SQL07 - SQL Server 2014 In-Memory Query Engine e MVCC

A Gentle Introduction…

Davide Mauri

dmauri@solidq.com - @mauridb

http://www.davidemauri.it

Grazie a

Sponsor

Davide Mauri

18 Years of experience on the SQL Server Platform

Specialized in Data Solution Architecture, Database Design, Performance Tuning, High-Performance DWH, Business Intelligence

Microsoft SQL Server MVP 

President of UGISS (Italian SQL Server UG)

Mentor @ SolidQ

Regular Speaker @ SQL Server events

Book, Articles & Video Author

Projects, Consulting, Mentoring & Training

Agenda

- In-Memory Databases?
- SQL Server In-Memory solution
  - OLTP
  - OLAP
- Hekaton
  - Overview Architetturale
  - Hekaton Tables & Indexes
  - Multiversion Concurrency Control
  - Native Compiled Stored Procedures
- Note per gli sviluppatori

In-Memory Databases?

- Negli ultimi anni è tornato di moda il concetto (IMDB)

- Tutti i più grandi vendor propongono una soluzione
  - Oracle TimesTen
  - IBM SolidDB
  - SAP HANA
  - Microsoft
    - Hekaton (XTP – eXtreme Transaction Processing)
    - xVelocity (VertiPaq)

- In realtà tutti gli RDBMS sono già in-memory (buffer cache)…quindi cosa c'è di *realmente* nuovo?

In-Memory Databases?

- La novità sono le strutture dati!

- Sono ottimizzate per essere usate in memoria (e SOLO in memoria)

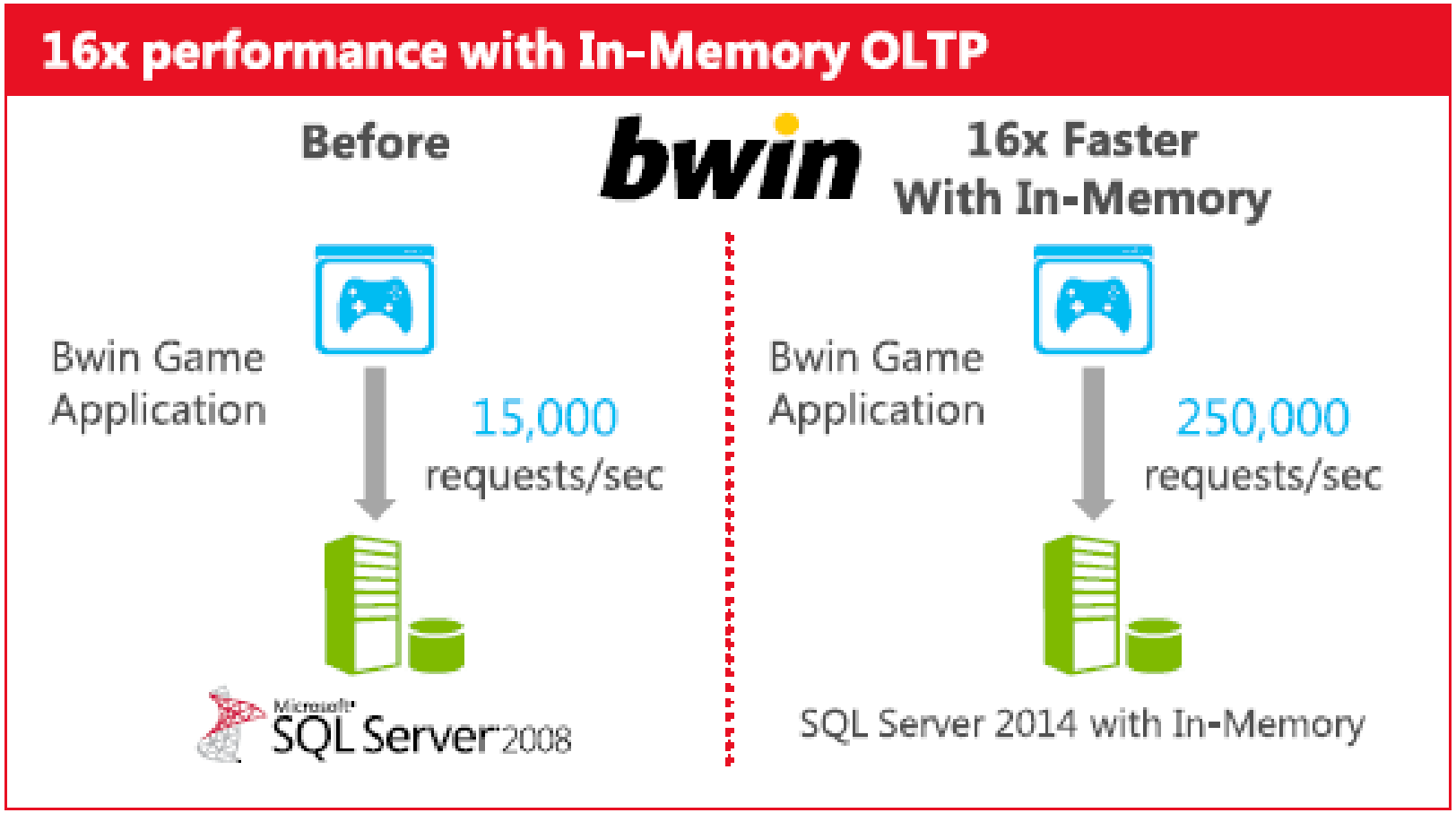
- Microsoft Research «Main Memory Databases»
  - http://research.microsoft.com/en-us/projects/main-memory_dbs/
  - In fase di studio e ricerca dal 2009
  - Perché le CPU non aumentano più di velocità…

