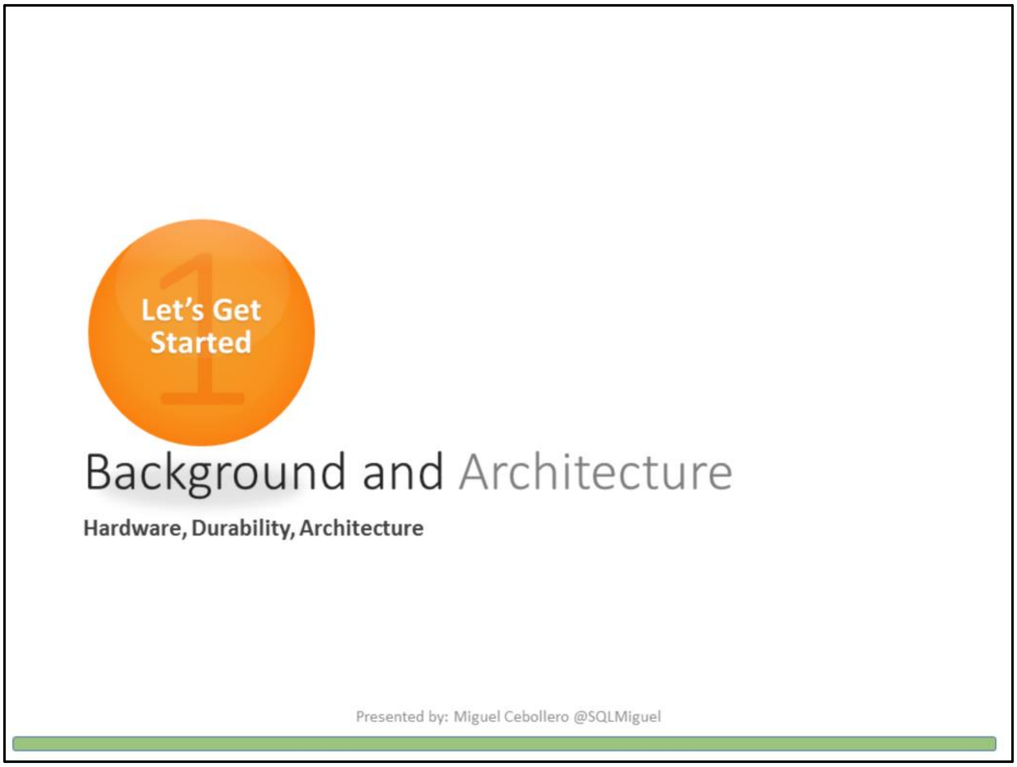
1 Let's Get Started

2 Programming

3 Indexing

New TSQL development capabilities

Presented by: Miguel Cebollero @SQLMiguel



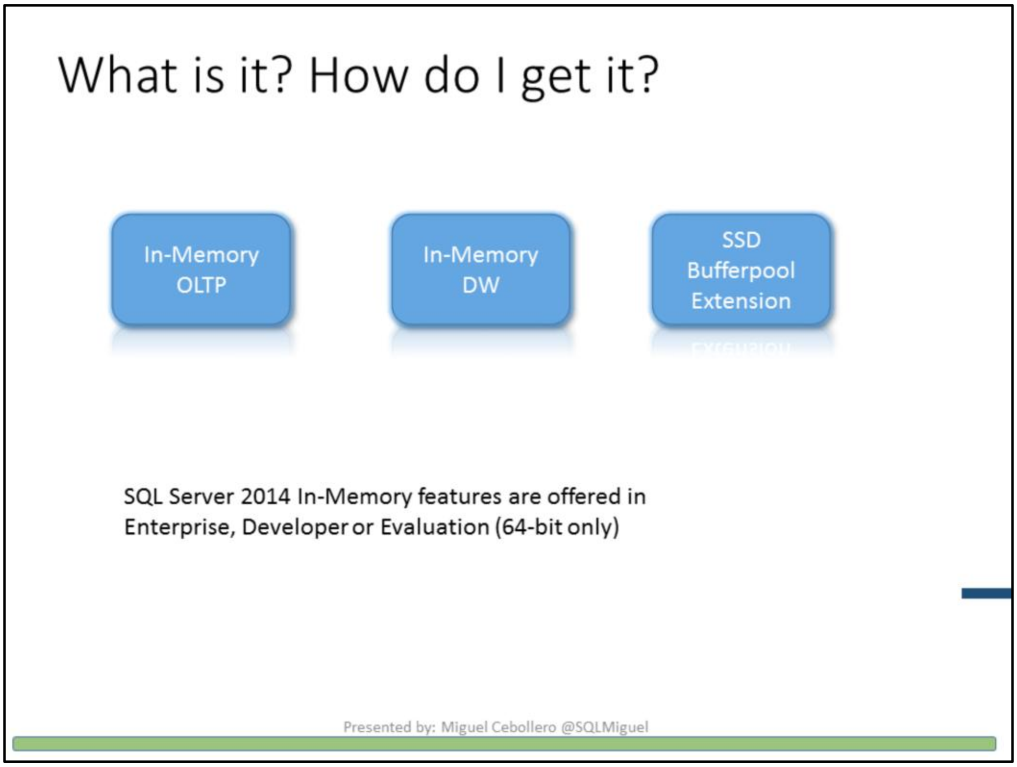
1

Let's Get Started

Background and Architecture

Hardware, Durability, Architecture

Presented by: Miguel Cebollero @SQLMiguel



What is it? How do I get it?

In-Memory OLTP

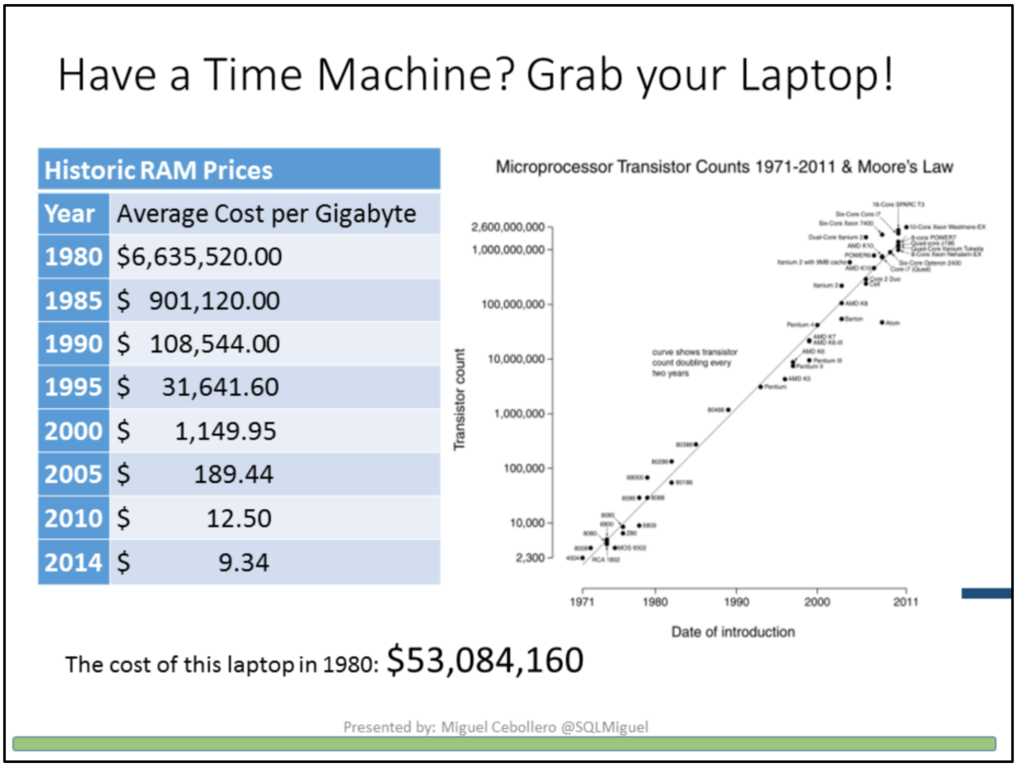
In-Memory DW

SSD Bufferpool Extension

SQL Server 2014 In-Memory features are offered in Enterprise, Developer or Evaluation (64-bit only)

Presented by: Miguel Cebollero @SQLMiguel

- In-Memory OLTP: The transactional lock-free, latch-free processing of transactional data on the database within the memory (RAM) of the server.
- In-Memory DW: Columnstore indexes that are contained within the memory (RAM) of the server.
- SSD Bufferpool Extension: provides seamless integration of the database engine buffer pool to improve I/O throughput.



Have a Time Machine? Grab your Laptop!

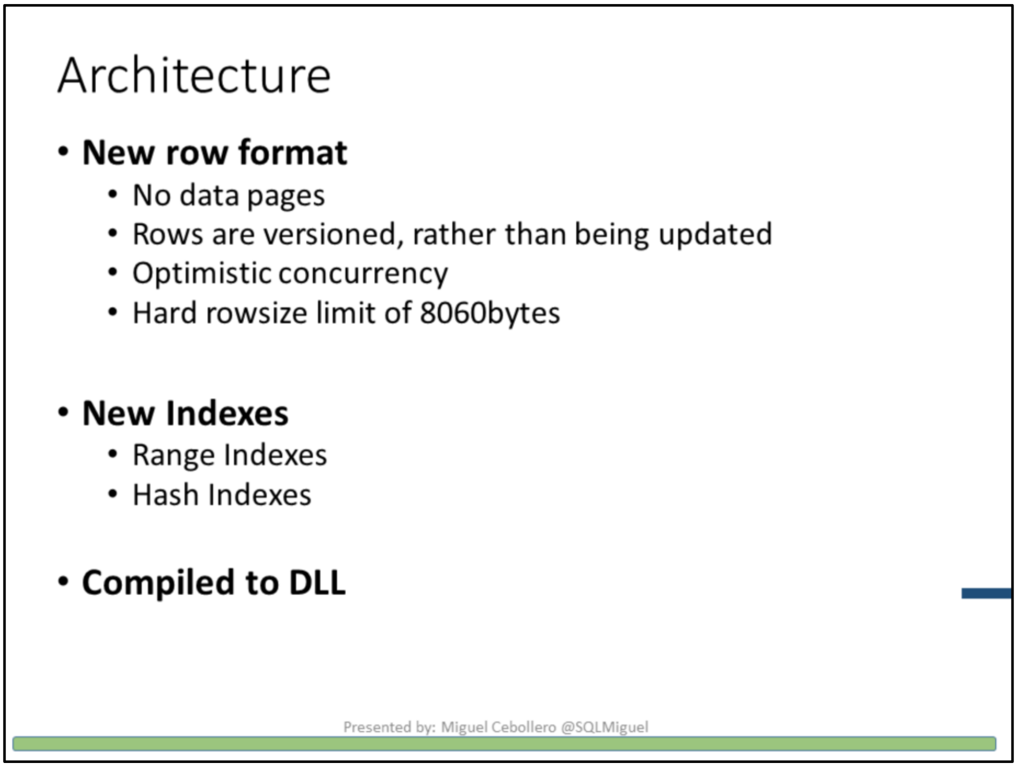
Historic RAM Prices	
Year	Average Cost per Gigabyte
1980	$6,635,520.00
1985	$ 901,120.00
1990	$ 108,544.00
1995	$ 31,641.60
2000	$ 1,149.95
2005	$ 189.44
2010	$ 12.50
2014	$ 9.34

The cost of this laptop in 1980: $53,084,160

Presented by: Miguel Cebollero @SQLMiguel

Why bother creating an in-memory database capability?
- Cost of Memory continues to plummet as the amount of memory continues to rise
- Microprocessor have hit a stagnant clock rate

- Source: Historical RAM Price; http://www.statisticbrain.com/average-historic-price-of-ram/
- Source: Moore's law; http://en.wikipedia.org/wiki/Moore's_law

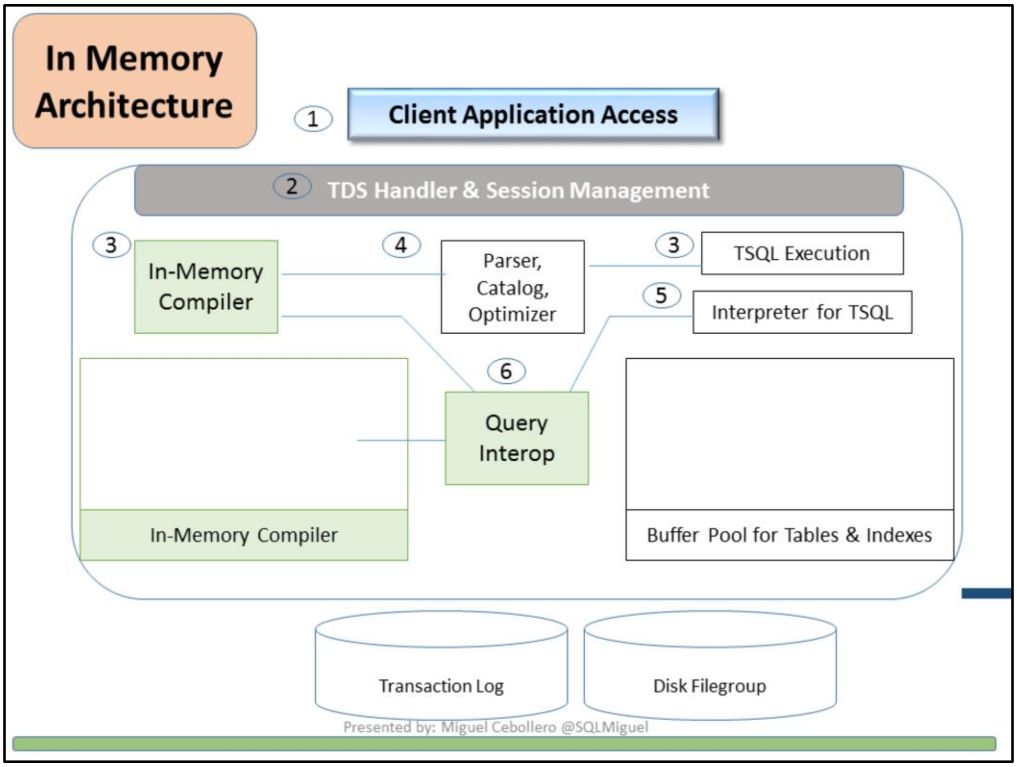


Architecture

- **New row format**
  - No data pages
  - Rows are versioned, rather than being updated
  - Optimistic concurrency
  - Hard rowsize limit of 8060bytes
- **New Indexes**
  - Range Indexes
  - Hash Indexes
- **Compiled to DLL**

Presented by: Miguel Cebollero @SQLMiguel

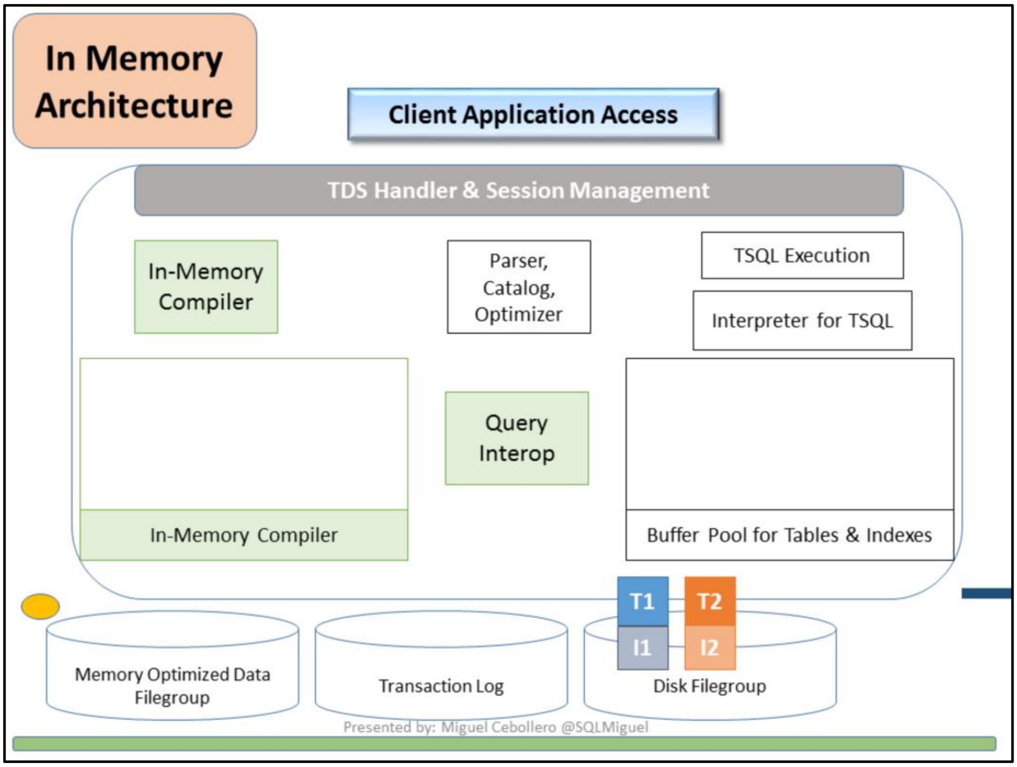
- Rows are linked via indexes in a linked list.