SQL Server In-Memory solution

- Hekaton
  - In-Memory Engine per OLTP
  - Disponibile dalla versione 2014

- xVelocity
  - «In-Memory Engine» per OLAP
    - Di fatto è un column-oriented storage, con alcune ottimizzazioni a livello di engine (Batch Processing)
  - Disponibile sia per SQL Server che per Analysis Services (con alcune differenze)
  - SQL Server
    - ColumnStore Indexes (dalla versione 2012. Con 2014: Clustered ed Updatable)
  - Analysis Services
    - PowerPivot (Client, dalla versione 2008R2)
    - Tabular (Server, dalla versione 2012)

SQL Server In-Memory solution



Hekaton

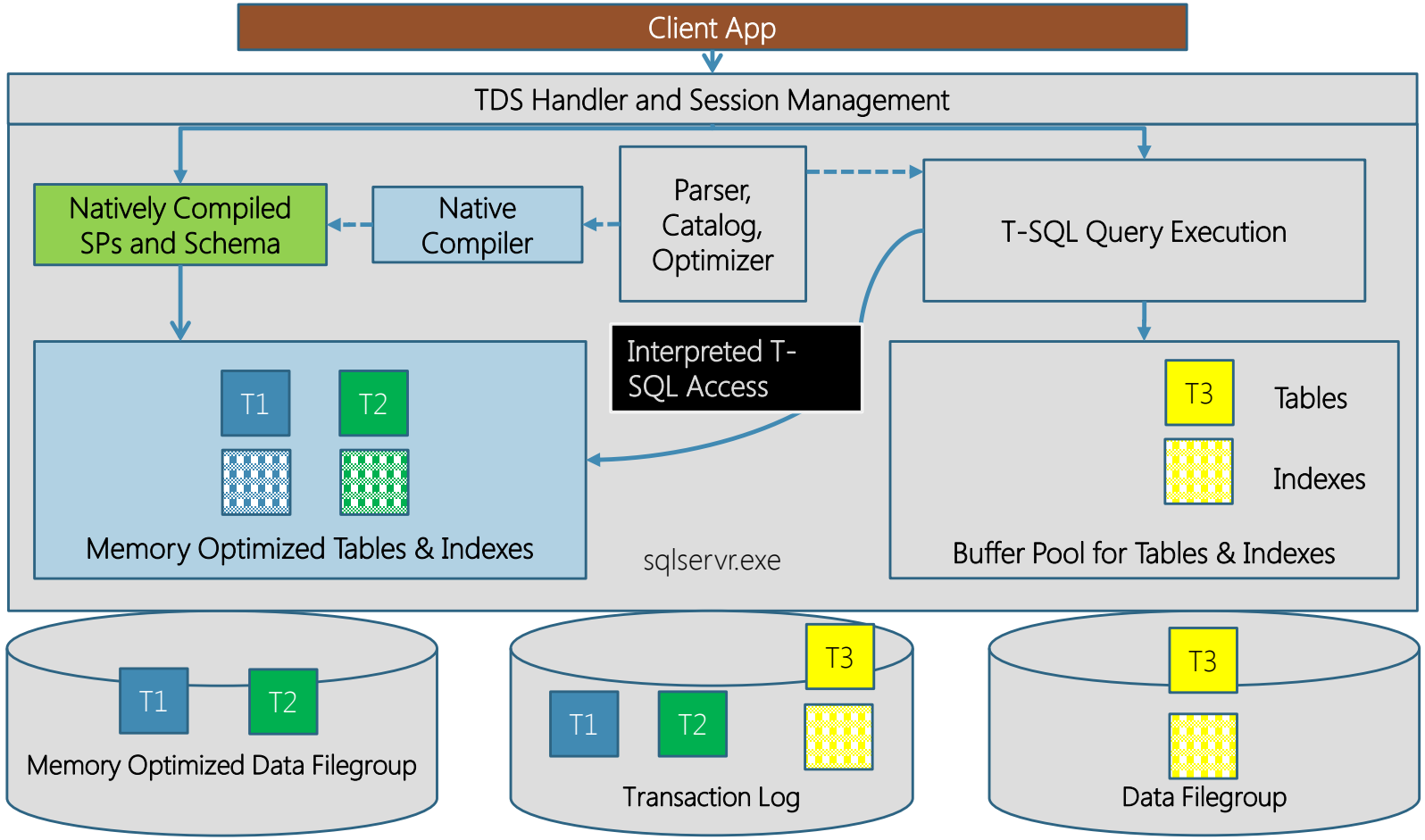
- L'engine di SQL Server è stato esteso per supportare una nuova struttura dati dove:
  - Viene abbandonata l'architettura pagine/extent
  - Vengono abbandonati gli indici basati su B+Tree

- Nuova struttura basata sul FILESTREAM

- Impostazione utilizzo al momento della creazione della tabella
  - Attivazione supporto a livello di database

- Trasparente per l'applicazione che ne fa uso

Hekaton – Engine Architeture



Existing SQL Component

In-mem OLTP Component

Generate .dll

Hekaton – Storage Architeture

- Coppia di file
  - Data File
  - Delta File

- Data files
  - ~128MB, scritti blocchi di 256KB
  - Memorizza solo le righe INSERITE
  - E' di fatto uno stream di righe inserite in ordine cronologico
- Delta files
  - Scritti a blocchi di 4KB
  - Memorizza gli ID delle righe cancellate (Update = INSERT & DELETE)

demo

Hekaton

Hekaton

- I dati di una tabella Hekaton sono SEMPRE in memoria
  - Caricati al momento dell'avvio di SQL Server
- La persistenza dei dati è garantita dall'utilizzo del transaction log
  - Ma è una cosa opzionale ☺
- SCHEMA_AND_DATA
  - Tabella persistente
- SCHEMA_ONLY
  - Persistente lo schema della tabella ma non i dati
  - Ogni operazione è comunque transazionale e supporta ACID al 100% (fintanto che l'istanza di SQL Server è attiva

Hekaton

- La tabelle Hekaton DEVONO avere almeno un indice
  - L'indice è quello che «tiene insieme» i dati
- Ci possono essere un massimo di 8 indici sulla tabella
- Ci sono due tipi di indici supportati
  - HASH
  - BW-Tree (Buzzword Tree ☺)
- Gli indici non sono scritti su disco ma vengono ricreati ogni volta

Hekaton

- Le tabelle Hekaton supportano un sottoinsieme delle feature delle tabelle disk-based

- Tipi di dati supportati
  - bit
  - tinyint, smallint, int, bigint
  - money, smallmoney
  - float, real
  - datetime, smalldatetime, datetime2, date, time
  - numeric and decimal types
  - NON LOB: char(n), varchar(n), nchar(n), nvarchar(n), sysname
  - NON LOB: binary(n), varbinary(n)
  - Uniqueidentifier