In Memory Architecture
1
Client Application Access
2
TDS Handler & Session Management
3
In-Memory Compiler
4
Parser, Catalog, Optimizer
3
TSQL Execution
5
Interpreter for TSQL
6
Query Interop
In-Memory Compiler
Buffer Pool for Tables & Indexes
Transaction Log
Disk Filegroup
Presented by: Miguel Cebollero @SQLMiguel

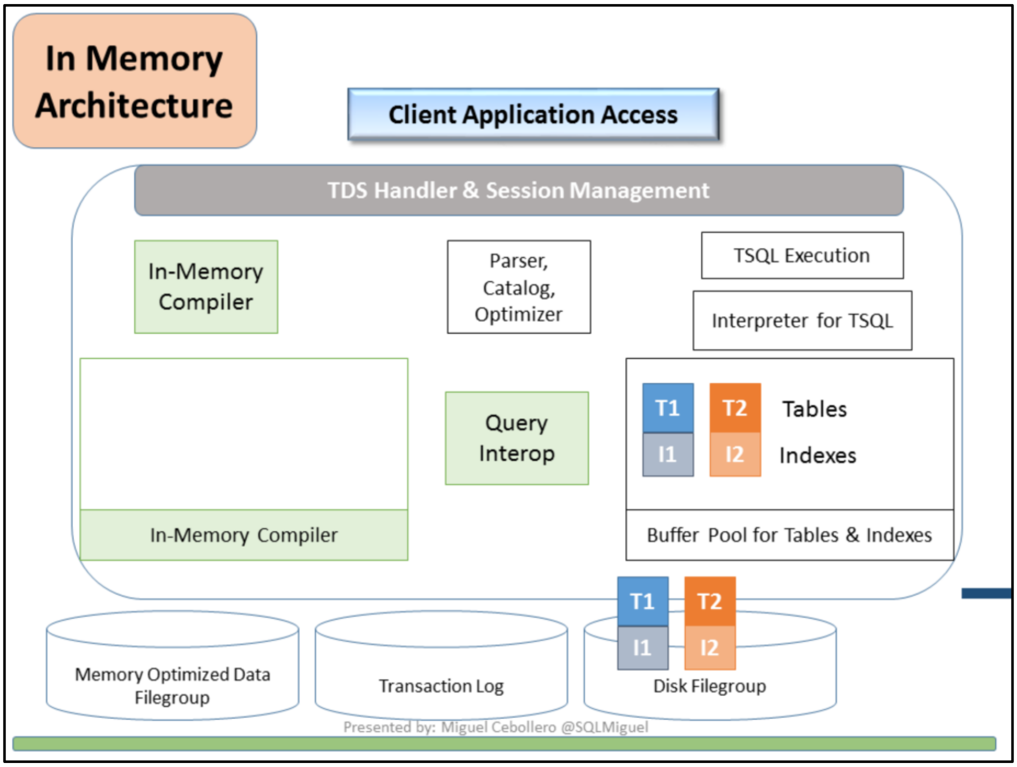
- 1. Client applications access SQL Server
- 2. Through a single TDS Handler / Session Manager
- 3. The handler decides to either access the In-Memory Compiler or TSQL Execution
- 4. The Parser is used by either In-Memory or TSQL Execution
- 5. Existing functionality
- 6. Query Interop allows traditional SQL access to new In-Memory tables

The optimizer is key, as it understands how to navigate new in-memory tables or stored procedures

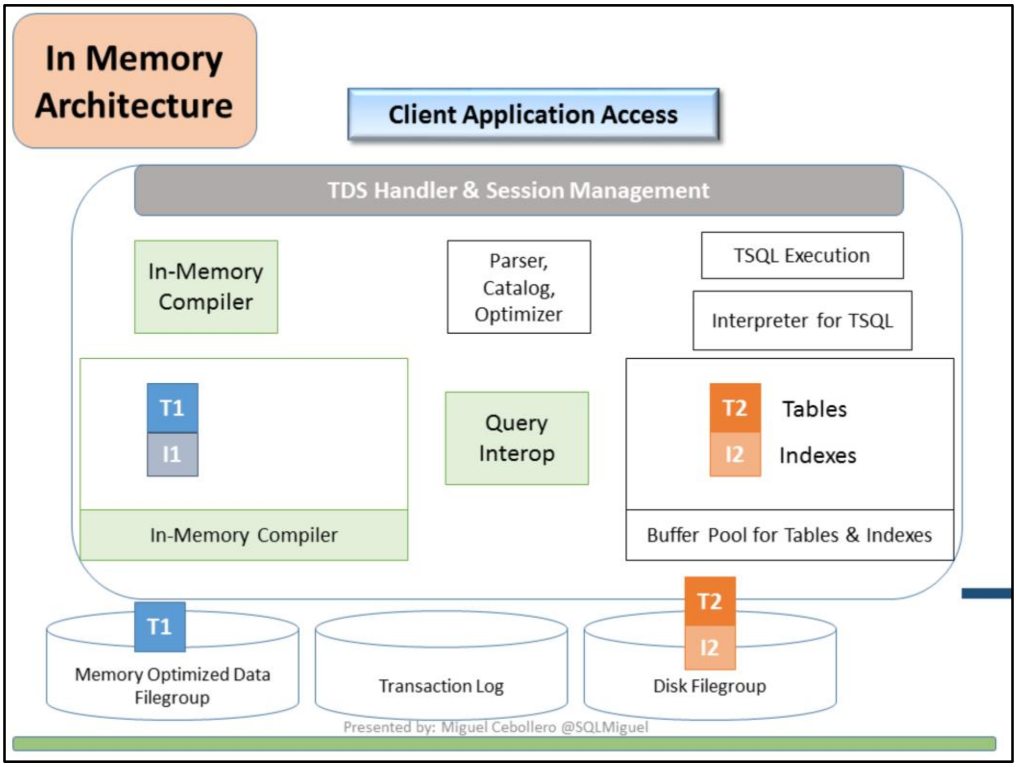
- Source of Diagram: SQL Server 2014 Mission Critical Performance Level 300 Deck.pdf; Microsoft
- Source: Microsoft SQL Server 2014 Hekaton CTP1 White Paper


In Memory Architecture
Client Application Access
TDS Handler & Session Management
In-Memory Compiler
Parser, Catalog, Optimizer
TSQL Execution
Interpreter for TSQL
Query Interop
In-Memory Compiler
Buffer Pool for Tables & Indexes
T1
T2
I1
I2
Memory Optimized Data Filegroup
Transaction Log
Disk Filegroup
Presented by: Miguel Cebollero @SQLMiguel

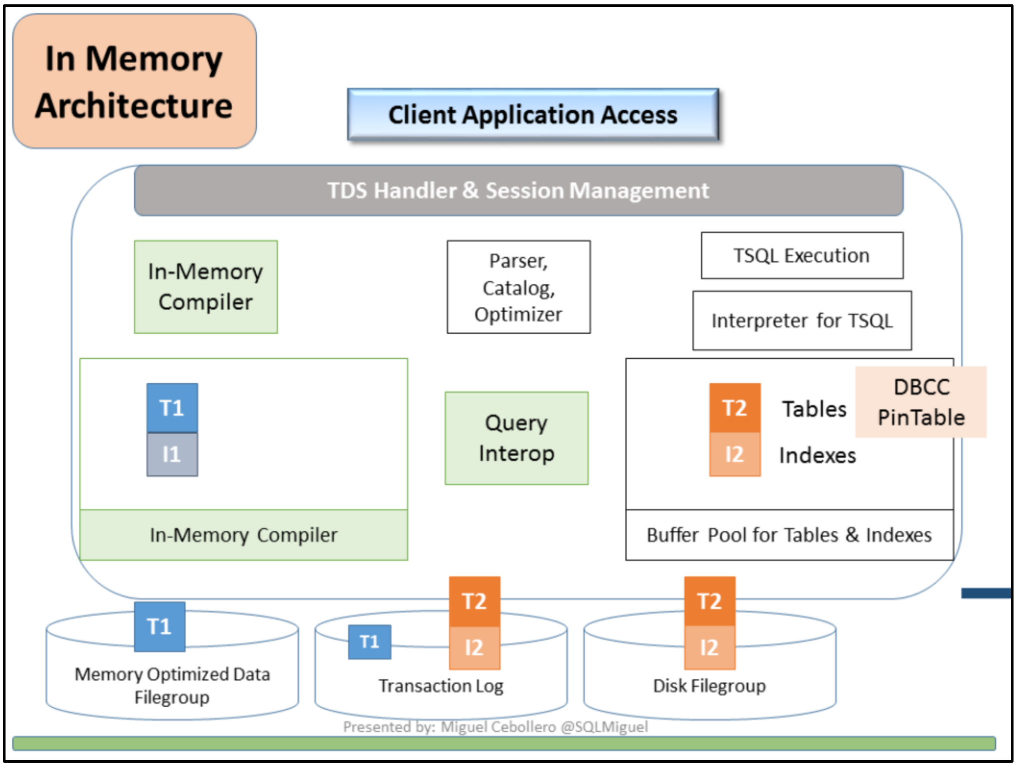
- Single Memory Optimized Data Filegroup is required
- Note: This filegroup can only be removed by dropping the database.


In Memory Architecture
Client Application Access
TDS Handler & Session Management
In-Memory Compiler
Parser, Catalog, Optimizer
TSQL Execution
Interpreter for TSQL
T1
T2
Tables
I1
I2
Indexes
Query Interop
In-Memory Compiler
Buffer Pool for Tables & Indexes
T1
T2
I1
I2
Memory Optimized Data Filegroup
Transaction Log
Disk Filegroup
Presented by: Miguel Cebollero @SQLMiguel

- Normal disk-based tables use the existing Buffer Pool to hold the rows of data needed to answer the current Client application request.


In Memory Architecture
Client Application Access
TDS Handler & Session Management
In-Memory Compiler
Parser, Catalog, Optimizer
TSQL Execution
Interpreter for TSQL
T1
I1
In-Memory Compiler
Query Interop
T2
Tables
I2
Indexes
Buffer Pool for Tables & Indexes
T1
Memory Optimized Data Filegroup
Transaction Log
T2
I2
Disk Filegroup
Presented by: Miguel Cebollero @SQLMiguel

- Table-1 was re-created as an In-Memory table.
- Notice the indexes ONLY reside in memory; therefore, not in the Filegroup. More efficient than Disk-based for this reason.
- Only the index metadata / schema resides on disk. All of the table and index data is in memory.
- On restart the index and table data is moved off of disk back into memory.


In Memory Architecture
Client Application Access
TDS Handler & Session Management
In-Memory Compiler
Parser, Catalog, Optimizer
TSQL Execution
Interpreter for TSQL
T1
I1
In-Memory Compiler
Query Interop
T2
Tables
I2
Indexes
DBCC PinTable
Buffer Pool for Tables & Indexes
T1
Memory Optimized Data Filegroup
T1
T2
I2
Transaction Log
T2
I2
Disk Filegroup
Presented by: Miguel Cebollero @SQLMiguel

- When writing to your tables there are differences between Disk-based and In-Memory
    - In-memory no index data being updated in the transaction log
    - Data written to Trans log is smaller
    - Data written to the filegroup is sequential and append only format
- Non-Durable tables will not generate any transaction logs and will not be written to the Filegroup.

Let's Get Started

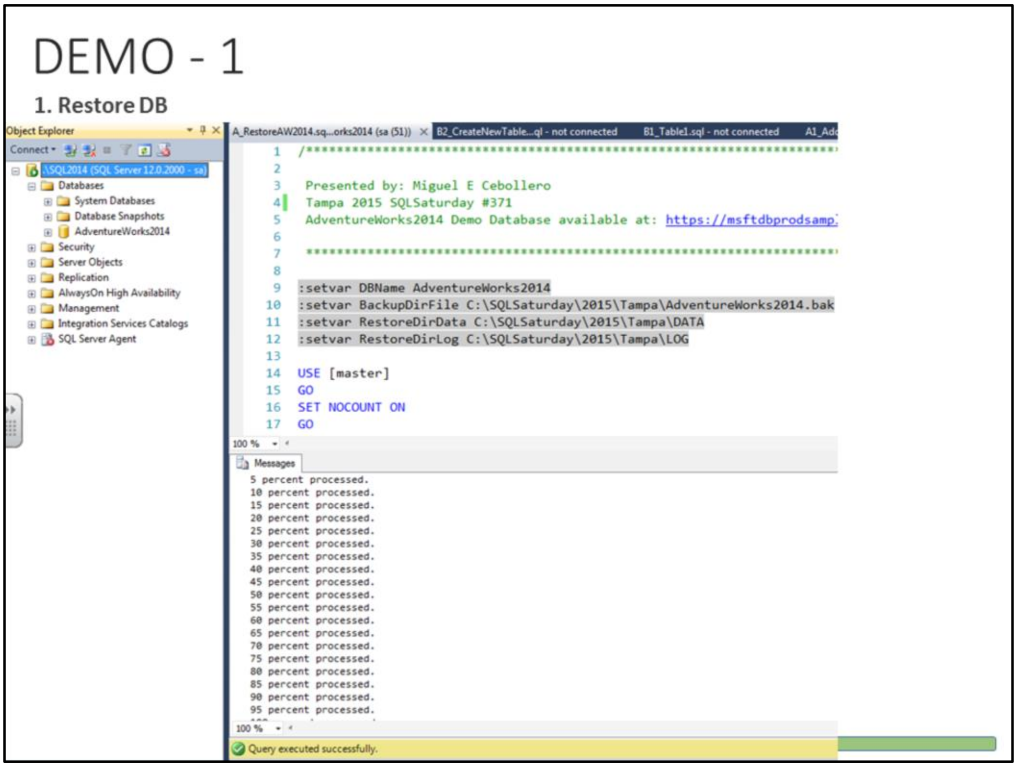
DEMO

Hardware, Durability, Architecture

Presented by: Miguel Cebollero @SQLMiguel

DEMO - 1

1. Restore DB


Object Explorer
Connect
\SQL2014 (SQL Server 12.0.2000 - sa)
Databases
System Databases
Database Snapshots
AdventureWorks2014
Security
Server Objects
Replication
AlwaysOn High Availability
Management
Integration Services Catalogs
SQL Server Agent
A_RestoreAW2014.sq...orks2014 (sa (51))
B2_CreateNewTable...ql - not connected
B1_Table1.sql - not connected
A1_Add
100 %
Messages

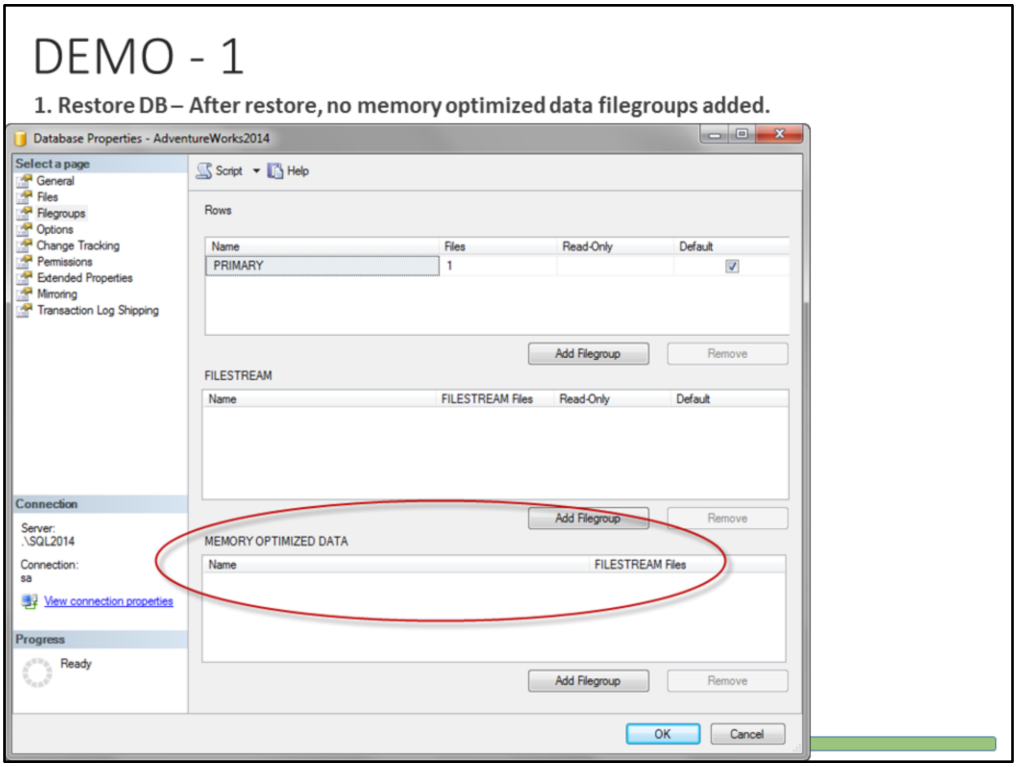
```
/**********************************************************************

 Presented by: Miguel E Cebollero
 Tampa 2015 SQLSaturday #371
 AdventureWorks2014 Demo Database available at: https://msftdbprodsamp

 ********************************************************************

:setvar DBName AdventureWorks2014
:setvar BackupDirFile C:\SQLSaturday\2015\Tampa\AdventureWorks2014.bak
:setvar RestoreDirData C:\SQLSaturday\2015\Tampa\DATA
:setvar RestoreDirLog C:\SQLSaturday\2015\Tampa\LOG

USE [master]
GO
SET NOCOUNT ON
GO
```

```
5 percent processed.
10 percent processed.
15 percent processed.
20 percent processed.
25 percent processed.
30 percent processed.
35 percent processed.
40 percent processed.
45 percent processed.
50 percent processed.
55 percent processed.
60 percent processed.
65 percent processed.
70 percent processed.
75 percent processed.
80 percent processed.
85 percent processed.
90 percent processed.
95 percent processed.
```

Query executed successfully.

```
/*****************************************************************
*********************************

 Presented by: Miguel E Cebollero
 Tampa 2015 SQLSaturday #371
 AdventureWorks2014 Demo Database available at:
https://msftdbprodsamples.codeplex.com/releases/view/125550


******************************************************************
*********************************/

:setvar DBName AdventureWorks2014
:setvar BackupDirFile C:\SQLSaturday\2015\Tampa\AdventureWorks2014.bak
:setvar RestoreDirData C:\SQLSaturday\2015\Tampa\DATA
:setvar RestoreDirLog C:\SQLSaturday\2015\Tampa\LOG

USE [master]
GO
SET NOCOUNT ON
GO
```

```
IF EXISTS (SELECT * FROM sys.databases where name='$(DBName)')
DROP DATABASE [$(DBName)];
GO

-- Restore AdventureWorks2014
RESTORE DATABASE [$(DBName)]
  FROM  DISK = N'$(BackupDirFile)'
  WITH  FILE = 1
  ,  MOVE N'AdventureWorks2014_Data' TO
N'$(RestoreDirData)\AdventureWorks2014_Data.mdf'
  ,  MOVE N'AdventureWorks2014_Log' TO
N'$(RestoreDirLog)\AdventureWorks2014_Log.ldf'
  ,  NOUNLOAD,  REPLACE,  STATS = 5;
GO

ALTER AUTHORIZATION ON DATABASE::[$(DBName)] TO [SA];
GO

ALTER DATABASE [$(DBName)]
SET MULTI_USER
GO

USE AdventureWorks2014;
GO
 UPDATE STATISTICS [Person].[Address];
 GO
```