Hekaton

- La dimensione di una riga di una tabella Hekaton è al massimo 8060 bytes
  - Check fatto al momento della creazione della tabella
  - E non dell'inserimento della riga come le tabelle disk-based
- Non c'è supporto per l'overflow dei dati
- Quindi non è possibile creare una tabella con due varchar(5000)

Hekaton

- Ci sono altre limitazioni nelle tabelle Hekaton da tenere presente
  - No DML triggers
  - No FOREIGN KEY o CHECK constraints
  - No IDENTITY columns
  - No UNIQUE indexes (a parte la PRIMARY KEY)
- La tabella non può essere modificata una volta creata
- Gli indici DEVONO essere specificati al momento della creazione della tabella

demo

Hekaton

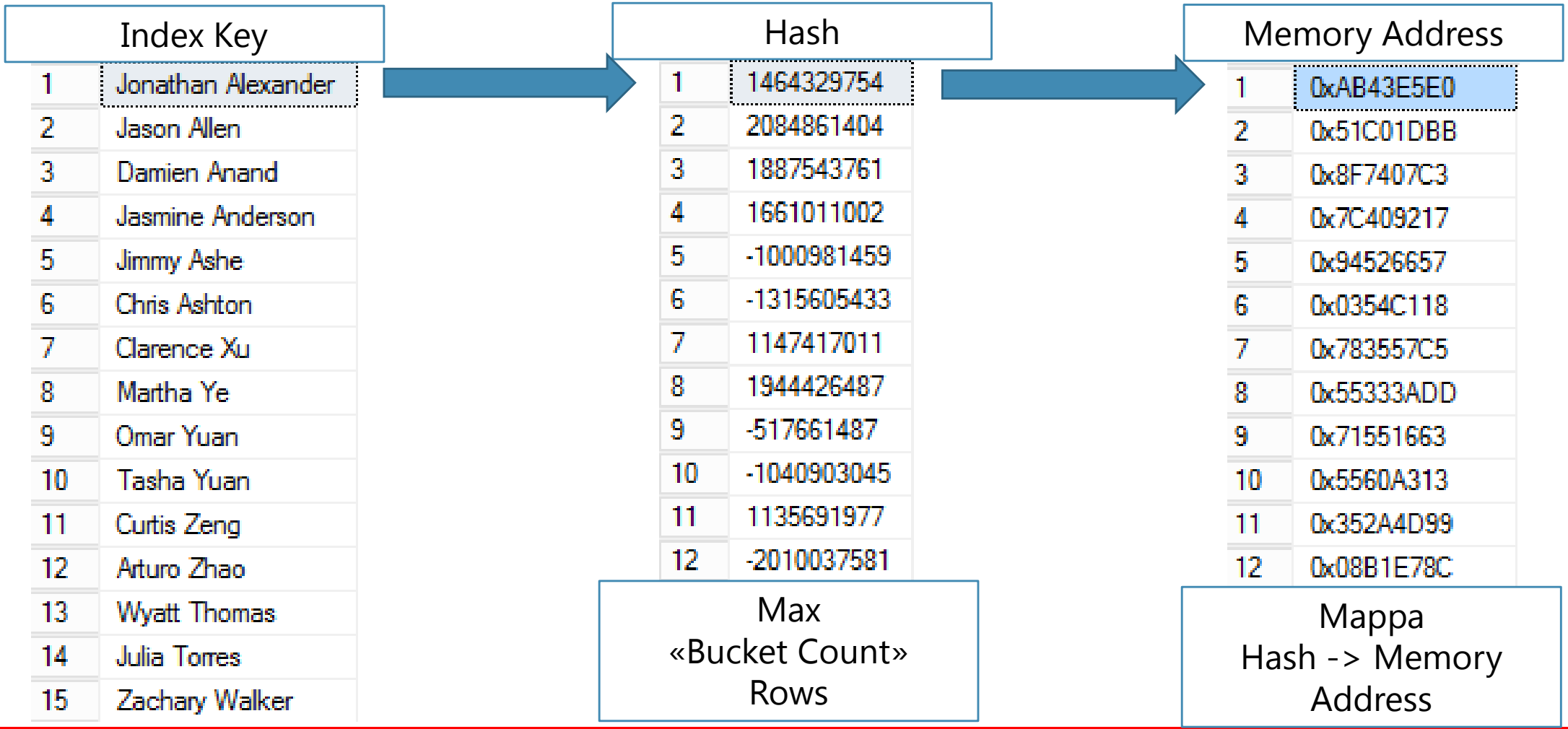
Hekaton

- Supporto per due tipi di indici
  - HASH: per le ricerche con operare di uguaglianza
  - BW-TREE: per tutto il resto

- Hash Index
  - Di base è un array di puntatori

- BW-Tree
  - Evoluzione del B-Tree
  - Le pagine non vengono MAI cancellate (version chain + delta process)

Hash Index


Index Key
1 Jonathan Alexander
2 Jason Allen
3 Damien Anand
4 Jasmine Anderson
5 Jimmy Ashe
6 Chris Ashton
7 Clarence Xu
8 Martha Ye
9 Omar Yuan
10 Tasha Yuan
11 Curtis Zeng
12 Arturo Zhao
13 Wyatt Thomas
14 Julia Torres
15 Zachary Walker
Hash
1 1464329754
2 2084861404
3 1887543761
4 1661011002
5 -1000981459
6 -1315605433
7 1147417011
8 1944426487
9 -517661487
10 -1040903045
11 1135691977
12 -2010037581
Max «Bucket Count» Rows
Memory Address
1 0xAB43E5E0
2 0x51C01DBB
3 0x8F7407C3
4 0x7C409217
5 0x94526657
6 0x0354C118
7 0x783557C5
8 0x55333ADD
9 0x71551663
10 0x5560A313
11 0x352A4D99
12 0x08B1E78C
Mappa Hash -> Memory Address

Hash Index

	Customer	PersonType	NameStyle	Title	EmailPromotion	MemoryAddress
1	Ramon Wu	IN	0	Mr.	2	0x47B9C500
2	Ramon Lin	IN	0	Mr.	1	0x7B9C4AAA
3	Ramon Zhou	IN	0	Mr.	0	0xB9C5CBC2
4	Jonathan Washington	IN	0	Mr.	2	0x3C1F8B10
5	Jonathan Butler	IN	0	Mr.	1	0x88CA48C5
6	Ramon Ye	IN	0	Mr.	2	0x47B9C5F0
7	Jonathan Foster	IN	0	Mr.	0	0x889020C5
8	Jonathan Gonzales	IN	0	Mr.	0	0x81FE9DAB
9	Jonathan Bryant	IN	0	Mr.	2	0x88CD9573
10	Ramon Zhao	IN	0	Mr.	1	0xB9C5CB38
11	Jonathan Alexander	IN	0	Mr.	0	0x24C9FAAC
12	Ramon Lu	IN	0	Mr.	0	0x47B9C4B0
13	Jonathan Russell	IN	0	Mr.	0	0x9CA27DD4
14	Jonathan Griffin	IN	0	Mr.	1	0x89C92E86
15	Jonathan Diaz	IN	0	Mr.	0	0xE788B708