DEMO - 1

1. Restore DB – After restore, no memory optimized data filegroups added.

Database Properties - AdventureWorks2014

Rows

Name	Files	Read-Only	Default
PRIMARY	1		☑

FILESTREAM

Name	FILESTREAM Files	Read-Only	Default

MEMORY OPTIMIZED DATA

Name	FILESTREAM Files

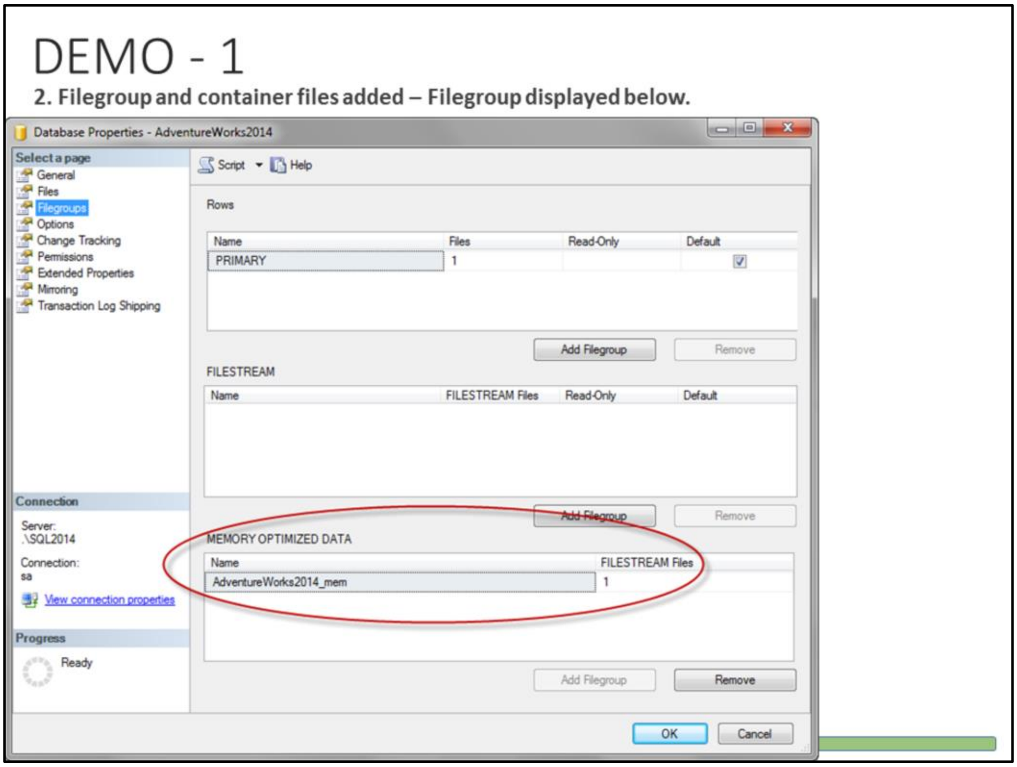
Connection

Server:
.\SQL2014

Connection:
sa

```
/*********************************************************
*****************************

 Presented by: Miguel E Cebollero
 Tampa 2015 SQLSaturday #371
 AdventureWorks2014 Demo Database available at:
https://msftdbprodsamples.codeplex.com/releases/view/125550


**********************************************************
****************************/

:setvar DBName AdventureWorks2014
:setvar BackupDirFile C:\SQLSaturday\2015\Tampa\AdventureWorks2014.bak
:setvar RestoreDirData C:\SQLSaturday\2015\Tampa\DATA
:setvar RestoreDirLog C:\SQLSaturday\2015\Tampa\LOG

USE [master]
GO
SET NOCOUNT ON
GO
```

```
IF EXISTS (SELECT * FROM sys.databases where name='$(DBName)')
DROP DATABASE [$(DBName)];
GO

-- Restore AdventureWorks2014
RESTORE DATABASE [$(DBName)]
  FROM  DISK = N'$(BackupDirFile)'
  WITH  FILE = 1
  ,  MOVE N'AdventureWorks2014_Data' TO
N'$(RestoreDirData)\AdventureWorks2014_Data.mdf'
  ,  MOVE N'AdventureWorks2014_Log' TO
N'$(RestoreDirLog)\AdventureWorks2014_Log.ldf'
  ,  NOUNLOAD,  REPLACE,  STATS = 5;
GO

ALTER AUTHORIZATION ON DATABASE::[$(DBName)] TO [SA];
GO

ALTER DATABASE [$(DBName)]
SET MULTI_USER
GO

USE AdventureWorks2014;
GO
 UPDATE STATISTICS [Person].[Address];
 GO
```



DEMO - 1

2. Filegroup and container files added – Filegroup displayed below.

Database Properties - AdventureWorks2014

Rows

Name	Files	Read-Only	Default
PRIMARY	1		☑

FILESTREAM

Name	FILESTREAM Files	Read-Only	Default

MEMORY OPTIMIZED DATA

Name	FILESTREAM Files
AdventureWorks2014_mem	1

Connection

Server:
.\SQL2014

Connection:
sa

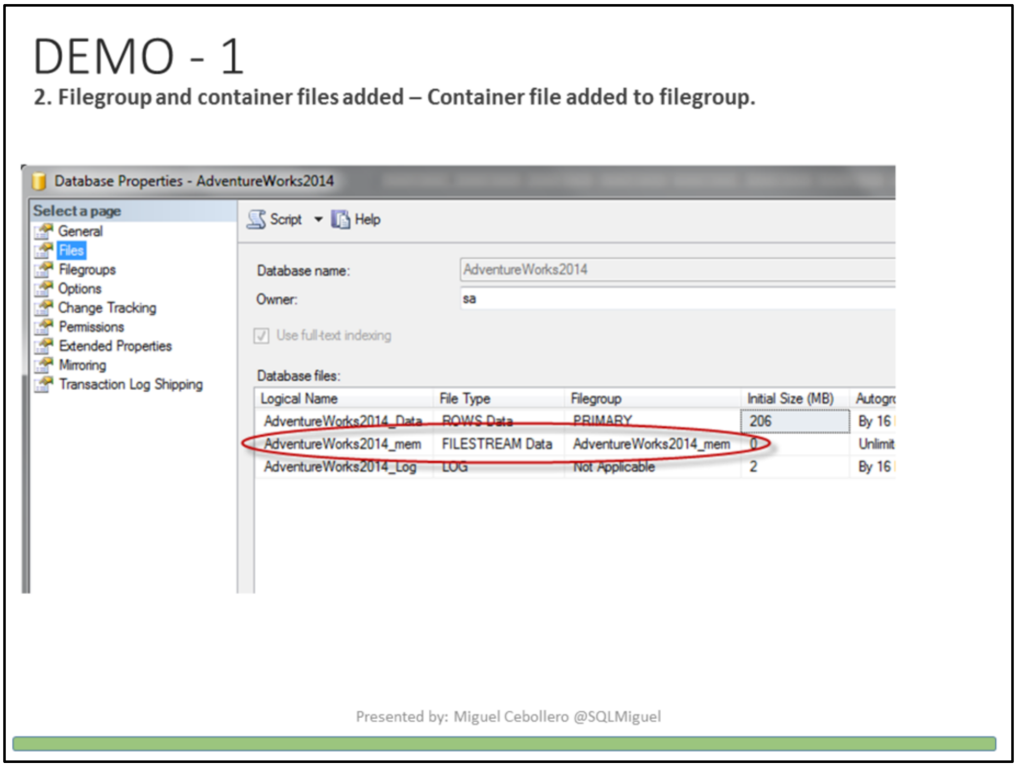
View connection properties

Progress

Ready

```
/***********************************************************
************************

 Notes: Add MEMORY_OPTIMIZED_DATA filegroup and container to enable in-
memory OLTP in the database
This code can be used to alter other existing databases.


************************************************************
***********************/

:setvar DBName "AdventureWorks2014"
:setvar CheckPointDir "C:\SQLSaturday\2015\Tampa\DATA\"

-- Create FILEGROUP
-- Can only have one memory optimized filegroup per database
IF NOT EXISTS (SELECT * FROM $(DBName).sys.data_spaces WHERE type='FX') --<
InMemory abbreviationI
ALTER DATABASE $(DBName)
  ADD FILEGROUP [$(DBName)_mem] CONTAINS MEMORY_OPTIMIZED_DATA  --<
*** This is where all the magic happens ***
```

```
GO

-- Create New File and add it to the new Filegroup
-- You can have multiple containers (files) per memory optimized filegroup
-- Advantage of multiple containers is for recovery purposes
IF NOT EXISTS (SELECT * FROM $(DBName).sys.data_spaces ds
JOIN $(DBName).sys.database_files df ON
 ds.data_space_id=df.data_space_id WHERE ds.type='FX') --< InMemory abbreviationI
ALTER DATABASE $(DBName)
  ADD FILE (name='$(DBName)_mem', filename='$(CheckPointDir)$(DBName)_mem')
  TO FILEGROUP [$(DBName)_mem]
GO

-- Navigate to file directory to see new files created

-- C:\SQLSaturday\2015\Tampa\DATA

-- Take a look at the properties of the database via SSMS

/*
-- Take a look as the filegroup properties via system view

 SELECT Name, Type, Type_Desc
   FROM AdventureWorks2014.sys.data_spaces
  WHERE type='FX' --< InMemory abbreviationI

*/
```



DEMO - 1

2. Filegroup and container files added – Container file added to filegroup.

Database name: AdventureWorks2014

Owner: sa

☑ Use full-text indexing

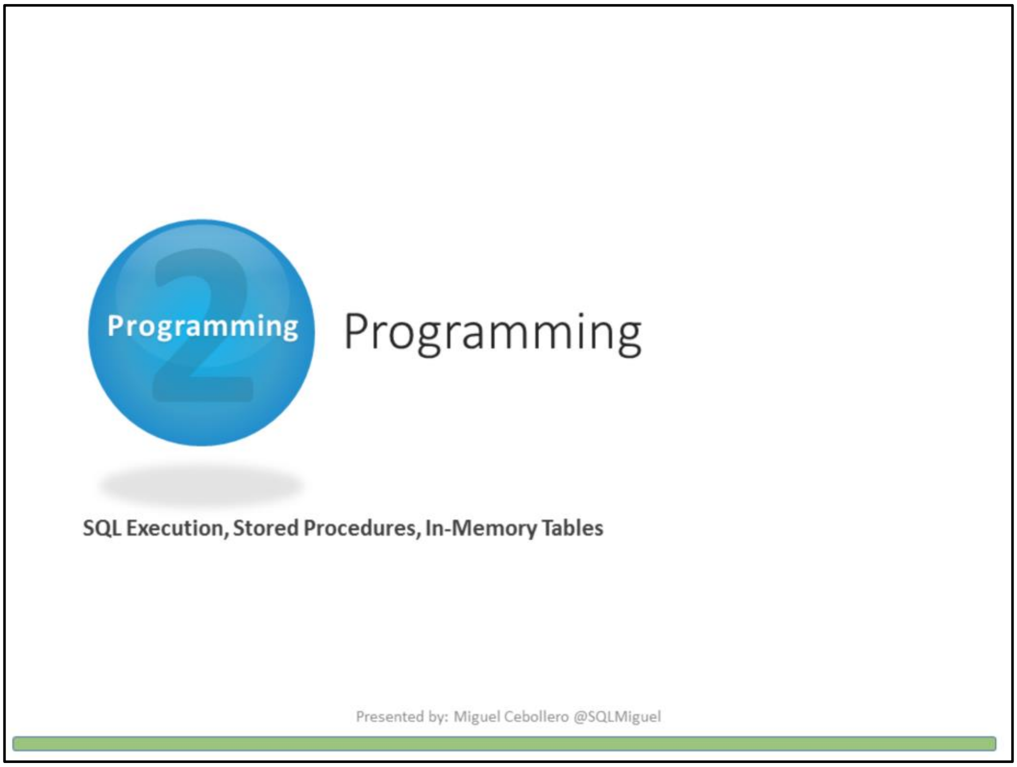
Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogr
AdventureWorks2014_Data	ROWS Data	PRIMARY	206	By 16
AdventureWorks2014_mem	FILESTREAM Data	AdventureWorks2014_mem	0	Unlimit
AdventureWorks2014_Log	LOG	Not Applicable	2	By 16

Presented by: Miguel Cebollero @SQLMiguel

```
/***********************************************************************************************
 Notes: Add MEMORY_OPTIMIZED_DATA filegroup and container to enable in-memory OLTP in the database
This code can be used to alter other existing databases.


***********************************************************************************************/

:setvar DBName "AdventureWorks2014"
:setvar CheckPointDir "C:\SQLSaturday\2015\Tampa\DATA\"

-- Create FILEGROUP
-- Can only have one memory optimized filegroup per database
IF NOT EXISTS (SELECT * FROM $(DBName).sys.data_spaces WHERE type='FX') --< InMemory abbreviationI
ALTER DATABASE $(DBName)
  ADD FILEGROUP [$(DBName)_mem] CONTAINS MEMORY_OPTIMIZED_DATA  --< *** This is where all the magic happens ***
```

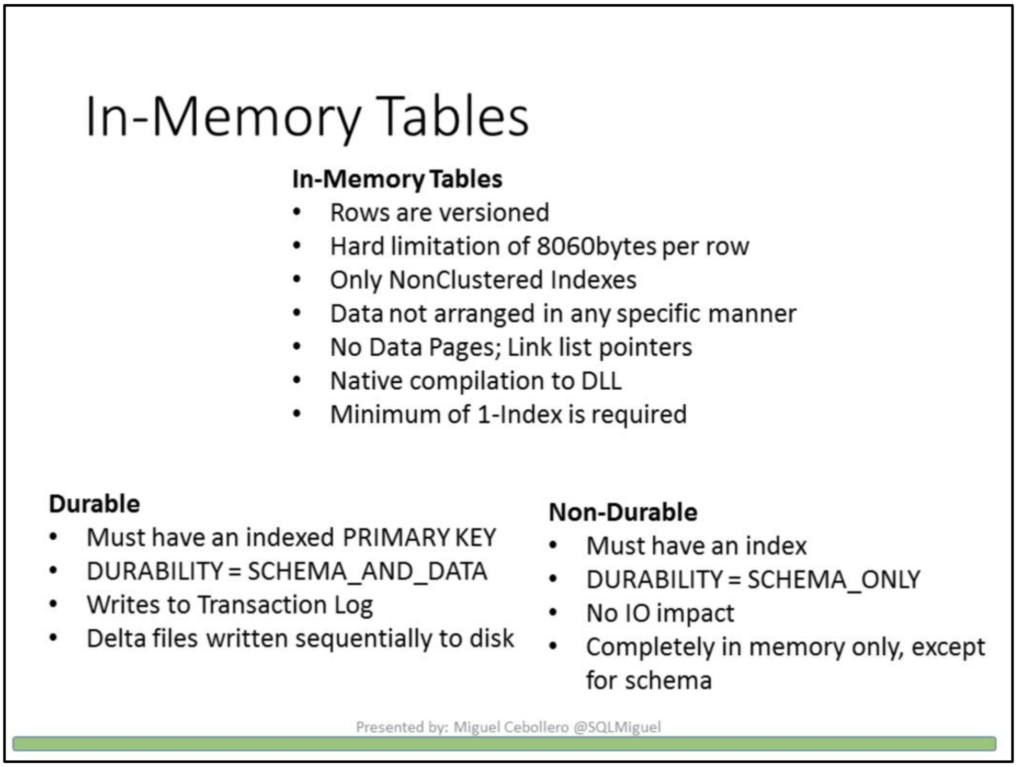
```
GO

-- Create New File and add it to the new Filegroup
-- You can have multiple containers (files) per memory optimized filegroup
-- Advantage of multiple containers is for recovery purposes
IF NOT EXISTS (SELECT * FROM $(DBName).sys.data_spaces ds
JOIN $(DBName).sys.database_files df ON
 ds.data_space_id=df.data_space_id WHERE ds.type='FX') --< InMemory abbreviationI
ALTER DATABASE $(DBName)
  ADD FILE (name='$(DBName)_mem', filename='$(CheckPointDir)$(DBName)_mem')
  TO FILEGROUP [$(DBName)_mem]
GO

-- Navigate to file directory to see new files created

-- C:\SQLSaturday\2015\Tampa\DATA

-- Take a look at the properties of the database via SSMS

/*
-- Take a look as the filegroup properties via system view

 SELECT Name, Type, Type_Desc
   FROM AdventureWorks2014.sys.data_spaces
  WHERE type='FX' --< InMemory abbreviationI

*/
```


2
Programming

Programming

SQL Execution, Stored Procedures, In-Memory Tables

Presented by: Miguel Cebollero @SQLMiguel



In-Memory Tables

In-Memory Tables

- Rows are versioned
- Hard limitation of 8060bytes per row
- Only NonClustered Indexes
- Data not arranged in any specific manner
- No Data Pages; Link list pointers
- Native compilation to DLL
- Minimum of 1-Index is required