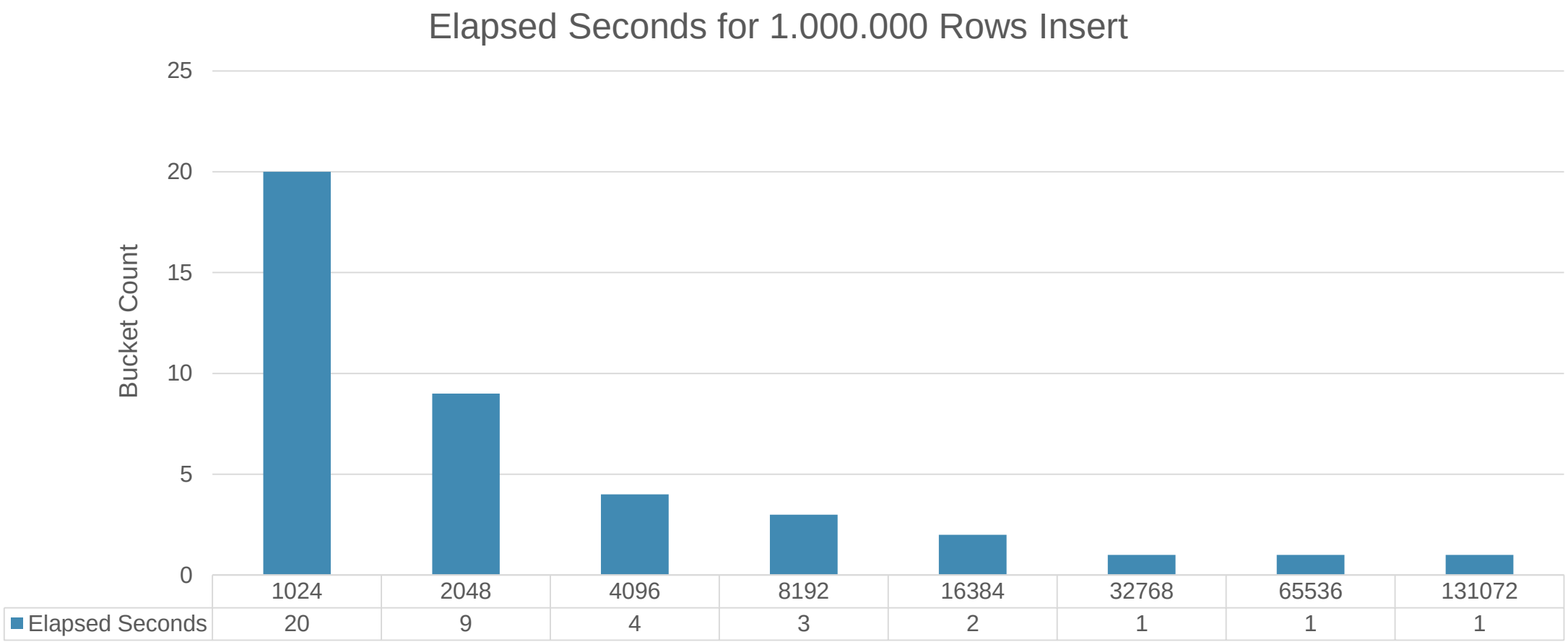
Hash Index

- E' bene scegliere correttamente la dimensione dell'Hash Bucket

- Se troppo basso ci sono collisioni. Le collisioni creano delle catene di puntatori (poiché un hash è legato a più valori) che rallentano le performance
  - In quanto bisogna scansionare «n» righe e non più una

- Valori troppo alti consumano memoria senza portare un beneficio effettivo
  - E no memory? no party! ☺

Hash Index


Elapsed Seconds for 1.000.000 Rows Insert
Bucket Count
25
20
15
10
5
0
1024
2048
4096
8192
16384
32768
65536
131072
Elapsed Seconds
20
9
4
3
2
1
1
1

BW Tree Index

- BW: BuzzWord ☺
  - Chiamato anche **Range Index**

- Variante lock & latch free di un B-Tree
  - Sviluppata da MS Research nel 2011
  - I puntatori delle pagine foglia hanno puntano a *Logical Page ID (PID)*
  - I PID vengono rimappati – tramite mapping tables – ad indirizzi fisici di memoria
  - Le pagine *non* vengono mai aggiornate
    - Rimosse e aggiunte
    - Aggiornamento della mapping table

demo

Hekaton

Multiversion Concurrency Control

- Per sfruttare a pieno le performance di un in-memory engine fortemente concorrente è necessario evitare di avere meccanismi sincronizzazione
  - No Lock, Latch o Spinlock!
- Bisogna cambiare l'approccio alla concorrenza, usando un modello nuovo: Multiversion Concurrency Control
  - Modello nato alla fine degli anni settanta / inizi anni 80
- Le modifiche ai dati e l'accesso agli stessi è sempre accompagnato da un timestamp che ne indica la versione

Multiversion Concurrency Control

- Le operazioni sui dati di tabelle in memory usano sempre un Optimistic Multiversion Concurrency Control (MVCC)

- Le righe non vengono modificate direttamente, ma vengono create nuove versioni della stessa riga con finestre temporali di validità più recenti

- Lavorando in concorrenza ottimistica, se due transazioni attive cercano di modificare la stessa riga, la seconda transazione viene abortita
  - E'importante gestire il retry dell'operazione ☺