Durable

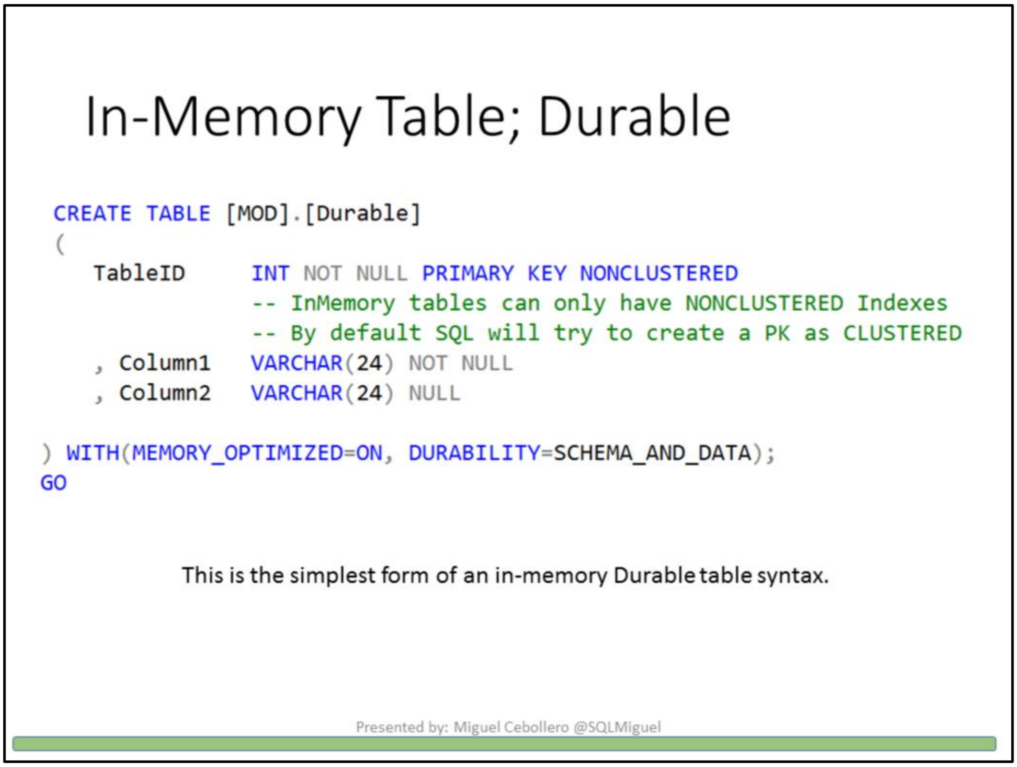
- Must have an indexed PRIMARY KEY
- DURABILITY = SCHEMA_AND_DATA
- Writes to Transaction Log
- Delta files written sequentially to disk

Non-Durable

- Must have an index
- DURABILITY = SCHEMA_ONLY
- No IO impact
- Completely in memory only, except for schema

Presented by: Miguel Cebollero @SQLMiguel

- Each row can have multiple versions, to allow concurrent reads and writes on the same row.
- The Indexes are what links the multiple rows into a table
- Tables cannot be altered. They would need to be dropped and recreated.
- http://msdn.microsoft.com/en-us/library/dn511014.aspx



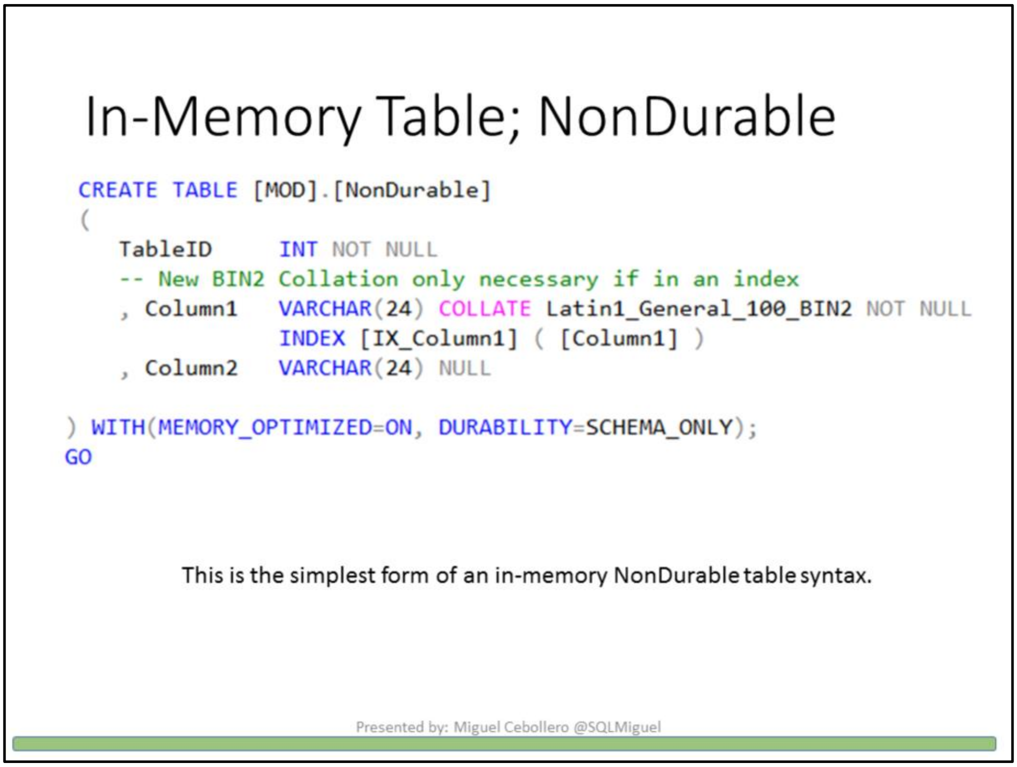
In-Memory Table; Durable

```
CREATE TABLE [MOD].[Durable]
(
   TableID     INT NOT NULL PRIMARY KEY NONCLUSTERED
               -- InMemory tables can only have NONCLUSTERED Indexes
               -- By default SQL will try to create a PK as CLUSTERED
   , Column1   VARCHAR(24) NOT NULL
   , Column2   VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

This is the simplest form of an in-memory Durable table syntax.

Presented by: Miguel Cebollero @SQLMiguel

- Syntax, MSDN reference: http://msdn.microsoft.com/en-us/library/dn133186.aspx



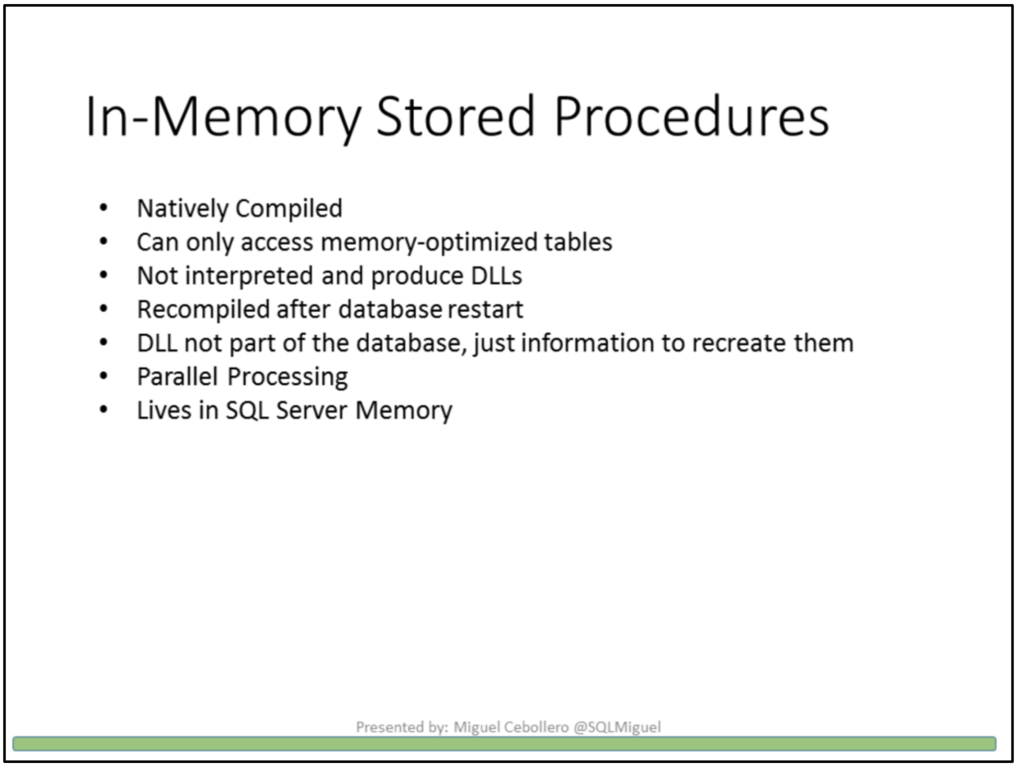
In-Memory Table; NonDurable

```
CREATE TABLE [MOD].[NonDurable]
(
   TableID     INT NOT NULL
   -- New BIN2 Collation only necessary if in an index
   , Column1   VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
               INDEX [IX_Column1] ( [Column1] )
   , Column2   VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

This is the simplest form of an in-memory NonDurable table syntax.

Presented by: Miguel Cebollero @SQLMiguel

- Syntax, MSDN reference: http://msdn.microsoft.com/en-us/library/dn133186.aspx

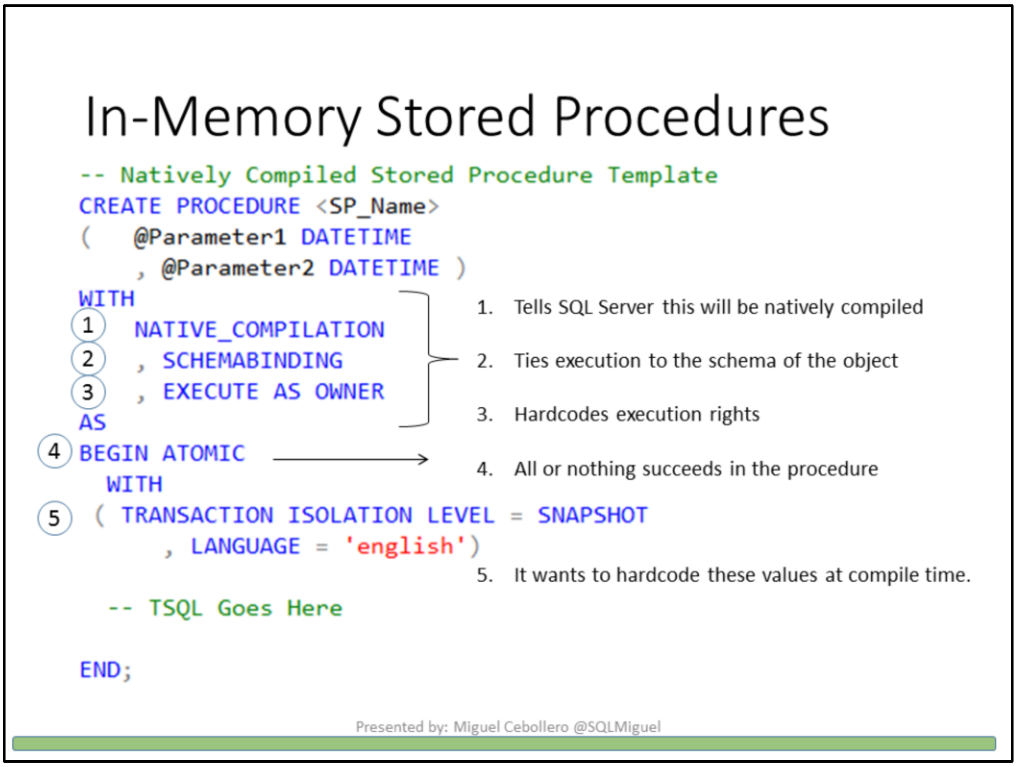


In-Memory Stored Procedures

- Natively Compiled
- Can only access memory-optimized tables
- Not interpreted and produce DLLs
- Recompiled after database restart
- DLL not part of the database, just information to recreate them
- Parallel Processing
- Lives in SQL Server Memory

Presented by: Miguel Cebollero @SQLMiguel

- This offers 10x or more performance gains. More gains for more complicated scenarios.
- If you need to access a mix between disk-based and memory optimized tables, you need to use traditional interpreted stored procedures.
-



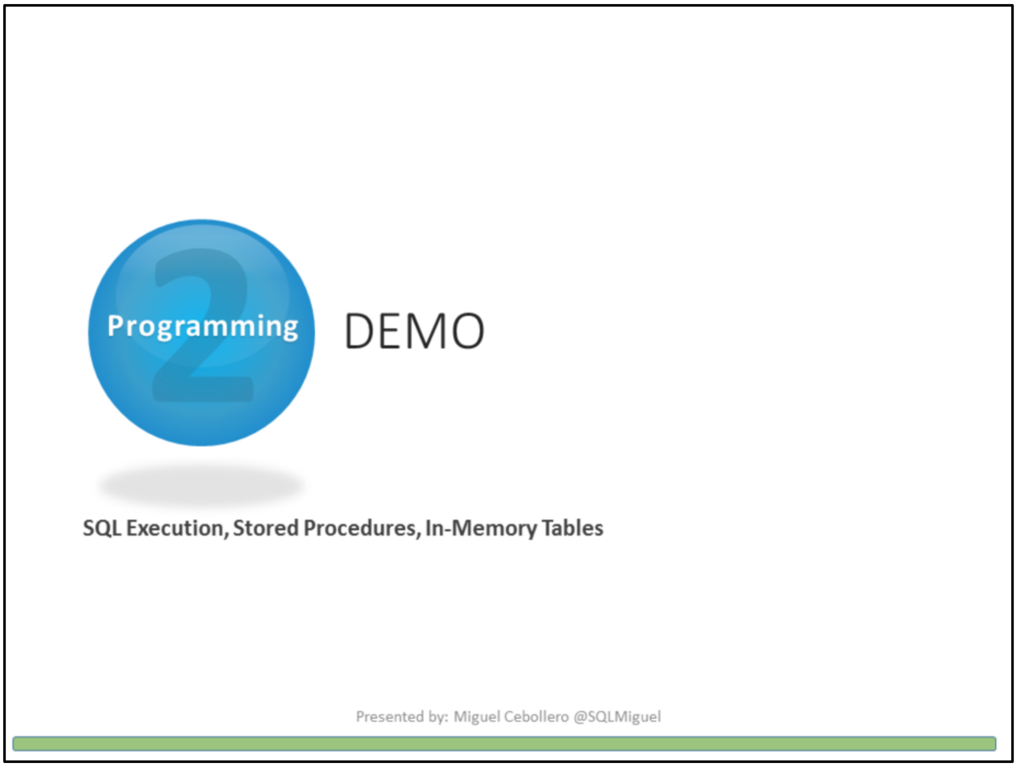
In-Memory Stored Procedures

```
    -- Natively Compiled Stored Procedure Template
    CREATE PROCEDURE <SP_Name>
    (   @Parameter1 DATETIME
      , @Parameter2 DATETIME )
    WITH
 ①      NATIVE_COMPILATION
 ②    , SCHEMABINDING
 ③    , EXECUTE AS OWNER
    AS
④  BEGIN ATOMIC
      WITH
⑤   ( TRANSACTION ISOLATION LEVEL = SNAPSHOT
        , LANGUAGE = 'english')

      -- TSQL Goes Here

    END;
```

1. Tells SQL Server this will be natively compiled
2. Ties execution to the schema of the object
3. Hardcodes execution rights
4. All or nothing succeeds in the procedure
5. It wants to hardcode these values at compile time.

Presented by: Miguel Cebollero @SQLMiguel

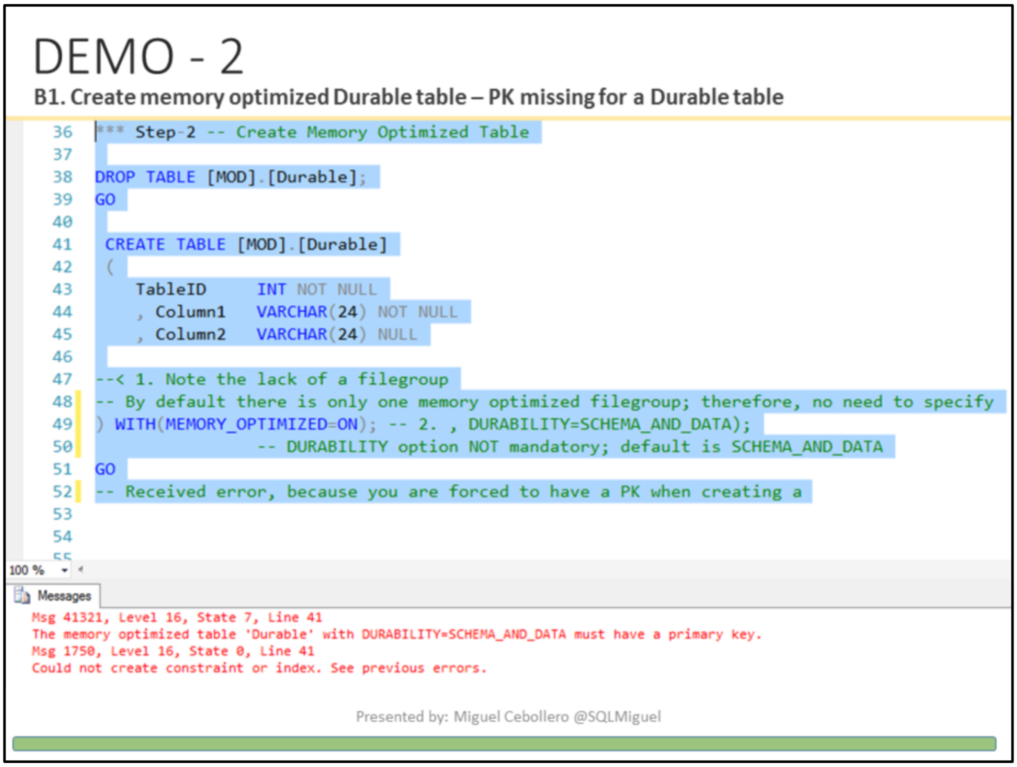
- This offers 10x or more performance gains. More gains for more complicated scenarios.
- If you need to access a mix between disk-based and memory optimized tables, you need to use traditional interpreted stored procedures.
-



DEMO

SQL Execution, Stored Procedures, In-Memory Tables

Presented by: Miguel Cebollero @SQLMiguel



DEMO - 2

B1. Create memory optimized Durable table – PK missing for a Durable table

```
36  *** Step-2 -- Create Memory Optimized Table
37
38  DROP TABLE [MOD].[Durable];
39  GO
40
41   CREATE TABLE [MOD].[Durable]
42   (
43      TableID     INT NOT NULL
44      , Column1   VARCHAR(24) NOT NULL
45      , Column2   VARCHAR(24) NULL
46
47  --< 1. Note the lack of a filegroup
48  -- By default there is only one memory optimized filegroup; therefore, no need to specify
49  ) WITH(MEMORY_OPTIMIZED=ON); -- 2. , DURABILITY=SCHEMA_AND_DATA);
50                -- DURABILITY option NOT mandatory; default is SCHEMA_AND_DATA
51  GO
52  -- Received error, because you are forced to have a PK when creating a
53
54
```

```
Msg 41321, Level 16, State 7, Line 41
The memory optimized table 'Durable' with DURABILITY=SCHEMA_AND_DATA must have a primary key.
Msg 1750, Level 16, State 0, Line 41
Could not create constraint or index. See previous errors.
```

Presented by: Miguel Cebollero @SQLMiguel

```
USE AdventureWorks2014;
GO

/*******************************************

Create new Memory Optimized Table

*******************************************/

-- Create new schema for the purposes of comparision with disk-based table
CREATE SCHEMA [MOD] AUTHORIZATION [dbo]; -- Memory Optimized Data
GO

*** Step-1 -- Create Disk-based table

-- DROP TABLE [MOD].[Durable]

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

) ON [PRIMARY] --< Note the filegroup
GO
-- No indexes, No PK, Just a Heap
```

```
*** Step-2 -- Create Memory Optimized Table

DROP TABLE [MOD].[Durable];
GO

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

--< 1. Note the lack of a filegroup
) WITH(MEMORY_OPTIMIZED=ON); -- 2. , DURABILITY=SCHEMA_AND_DATA);
-- DURABILITY option NOT mandatory
--  default is SCHEMA_AND_DATA
GO
```

```
*** Step-3 -- Missing PRIMARY KEY

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY --< Add our Primary Key
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
*** Step-4 -- PRIMARY KEY fix; NONCLUSTERED
   -- Simplest syntax for a DURABLE
   --in-memory table.
 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY NONCLUSTERED
-- InMemory tables can only have NONCLUSTERED Indexes
-- By default SQL will try to create a PK as CLUSTERED
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
```

```
GO
```

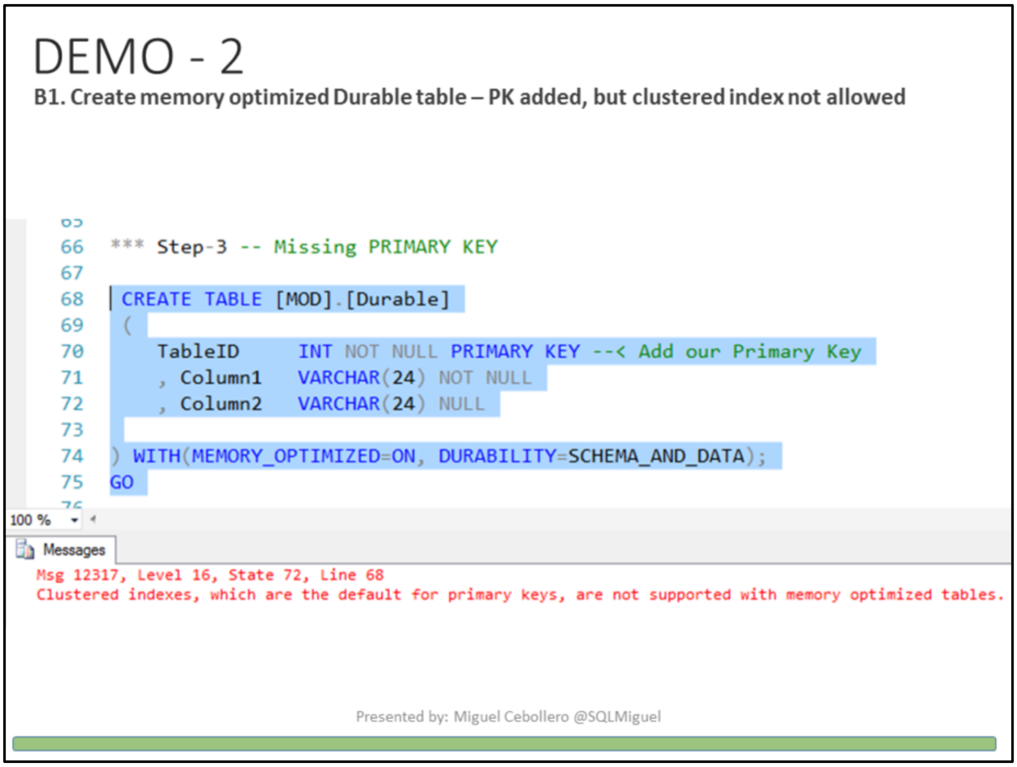
```
*** Step-5 -- NonDurable

 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- 1) New Inline syntax for SQL2014
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

, INDEX [IX_Column2] ( [Column2] ) -- 2) Can still declare index after columns
-- 3) NONCLUSTERED
--a. The "NONCLUSTERED" hint is optional,
--            unless being defined against the primary key

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
--< SCHEMA_ONLY option; literally no safety net
-- Use cases: Staging, Website Session State
GO
```

*** Step-6 -- NonDurable; Fix NULLABLE Column

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NOT NULL --< Fix NULLABLE Column
, Column3VARCHAR(3850) NULL --< Fix our row size limitation issues
, Column4VARCHAR(3850) NULL --< Fix our row size limitation issues

, INDEX [IX_Column2] ( [Column2] )

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

*** Step-7 -- NonDurable; Fix BIN2 collation
-- Simplest form of a NonDurable table

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- New BIN2 Collation
, Column1VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
INDEX [IX_Column1] ( [Column1] )
-- New BIN2 Collation only necessary if in an index
, Column2VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
, Column3VARCHAR(3850) NULL
, Column4VARCHAR(3850) NULL

, INDEX [IX_Column2] ( [Column2] )
-- The string data type must be defined using a BIN2 collation

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO




/*  -- Look at our new table in an existing system view
    -- New 2014 columns are available

 select t.name as 'Table Name'
, t.schema_id
, t.object_id
, filestream_data_space_id
, is_memory_optimized
, durability
, durability_desc
   from sys.tables t
  where type='U'
    andt.schema_id = SCHEMA_ID(N'MOD');

*/


-- Look at our table via SSMS properties


-- Location of the dll files that represent the structure of the table we just created
-- C:\Program Files\Microsoft SQL Server\MSSQL12.SQL2014\MSSQL\DATA
```



DEMO - 2

B1. Create memory optimized Durable table – PK added, but clustered index not allowed

```
*** Step-3 -- Missing PRIMARY KEY

CREATE TABLE [MOD].[Durable]
(
    TableID      INT NOT NULL PRIMARY KEY --< Add our Primary Key
    , Column1    VARCHAR(24) NOT NULL
    , Column2    VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
Msg 12317, Level 16, State 72, Line 68
Clustered indexes, which are the default for primary keys, are not supported with memory optimized tables.
```

Presented by: Miguel Cebollero @SQLMiguel

```
USE AdventureWorks2014;
GO

/********************************************

Create new Memory Optimized Table

********************************************/

-- Create new schema for the purposes of comparision with disk-based table
CREATE SCHEMA [MOD] AUTHORIZATION [dbo]; -- Memory Optimized Data
GO

*** Step-1 -- Create Disk-based table

-- DROP TABLE [MOD].[Durable]

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

) ON [PRIMARY] --< Note the filegroup
GO
-- No indexes, No PK, Just a Heap
```

```
*** Step-2 -- Create Memory Optimized Table

DROP TABLE [MOD].[Durable];
GO

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

--< 1. Note the lack of a filegroup
) WITH(MEMORY_OPTIMIZED=ON); -- 2. , DURABILITY=SCHEMA_AND_DATA);
-- DURABILITY option NOT mandatory
--  default is SCHEMA_AND_DATA
GO
```

```
*** Step-3 -- Missing PRIMARY KEY

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY --< Add our Primary Key
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
*** Step-4 -- PRIMARY KEY fix; NONCLUSTERED
   -- Simplest syntax for a DURABLE
   --in-memory table.
 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY NONCLUSTERED
-- InMemory tables can only have NONCLUSTERED Indexes
-- By default SQL will try to create a PK as CLUSTERED
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
```

```
GO
```

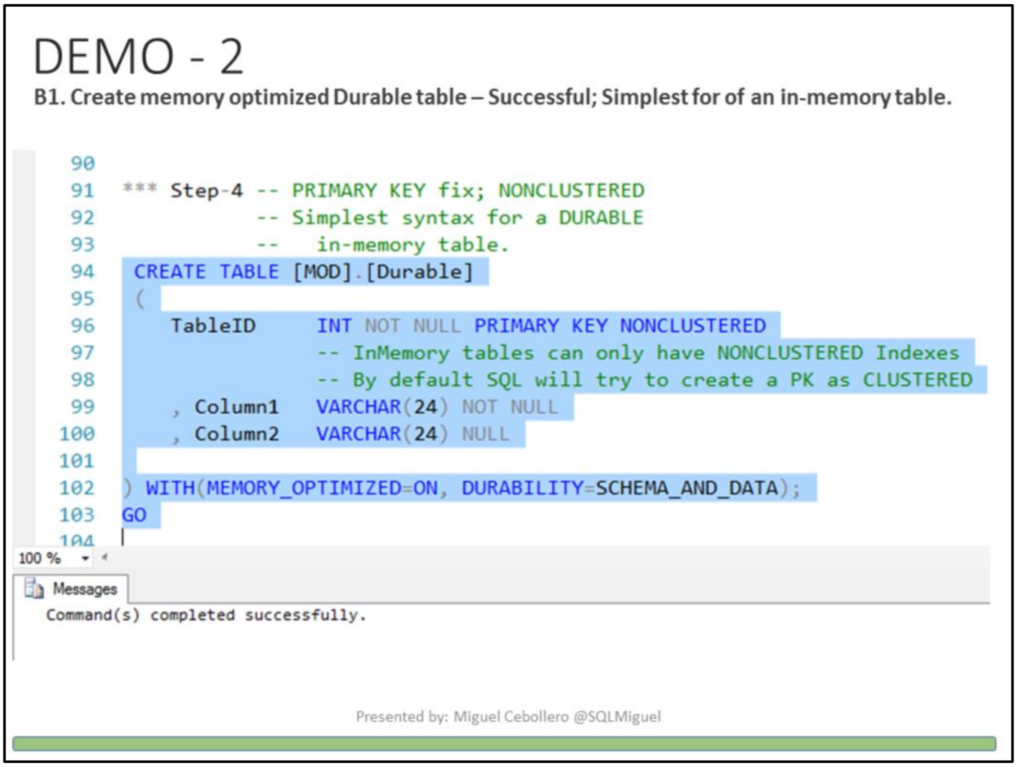
*** Step-5 -- NonDurable

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- 1) New Inline syntax for SQL2014
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

, INDEX [IX_Column2] ( [Column2] ) -- 2) Can still declare index after columns
-- 3) NONCLUSTERED
--a. The "NONCLUSTERED" hint is optional,
--              unless being defined against the primary key

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
--< SCHEMA_ONLY option; literally no safety net
-- Use cases: Staging, Website Session State
GO
```

```
*** Step-6 -- NonDurable; Fix NULLABLE Column

 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NOT NULL --< Fix NULLABLE Column
, Column3VARCHAR(3850) NULL --< Fix our row size limitation issues
, Column4VARCHAR(3850) NULL --< Fix our row size limitation issues

, INDEX [IX_Column2] ( [Column2] )

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

```
*** Step-7 -- NonDurable; Fix BIN2 collation
-- Simplest form of a NonDurable table
```

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- New BIN2 Collation
, Column1VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
INDEX [IX_Column1] ( [Column1] )
-- New BIN2 Collation only necessary if in an index
, Column2VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
, Column3VARCHAR(3850) NULL
, Column4VARCHAR(3850) NULL

, INDEX [IX_Column2] ( [Column2] )
-- The string data type must be defined using a BIN2 collation

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO




/*  -- Look at our new table in an existing system view
    -- New 2014 columns are available

 select t.name as 'Table Name'
, t.schema_id
, t.object_id
, filestream_data_space_id
, is_memory_optimized
, durability
, durability_desc
   from sys.tables t
  where type='U'
    andt.schema_id = SCHEMA_ID(N'MOD');

*/


-- Look at our table via SSMS properties


-- Location of the dll files that represent the structure of the table we just created
-- C:\Program Files\Microsoft SQL Server\MSSQL12.SQL2014\MSSQL\DATA
```



DEMO - 2

B1. Create memory optimized Durable table – Successful; Simplest for of an in-memory table.

```
*** Step-4 -- PRIMARY KEY fix; NONCLUSTERED
           -- Simplest syntax for a DURABLE
           --   in-memory table.
CREATE TABLE [MOD].[Durable]
(
   TableID      INT NOT NULL PRIMARY KEY NONCLUSTERED
                -- InMemory tables can only have NONCLUSTERED Indexes
                -- By default SQL will try to create a PK as CLUSTERED
   , Column1    VARCHAR(24) NOT NULL
   , Column2    VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
Command(s) completed successfully.
```

Presented by: Miguel Cebollero @SQLMiguel

```
USE AdventureWorks2014;
GO

/*******************************************

Create new Memory Optimized Table

*******************************************/

-- Create new schema for the purposes of comparision with disk-based table
CREATE SCHEMA [MOD] AUTHORIZATION [dbo]; -- Memory Optimized Data
GO

*** Step-1 -- Create Disk-based table

-- DROP TABLE [MOD].[Durable]

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

) ON [PRIMARY] --< Note the filegroup
GO
-- No indexes, No PK, Just a Heap
```

```
*** Step-2 -- Create Memory Optimized Table

DROP TABLE [MOD].[Durable];
GO

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

--< 1. Note the lack of a filegroup
) WITH(MEMORY_OPTIMIZED=ON); -- 2. , DURABILITY=SCHEMA_AND_DATA);
-- DURABILITY option NOT mandatory
--  default is SCHEMA_AND_DATA
GO
```

```
*** Step-3 -- Missing PRIMARY KEY

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY --< Add our Primary Key
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
*** Step-4 -- PRIMARY KEY fix; NONCLUSTERED
   -- Simplest syntax for a DURABLE
   --in-memory table.
 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY NONCLUSTERED
-- InMemory tables can only have NONCLUSTERED Indexes
-- By default SQL will try to create a PK as CLUSTERED
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
```

```
GO
```

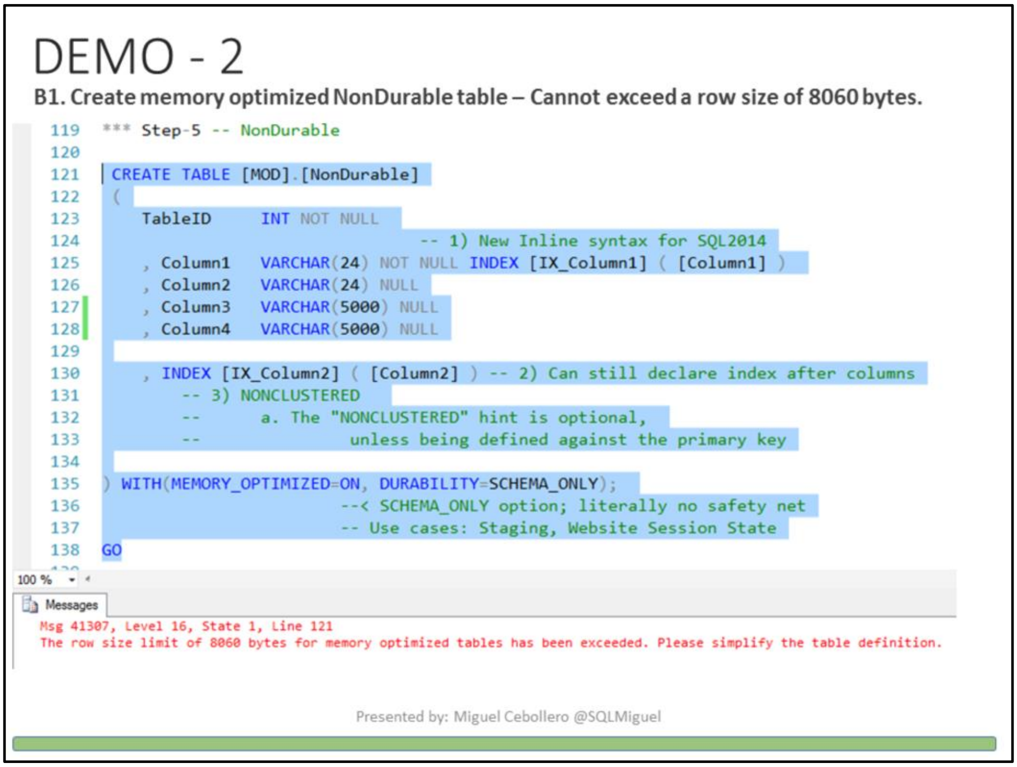
```
*** Step-5 -- NonDurable

 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- 1) New Inline syntax for SQL2014
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

, INDEX [IX_Column2] ( [Column2] ) -- 2) Can still declare index after columns
-- 3) NONCLUSTERED
--a. The "NONCLUSTERED" hint is optional,
--            unless being defined against the primary key

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
--< SCHEMA_ONLY option; literally no safety net
-- Use cases: Staging, Website Session State
GO
```

*** Step-6 -- NonDurable; Fix NULLABLE Column

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NOT NULL --< Fix NULLABLE Column
, Column3VARCHAR(3850) NULL --< Fix our row size limitation issues
, Column4VARCHAR(3850) NULL --< Fix our row size limitation issues

, INDEX [IX_Column2] ( [Column2] )

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

*** Step-7 -- NonDurable; Fix BIN2 collation
-- Simplest form of a NonDurable table

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- New BIN2 Collation
, Column1VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
INDEX [IX_Column1] ( [Column1] )
-- New BIN2 Collation only necessary if in an index
, Column2VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
, Column3VARCHAR(3850) NULL
, Column4VARCHAR(3850) NULL

, INDEX [IX_Column2] ( [Column2] )
-- The string data type must be defined using a BIN2 collation

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO




/*  -- Look at our new table in an existing system view
    -- New 2014 columns are available

 select t.name as 'Table Name'
, t.schema_id
, t.object_id
, filestream_data_space_id
, is_memory_optimized
, durability
, durability_desc
   from sys.tables t
  where type='U'
    andt.schema_id = SCHEMA_ID(N'MOD');

*/


-- Look at our table via SSMS properties


-- Location of the dll files that represent the structure of the table we just created
-- C:\Program Files\Microsoft SQL Server\MSSQL12.SQL2014\MSSQL\DATA
```



DEMO - 2

B1. Create memory optimized NonDurable table – Cannot exceed a row size of 8060 bytes.

```
*** Step-5 -- NonDurable

CREATE TABLE [MOD].[NonDurable]
(
   TableID     INT NOT NULL
                                -- 1) New Inline syntax for SQL2014
   , Column1   VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
   , Column2   VARCHAR(24) NULL
   , Column3   VARCHAR(5000) NULL
   , Column4   VARCHAR(5000) NULL

   , INDEX [IX_Column2] ( [Column2] ) -- 2) Can still declare index after columns
      -- 3) NONCLUSTERED
      --      a. The "NONCLUSTERED" hint is optional,
      --              unless being defined against the primary key

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
                         --< SCHEMA_ONLY option; literally no safety net
                         -- Use cases: Staging, Website Session State
GO
```

```
Msg 41307, Level 16, State 1, Line 121
The row size limit of 8060 bytes for memory optimized tables has been exceeded. Please simplify the table definition.
```

Presented by: Miguel Cebollero @SQLMiguel

```
USE AdventureWorks2014;
GO

/*******************************************

Create new Memory Optimized Table

*******************************************/

-- Create new schema for the purposes of comparision with disk-based table
CREATE SCHEMA [MOD] AUTHORIZATION [dbo]; -- Memory Optimized Data
GO

*** Step-1 -- Create Disk-based table

-- DROP TABLE [MOD].[Durable]

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

) ON [PRIMARY] --< Note the filegroup
GO
-- No indexes, No PK, Just a Heap
```

```
*** Step-2 -- Create Memory Optimized Table

DROP TABLE [MOD].[Durable];
GO

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

--< 1. Note the lack of a filegroup
) WITH(MEMORY_OPTIMIZED=ON); -- 2. , DURABILITY=SCHEMA_AND_DATA);
-- DURABILITY option NOT mandatory
--  default is SCHEMA_AND_DATA
GO
```

```
*** Step-3 -- Missing PRIMARY KEY

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY --< Add our Primary Key
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
*** Step-4 -- PRIMARY KEY fix; NONCLUSTERED
   -- Simplest syntax for a DURABLE
   --in-memory table.
 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY NONCLUSTERED
-- InMemory tables can only have NONCLUSTERED Indexes
-- By default SQL will try to create a PK as CLUSTERED
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
```

```
GO
```

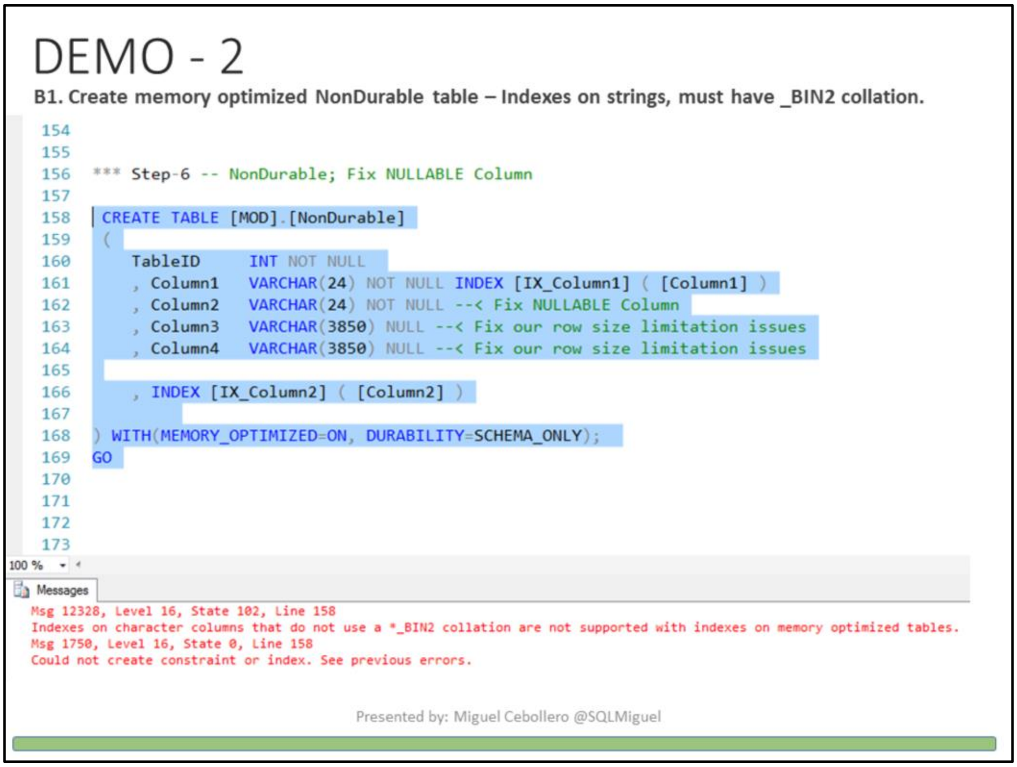
```
*** Step-5 -- NonDurable

 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- 1) New Inline syntax for SQL2014
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

, INDEX [IX_Column2] ( [Column2] ) -- 2) Can still declare index after columns
-- 3) NONCLUSTERED
--a. The "NONCLUSTERED" hint is optional,
--              unless being defined against the primary key

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
--< SCHEMA_ONLY option; literally no safety net
-- Use cases: Staging, Website Session State
GO
```

*** Step-6 -- NonDurable; Fix NULLABLE Column

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NOT NULL --< Fix NULLABLE Column
, Column3VARCHAR(3850) NULL --< Fix our row size limitation issues
, Column4VARCHAR(3850) NULL --< Fix our row size limitation issues

, INDEX [IX_Column2] ( [Column2] )

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

*** Step-7 -- NonDurable; Fix BIN2 collation
-- Simplest form of a NonDurable table

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- New BIN2 Collation
, Column1VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
INDEX [IX_Column1] ( [Column1] )
-- New BIN2 Collation only necessary if in an index
, Column2VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
, Column3VARCHAR(3850) NULL
, Column4VARCHAR(3850) NULL

, INDEX [IX_Column2] ( [Column2] )
-- The string data type must be defined using a BIN2 collation

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO




/*  -- Look at our new table in an existing system view
    -- New 2014 columns are available

 select t.name as 'Table Name'
, t.schema_id
, t.object_id
, filestream_data_space_id
, is_memory_optimized
, durability
, durability_desc
   from sys.tables t
  where type='U'
    andt.schema_id = SCHEMA_ID(N'MOD');

*/


-- Look at our table via SSMS properties


-- Location of the dll files that represent the structure of the table we just created
-- C:\Program Files\Microsoft SQL Server\MSSQL12.SQL2014\MSSQL\DATA
```



DEMO - 2

B1. Create memory optimized NonDurable table – Indexes on strings, must have _BIN2 collation.

```
*** Step-6 -- NonDurable; Fix NULLABLE Column

CREATE TABLE [MOD].[NonDurable]
(
    TableID     INT NOT NULL
    , Column1   VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
    , Column2   VARCHAR(24) NOT NULL --< Fix NULLABLE Column
    , Column3   VARCHAR(3850) NULL --< Fix our row size limitation issues
    , Column4   VARCHAR(3850) NULL --< Fix our row size limitation issues

    , INDEX [IX_Column2] ( [Column2] )

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

```
Msg 12328, Level 16, State 102, Line 158
Indexes on character columns that do not use a *_BIN2 collation are not supported with indexes on memory optimized tables.
Msg 1750, Level 16, State 0, Line 158
Could not create constraint or index. See previous errors.
```

Presented by: Miguel Cebollero @SQLMiguel

```
USE AdventureWorks2014;
GO

/*******************************************

Create new Memory Optimized Table

*******************************************/

-- Create new schema for the purposes of comparision with disk-based table
CREATE SCHEMA [MOD] AUTHORIZATION [dbo]; -- Memory Optimized Data
GO

*** Step-1 -- Create Disk-based table

-- DROP TABLE [MOD].[Durable]

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

) ON [PRIMARY] --< Note the filegroup
GO
-- No indexes, No PK, Just a Heap
```

```
*** Step-2 -- Create Memory Optimized Table

DROP TABLE [MOD].[Durable];
GO

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

--< 1. Note the lack of a filegroup
) WITH(MEMORY_OPTIMIZED=ON); -- 2. , DURABILITY=SCHEMA_AND_DATA);
-- DURABILITY option NOT mandatory
--  default is SCHEMA_AND_DATA
GO
```

```
*** Step-3 -- Missing PRIMARY KEY

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY --< Add our Primary Key
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
*** Step-4 -- PRIMARY KEY fix; NONCLUSTERED
   -- Simplest syntax for a DURABLE
   --in-memory table.
 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY NONCLUSTERED
-- InMemory tables can only have NONCLUSTERED Indexes
-- By default SQL will try to create a PK as CLUSTERED
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
```

```
GO
```

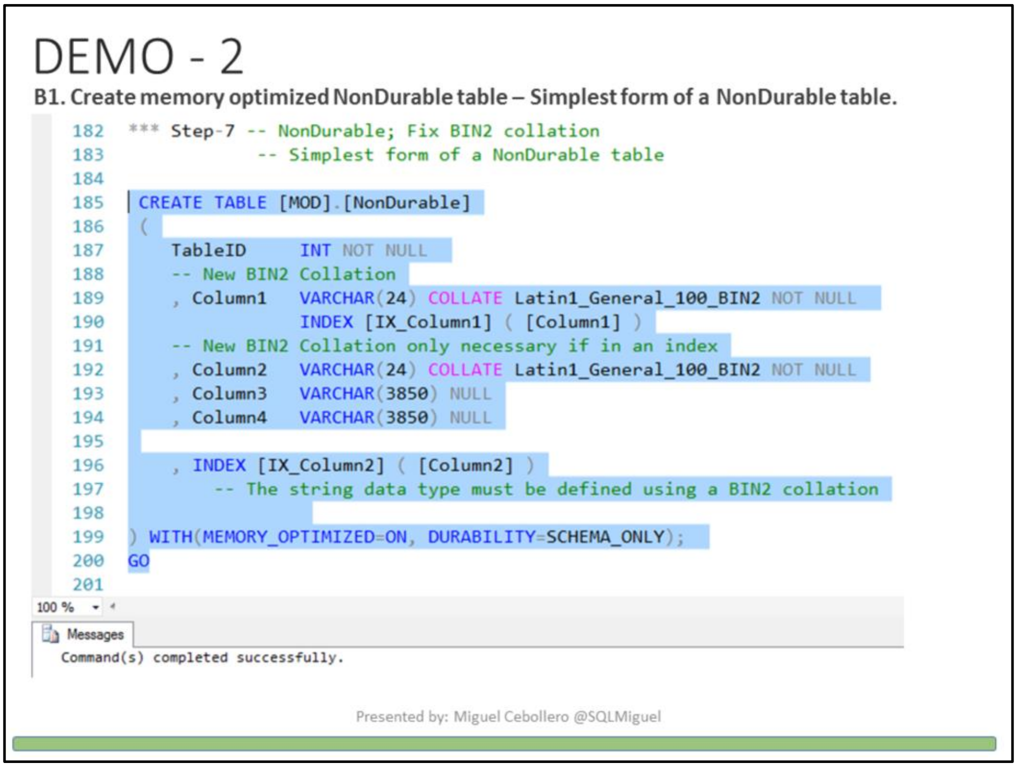
```
*** Step-5 -- NonDurable

 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- 1) New Inline syntax for SQL2014
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

, INDEX [IX_Column2] ( [Column2] ) -- 2) Can still declare index after columns
-- 3) NONCLUSTERED
--a. The "NONCLUSTERED" hint is optional,
--              unless being defined against the primary key

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
--< SCHEMA_ONLY option; literally no safety net
-- Use cases: Staging, Website Session State
GO
```

*** Step-6 -- NonDurable; Fix NULLABLE Column

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NOT NULL --< Fix NULLABLE Column
, Column3VARCHAR(3850) NULL --< Fix our row size limitation issues
, Column4VARCHAR(3850) NULL --< Fix our row size limitation issues

, INDEX [IX_Column2] ( [Column2] )

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

*** Step-7 -- NonDurable; Fix BIN2 collation
-- Simplest form of a NonDurable table

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- New BIN2 Collation
, Column1VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
INDEX [IX_Column1] ( [Column1] )
-- New BIN2 Collation only necessary if in an index
, Column2VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
, Column3VARCHAR(3850) NULL
, Column4VARCHAR(3850) NULL

, INDEX [IX_Column2] ( [Column2] )
-- The string data type must be defined using a BIN2 collation

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO




/*  -- Look at our new table in an existing system view
    -- New 2014 columns are available

 select t.name as 'Table Name'
, t.schema_id
, t.object_id
, filestream_data_space_id
, is_memory_optimized
, durability
, durability_desc
   from sys.tables t
  where type='U'
    andt.schema_id = SCHEMA_ID(N'MOD');

*/


-- Look at our table via SSMS properties


-- Location of the dll files that represent the structure of the table we just created
-- C:\Program Files\Microsoft SQL Server\MSSQL12.SQL2014\MSSQL\DATA
```



DEMO - 2

B1. Create memory optimized NonDurable table – Simplest form of a NonDurable table.

```
*** Step-7 -- NonDurable; Fix BIN2 collation
            -- Simplest form of a NonDurable table

CREATE TABLE [MOD].[NonDurable]
(
   TableID     INT NOT NULL
   -- New BIN2 Collation
   , Column1   VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
               INDEX [IX_Column1] ( [Column1] )
   -- New BIN2 Collation only necessary if in an index
   , Column2   VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
   , Column3   VARCHAR(3850) NULL
   , Column4   VARCHAR(3850) NULL

   , INDEX [IX_Column2] ( [Column2] )
       -- The string data type must be defined using a BIN2 collation

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO

```

```
Command(s) completed successfully.
```

Presented by: Miguel Cebollero @SQLMiguel

USE AdventureWorks2014;
GO

/**

Create new Memory Optimized Table

**/

-- Create new schema for the purposes of comparision with disk-based table
CREATE SCHEMA [MOD] AUTHORIZATION [dbo]; -- Memory Optimized Data
GO

*** Step-1 -- Create Disk-based table

-- DROP TABLE [MOD].[Durable]

CREATE TABLE [MOD].[Durable]
(
TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL

```
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

) ON [PRIMARY] --< Note the filegroup
GO
-- No indexes, No PK, Just a Heap
```

```
*** Step-2 -- Create Memory Optimized Table

DROP TABLE [MOD].[Durable];
GO

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

--< 1. Note the lack of a filegroup
) WITH(MEMORY_OPTIMIZED=ON); -- 2. , DURABILITY=SCHEMA_AND_DATA);
-- DURABILITY option NOT mandatory
--  default is SCHEMA_AND_DATA
GO
```

```
*** Step-3 -- Missing PRIMARY KEY

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY --< Add our Primary Key
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
*** Step-4 -- PRIMARY KEY fix; NONCLUSTERED
   -- Simplest syntax for a DURABLE
   --in-memory table.
 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY NONCLUSTERED
-- InMemory tables can only have NONCLUSTERED Indexes
-- By default SQL will try to create a PK as CLUSTERED
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
```

```
GO
```

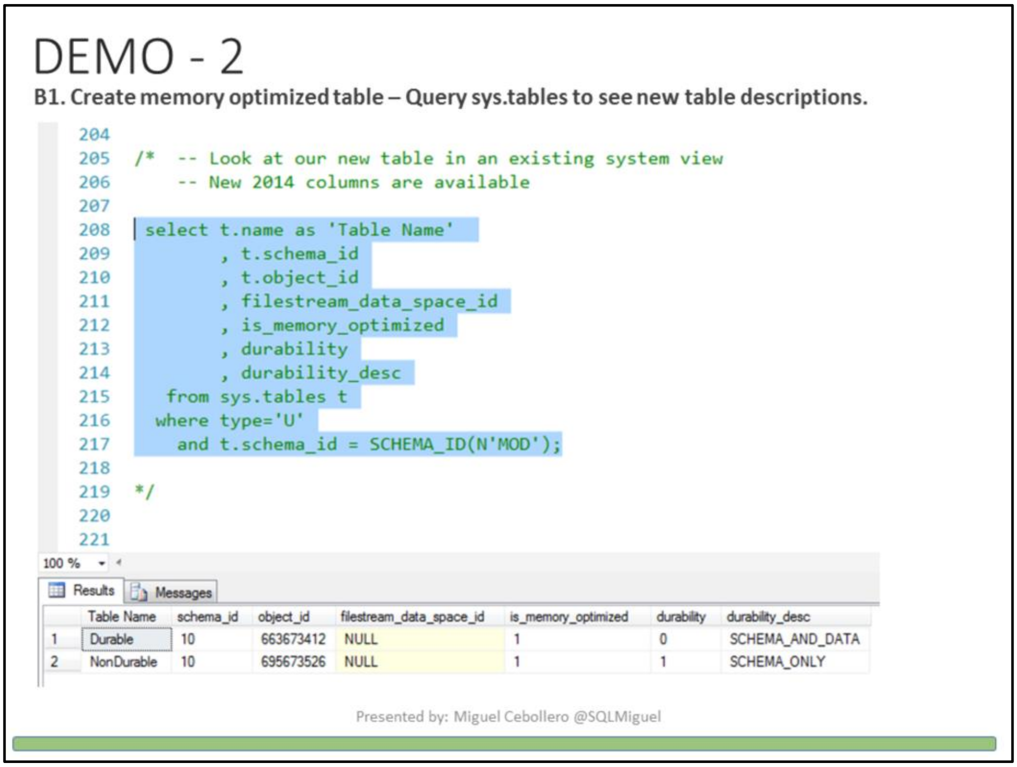
```
*** Step-5 -- NonDurable

 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- 1) New Inline syntax for SQL2014
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

, INDEX [IX_Column2] ( [Column2] ) -- 2) Can still declare index after columns
-- 3) NONCLUSTERED
--a. The "NONCLUSTERED" hint is optional,
--              unless being defined against the primary key

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
--< SCHEMA_ONLY option; literally no safety net
-- Use cases: Staging, Website Session State
GO
```

*** Step-6 -- NonDurable; Fix NULLABLE Column

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NOT NULL --< Fix NULLABLE Column
, Column3VARCHAR(3850) NULL --< Fix our row size limitation issues
, Column4VARCHAR(3850) NULL --< Fix our row size limitation issues

, INDEX [IX_Column2] ( [Column2] )

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

*** Step-7 -- NonDurable; Fix BIN2 collation
-- Simplest form of a NonDurable table

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- New BIN2 Collation
, Column1VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
INDEX [IX_Column1] ( [Column1] )
-- New BIN2 Collation only necessary if in an index
, Column2VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
, Column3VARCHAR(3850) NULL
, Column4VARCHAR(3850) NULL

, INDEX [IX_Column2] ( [Column2] )
-- The string data type must be defined using a BIN2 collation

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO




/*  -- Look at our new table in an existing system view
    -- New 2014 columns are available

 select t.name as 'Table Name'
, t.schema_id
, t.object_id
, filestream_data_space_id
, is_memory_optimized
, durability
, durability_desc
   from sys.tables t
  where type='U'
    andt.schema_id = SCHEMA_ID(N'MOD');

*/


-- Look at our table via SSMS properties


-- Location of the dll files that represent the structure of the table we just created
-- C:\Program Files\Microsoft SQL Server\MSSQL12.SQL2014\MSSQL\DATA
```



DEMO - 2

B1. Create memory optimized table – Query sys.tables to see new table descriptions.

```
/*  -- Look at our new table in an existing system view
    -- New 2014 columns are available

select t.name as 'Table Name'
        , t.schema_id
        , t.object_id
        , filestream_data_space_id
        , is_memory_optimized
        , durability
        , durability_desc
   from sys.tables t
  where type='U'
    and t.schema_id = SCHEMA_ID(N'MOD');

*/
```

	Table Name	schema_id	object_id	filestream_data_space_id	is_memory_optimized	durability	durability_desc
1	Durable	10	663673412	NULL	1	0	SCHEMA_AND_DATA
2	NonDurable	10	695673526	NULL	1	1	SCHEMA_ONLY

Presented by: Miguel Cebollero @SQLMiguel

```
USE AdventureWorks2014;
GO

/*******************************************

Create new Memory Optimized Table

*******************************************/

-- Create new schema for the purposes of comparision with disk-based table
CREATE SCHEMA [MOD] AUTHORIZATION [dbo]; -- Memory Optimized Data
GO

*** Step-1 -- Create Disk-based table

-- DROP TABLE [MOD].[Durable]

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

) ON [PRIMARY] --< Note the filegroup
GO
-- No indexes, No PK, Just a Heap
```

```
*** Step-2 -- Create Memory Optimized Table

DROP TABLE [MOD].[Durable];
GO

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

--< 1. Note the lack of a filegroup
) WITH(MEMORY_OPTIMIZED=ON); -- 2. , DURABILITY=SCHEMA_AND_DATA);
-- DURABILITY option NOT mandatory
--  default is SCHEMA_AND_DATA
GO
```

```
*** Step-3 -- Missing PRIMARY KEY

 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY --< Add our Primary Key
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
GO
```

```
*** Step-4 -- PRIMARY KEY fix; NONCLUSTERED
   -- Simplest syntax for a DURABLE
   --in-memory table.
 CREATE TABLE [MOD].[Durable]
 (
 TableIDINT NOT NULL PRIMARY KEY NONCLUSTERED
-- InMemory tables can only have NONCLUSTERED Indexes
-- By default SQL will try to create a PK as CLUSTERED
, Column1VARCHAR(24) NOT NULL
, Column2VARCHAR(24) NULL

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_AND_DATA);
```

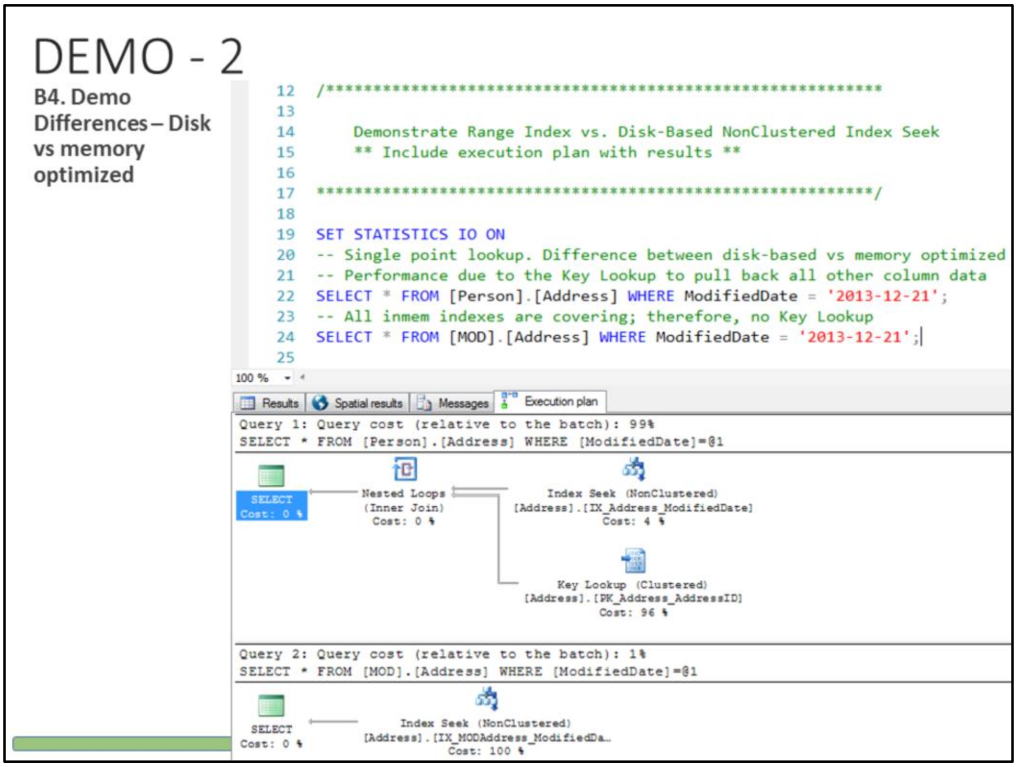
```
GO
```

```
*** Step-5 -- NonDurable

 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- 1) New Inline syntax for SQL2014
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NULL
, Column3VARCHAR(4000) NULL
, Column4VARCHAR(4000) NULL

, INDEX [IX_Column2] ( [Column2] ) -- 2) Can still declare index after columns
-- 3) NONCLUSTERED
--a. The "NONCLUSTERED" hint is optional,
--              unless being defined against the primary key

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
--< SCHEMA_ONLY option; literally no safety net
-- Use cases: Staging, Website Session State
GO
```

```
*** Step-6 -- NonDurable; Fix NULLABLE Column

 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
, Column1VARCHAR(24) NOT NULL INDEX [IX_Column1] ( [Column1] )
, Column2VARCHAR(24) NOT NULL --< Fix NULLABLE Column
, Column3VARCHAR(3850) NULL --< Fix our row size limitation issues
, Column4VARCHAR(3850) NULL --< Fix our row size limitation issues

, INDEX [IX_Column2] ( [Column2] )

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO
```

```
*** Step-7 -- NonDurable; Fix BIN2 collation
-- Simplest form of a NonDurable table
```

```
 CREATE TABLE [MOD].[NonDurable]
 (
 TableIDINT NOT NULL
-- New BIN2 Collation
, Column1VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
INDEX [IX_Column1] ( [Column1] )
-- New BIN2 Collation only necessary if in an index
, Column2VARCHAR(24) COLLATE Latin1_General_100_BIN2 NOT NULL
, Column3VARCHAR(3850) NULL
, Column4VARCHAR(3850) NULL

, INDEX [IX_Column2] ( [Column2] )
-- The string data type must be defined using a BIN2 collation

) WITH(MEMORY_OPTIMIZED=ON, DURABILITY=SCHEMA_ONLY);
GO




/*  -- Look at our new table in an existing system view
    -- New 2014 columns are available

 select t.name as 'Table Name'
, t.schema_id
, t.object_id
, filestream_data_space_id
, is_memory_optimized
, durability
, durability_desc
   from sys.tables t
  where type='U'
    andt.schema_id = SCHEMA_ID(N'MOD');

*/


-- Look at our table via SSMS properties


-- Location of the dll files that represent the structure of the table we just created
-- C:\Program Files\Microsoft SQL Server\MSSQL12.SQL2014\MSSQL\DATA
```



DEMO - 2

B4. Demo Differences – Disk vs memory optimized

```
/**************************************************************

   Demonstrate Range Index vs. Disk-Based NonClustered Index Seek
   ** Include execution plan with results **

**************************************************************/

SET STATISTICS IO ON
-- Single point lookup. Difference between disk-based vs memory optimized
-- Performance due to the Key Lookup to pull back all other column data
SELECT * FROM [Person].[Address] WHERE ModifiedDate = '2013-12-21';
-- All inmem indexes are covering; therefore, no Key Lookup
SELECT * FROM [MOD].[Address] WHERE ModifiedDate = '2013-12-21';
```

```
Query 1: Query cost (relative to the batch): 99%
SELECT * FROM [Person].[Address] WHERE [ModifiedDate]=@1
```

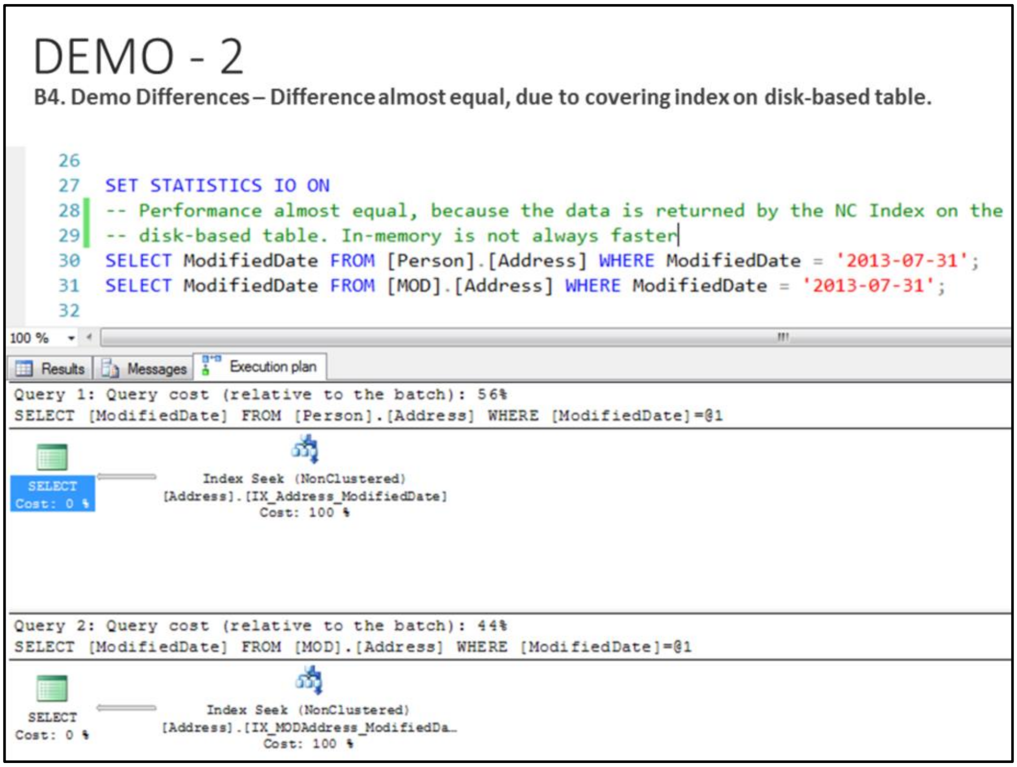
SELECT
Cost: 0 %
Nested Loops
(Inner Join)
Cost: 0 %
Index Seek (NonClustered)
[Address].[IX_Address_ModifiedDate]
Cost: 4 %
Key Lookup (Clustered)
[Address].[PK_Address_AddressID]
Cost: 96 %

```
Query 2: Query cost (relative to the batch): 1%
SELECT * FROM [MOD].[Address] WHERE [ModifiedDate]=@1
```

SELECT
Cost: 0 %
Index Seek (NonClustered)
[Address].[IX_MODAddress_ModifiedDa...
Cost: 100 %

```
USE AdventureWorks2014;
GO

-- Single point lookup using a Range Index
CHECKPOINT
GO
DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

/**********************************************************

Demonstrate Range Index vs. Disk-Based NonClustered Index Seek
** Include execution plan with results **

**********************************************************/

SET STATISTICS IO ON
-- Single point lookup. Difference between disk-based vs memory optimized
-- Performance due to the Key Lookup to pull back all other column data
```

```
SELECT * FROM [Person].[Address] WHERE ModifiedDate = '2013-12-21';
-- All inmem indexes are covering; therefore, no Key Lookup
SELECT * FROM [MOD].[Address] WHERE ModifiedDate = '2013-12-21';


SET STATISTICS IO ON
-- Performance almost equal, because the data is returned by the NC Index
SELECT ModifiedDate FROM [Person].[Address] WHERE ModifiedDate = '2013-07-31';
SELECT ModifiedDate FROM [MOD].[Address] WHERE ModifiedDate = '2013-07-31';

-- Perform a range query
SELECT * FROM [Person].[Address] WHERE ModifiedDate BETWEEN '2013-12-01'
AND '2013-12-21';
-- As expected the memory optimized table performs better
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21';

-- NonClustered Index in ASC order for our MOD table.
--, ModifiedDateDATETIME NOT NULL INDEX [IX_MODAddress_ModifiedDate]
NONCLUSTERED

-- Performs Index Seek
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21' ORDER BY ModifiedDate;
-- Performs Index Seek, but must also sort the data; cannot return the data in
opposite order of the index.
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21' ORDER BY ModifiedDate DESC; -- ASC;
```



DEMO - 2

B4. Demo Differences – Difference almost equal, due to covering index on disk-based table.

```
SET STATISTICS IO ON
-- Performance almost equal, because the data is returned by the NC Index on the
-- disk-based table. In-memory is not always faster
SELECT ModifiedDate FROM [Person].[Address] WHERE ModifiedDate = '2013-07-31';
SELECT ModifiedDate FROM [MOD].[Address] WHERE ModifiedDate = '2013-07-31';
```

```
Query 1: Query cost (relative to the batch): 56%
SELECT [ModifiedDate] FROM [Person].[Address] WHERE [ModifiedDate]=@1
```

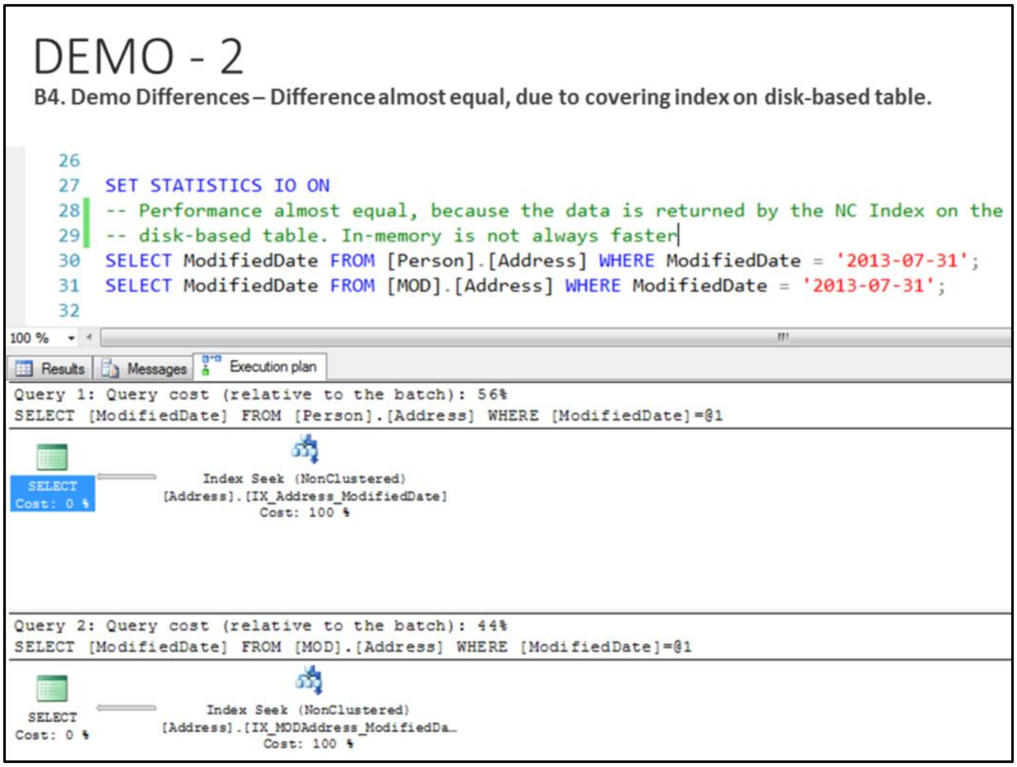
SELECT
Cost: 0 %
Index Seek (NonClustered)
[Address].[IX_Address_ModifiedDate]
Cost: 100 %

```
Query 2: Query cost (relative to the batch): 44%
SELECT [ModifiedDate] FROM [MOD].[Address] WHERE [ModifiedDate]=@1
```

SELECT
Cost: 0 %
Index Seek (NonClustered)
[Address].[IX_MODAddress_ModifiedDa…
Cost: 100 %

```
USE AdventureWorks2014;
GO

-- Single point lookup using a Range Index
CHECKPOINT
GO
DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

/*****************************************************************

Demonstrate Range Index vs. Disk-Based NonClustered Index Seek
** Include execution plan with results **

*****************************************************************/

SET STATISTICS IO ON
-- Single point lookup. Difference between disk-based vs memory optimized
-- Performance due to the Key Lookup to pull back all other column data
```

```
SELECT * FROM [Person].[Address] WHERE ModifiedDate = '2013-12-21';
-- All inmem indexes are covering; therefore, no Key Lookup
SELECT * FROM [MOD].[Address] WHERE ModifiedDate = '2013-12-21';


SET STATISTICS IO ON
-- Performance almost equal, because the data is returned by the NC Index
SELECT ModifiedDate FROM [Person].[Address] WHERE ModifiedDate = '2013-07-31';
SELECT ModifiedDate FROM [MOD].[Address] WHERE ModifiedDate = '2013-07-31';

-- Perform a range query
SELECT * FROM [Person].[Address] WHERE ModifiedDate BETWEEN '2013-12-01'
AND '2013-12-21';
-- As expected the memory optimized table performs better
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21';

-- NonClustered Index in ASC order for our MOD table.
--, ModifiedDateDATETIME NOT NULL INDEX [IX_MODAddress_ModifiedDate]
NONCLUSTERED

-- Performs Index Seek
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21' ORDER BY ModifiedDate;
-- Performs Index Seek, but must also sort the data; cannot return the data in
opposite order of the index.
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21' ORDER BY ModifiedDate DESC; -- ASC;
```



DEMO - 2

B4. Demo Differences – Difference almost equal, due to covering index on disk-based table.

```
SET STATISTICS IO ON
-- Performance almost equal, because the data is returned by the NC Index on the
-- disk-based table. In-memory is not always faster
SELECT ModifiedDate FROM [Person].[Address] WHERE ModifiedDate = '2013-07-31';
SELECT ModifiedDate FROM [MOD].[Address] WHERE ModifiedDate = '2013-07-31';
```

```
Query 1: Query cost (relative to the batch): 56%
SELECT [ModifiedDate] FROM [Person].[Address] WHERE [ModifiedDate]=@1
```

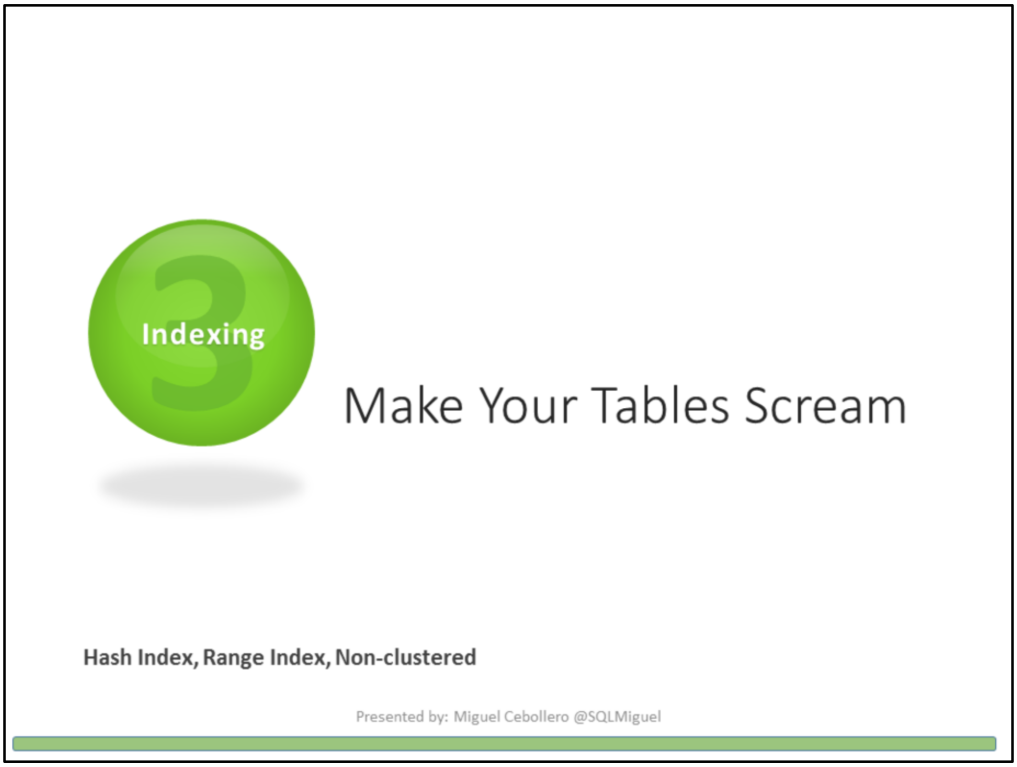
SELECT
Cost: 0 %
Index Seek (NonClustered)
[Address].[IX_Address_ModifiedDate]
Cost: 100 %

```
Query 2: Query cost (relative to the batch): 44%
SELECT [ModifiedDate] FROM [MOD].[Address] WHERE [ModifiedDate]=@1
```

SELECT
Cost: 0 %
Index Seek (NonClustered)
[Address].[IX_MODAddress_ModifiedDa…
Cost: 100 %

```
USE AdventureWorks2014;
GO

-- Single point lookup using a Range Index
CHECKPOINT
GO
DBCC DROPCLEANBUFFERS
GO
DBCC FREEPROCCACHE
GO

/********************************************************

Demonstrate Range Index vs. Disk-Based NonClustered Index Seek
** Include execution plan with results **

********************************************************/

SET STATISTICS IO ON
-- Single point lookup. Difference between disk-based vs memory optimized
-- Performance due to the Key Lookup to pull back all other column data
```

```
SELECT * FROM [Person].[Address] WHERE ModifiedDate = '2013-12-21';
-- All inmem indexes are covering; therefore, no Key Lookup
SELECT * FROM [MOD].[Address] WHERE ModifiedDate = '2013-12-21';


SET STATISTICS IO ON
-- Performance almost equal, because the data is returned by the NC Index
SELECT ModifiedDate FROM [Person].[Address] WHERE ModifiedDate = '2013-07-31';
SELECT ModifiedDate FROM [MOD].[Address] WHERE ModifiedDate = '2013-07-31';

-- Perform a range query
SELECT * FROM [Person].[Address] WHERE ModifiedDate BETWEEN '2013-12-01'
AND '2013-12-21';
-- As expected the memory optimized table performs better
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21';

-- NonClustered Index in ASC order for our MOD table.
--, ModifiedDateDATETIME NOT NULL INDEX [IX_MODAddress_ModifiedDate]
NONCLUSTERED

-- Performs Index Seek
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21' ORDER BY ModifiedDate;
-- Performs Index Seek, but must also sort the data; cannot return the data in
opposite order of the index.
SELECT * FROM [MOD].[Address] WHERE ModifiedDate BETWEEN '2013-12-01' AND
'2013-12-21' ORDER BY ModifiedDate DESC; -- ASC;
```



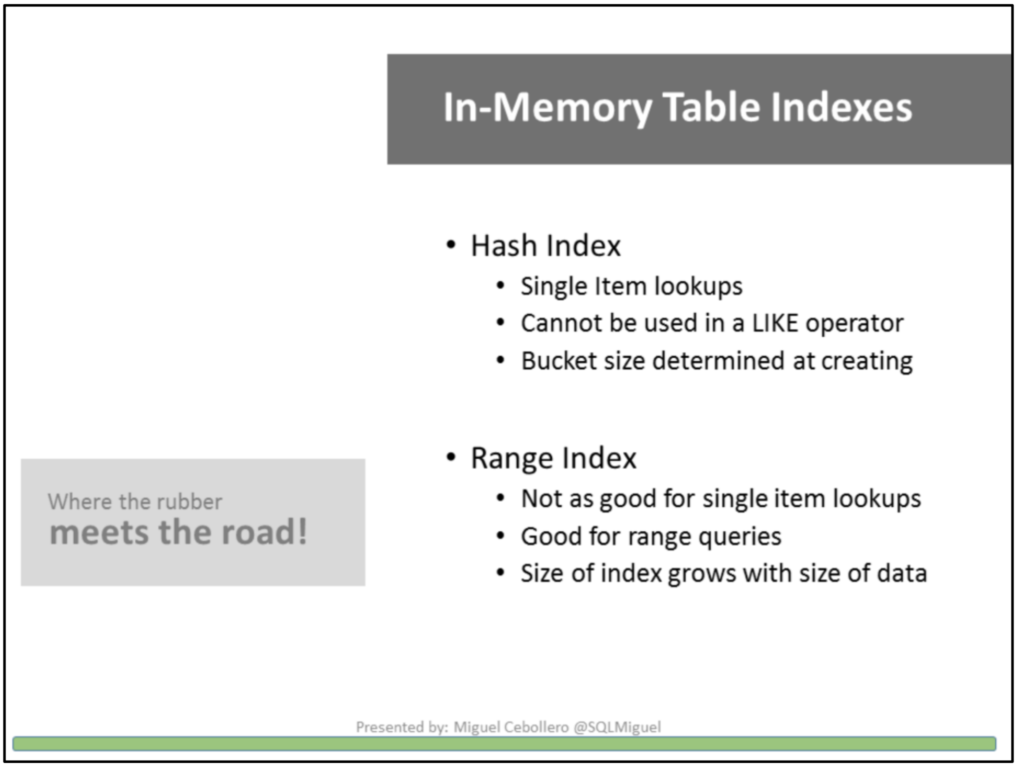
3

Indexing

Make Your Tables Scream

Hash Index, Range Index, Non-clustered

Presented by: Miguel Cebollero @SQLMiguel

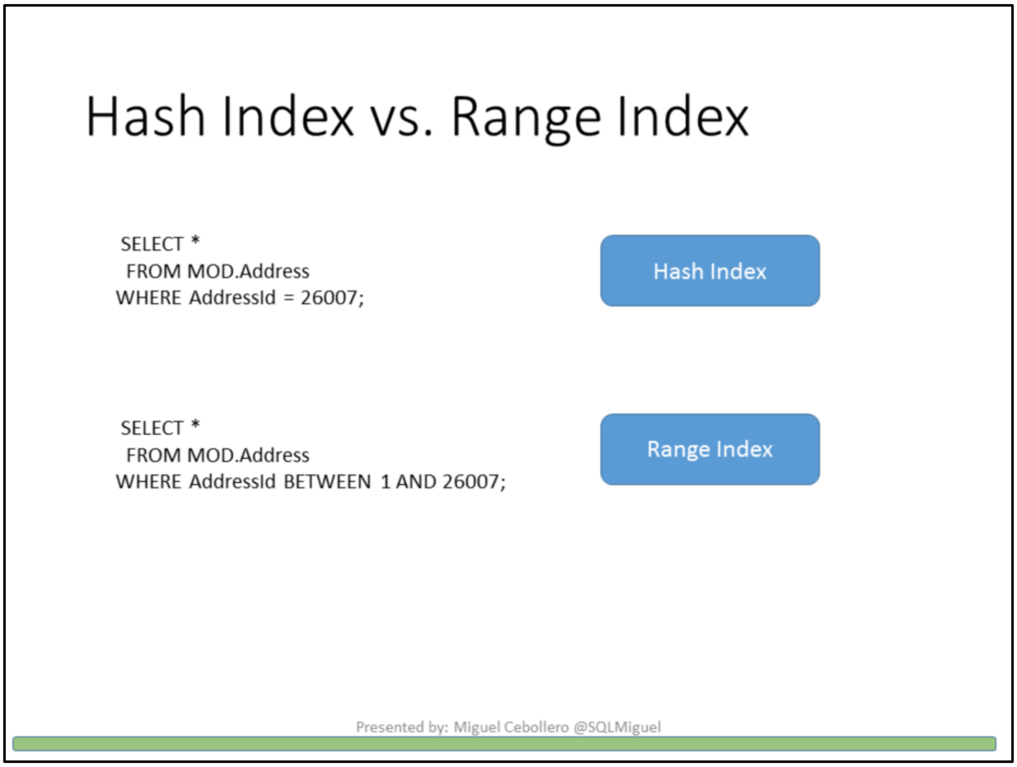


In-Memory Table Indexes

- Hash Index
  - Single Item lookups
  - Cannot be used in a LIKE operator
  - Bucket size determined at creating
- Range Index
  - Not as good for single item lookups
  - Good for range queries
  - Size of index grows with size of data

Where the rubber
meets the road!

Presented by: Miguel Cebollero @SQLMiguel



Hash Index vs. Range Index

```
SELECT *
 FROM MOD.Address
WHERE AddressId = 26007;
```

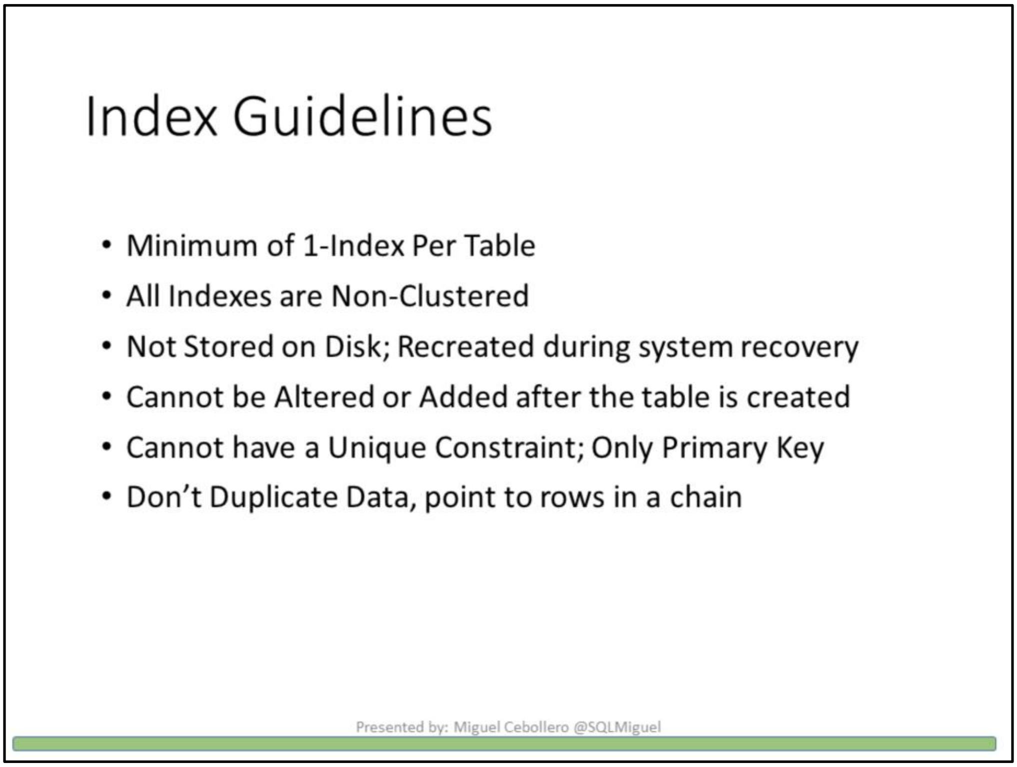
Hash Index

```
SELECT *
 FROM MOD.Address
WHERE AddressId BETWEEN 1 AND 26007;
```

Range Index

Presented by: Miguel Cebollero @SQLMiguel

- This offers 10x or more performance gains. More gains for more complicated scenarios.
- If you need to access a mix between disk-based and memory optimized tables, you need to use traditional interpreted stored procedures.
-

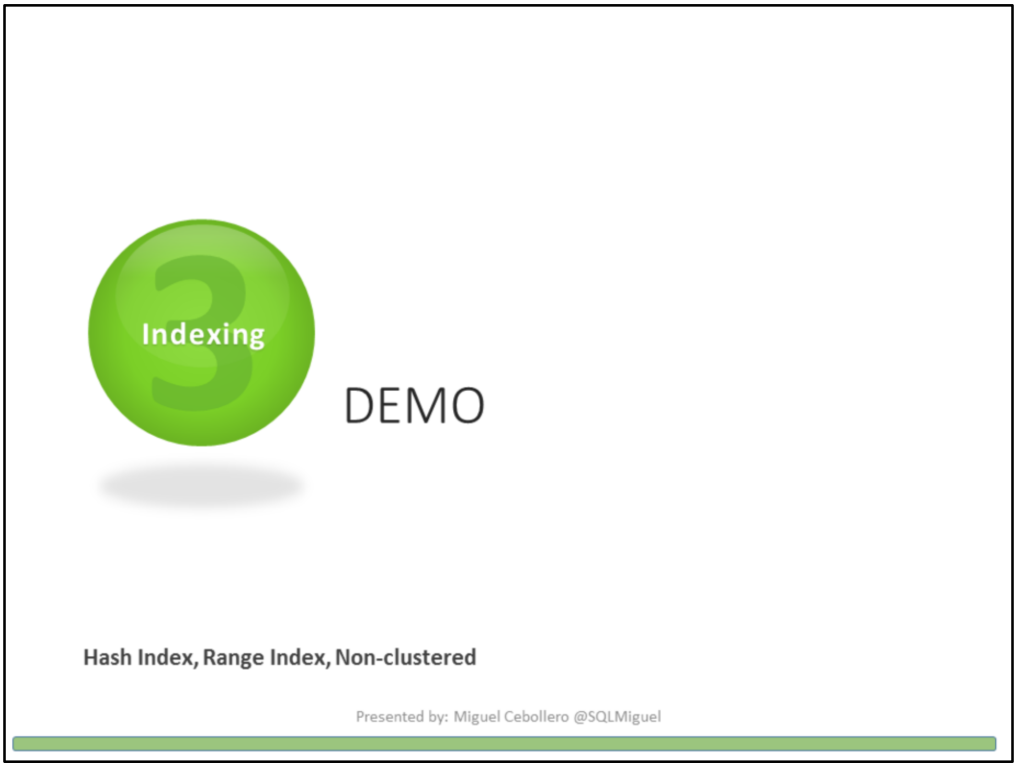


Index Guidelines

- Minimum of 1-Index Per Table
- All Indexes are Non-Clustered
- Not Stored on Disk; Recreated during system recovery
- Cannot be Altered or Added after the table is created
- Cannot have a Unique Constraint; Only Primary Key
- Don’t Duplicate Data, point to rows in a chain

Presented by: Miguel Cebollero @SQLMiguel

- This offers 10x or more performance gains. More gains for more complicated scenarios.
- If you need to access a mix between disk-based and memory optimized tables, you need to use traditional interpreted stored procedures.
-



Indexing

DEMO

Hash Index, Range Index, Non-clustered

Presented by: Miguel Cebollero @SQLMiguel

References / Links

- http://www.BIDataPartners.com

SQL2014 In-Memory

Presented by: Miguel Cebollero @SQLMiguel