Multiversion Concurrency Control

- Un contatore monotonico crescente, identificato come timestamp, viene incrementato ad ogni commit di una transazione

- Ogni versione di un record possiede due timestamps:
  - Begin-Ts – commit time della transazione che l'ha generata
  - End-Ts – commit time della transazione che l'ha "eliminata"

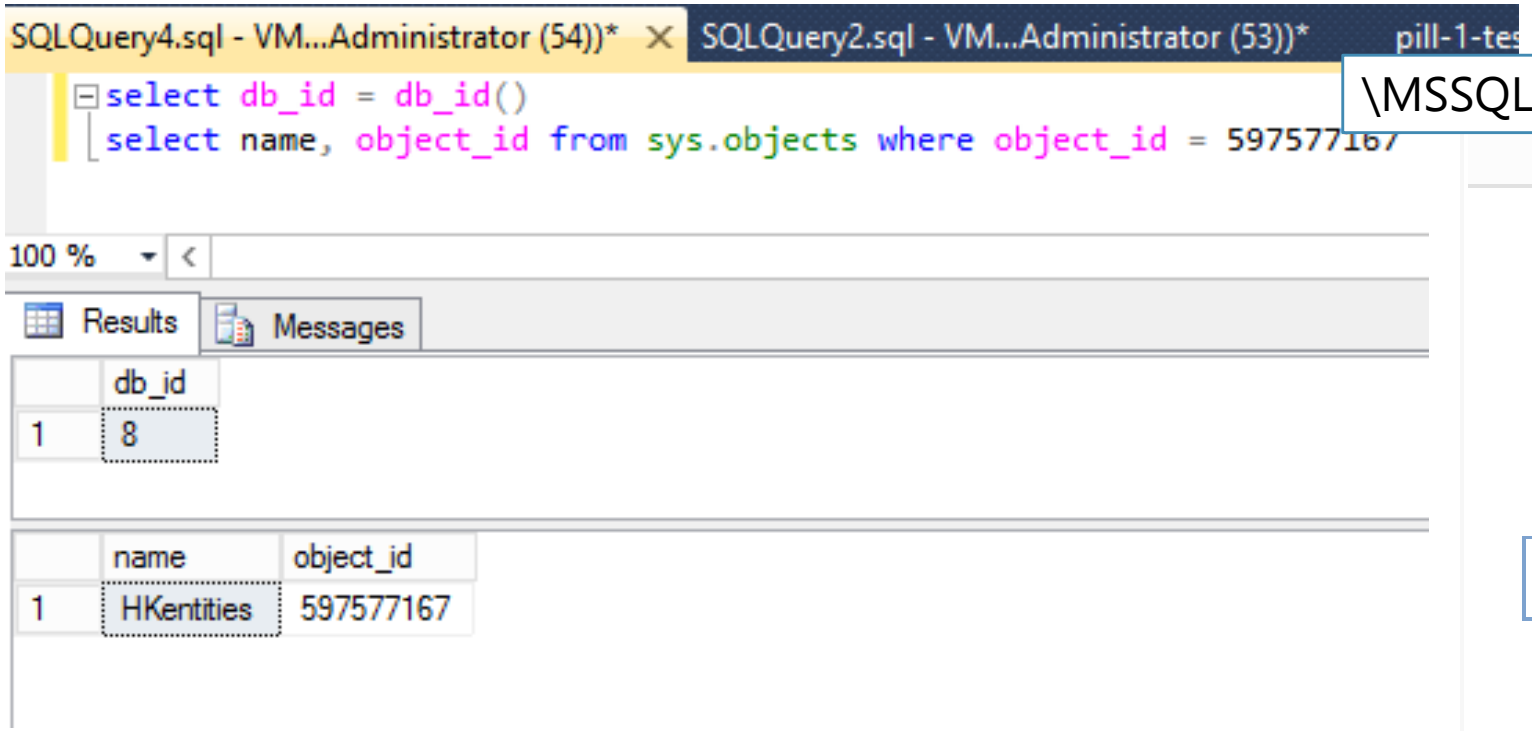
- Il periodo di validità (valid time) della versione di un record denota il range di timestamps in cui tale versione risulta visibile alle altre transazioni

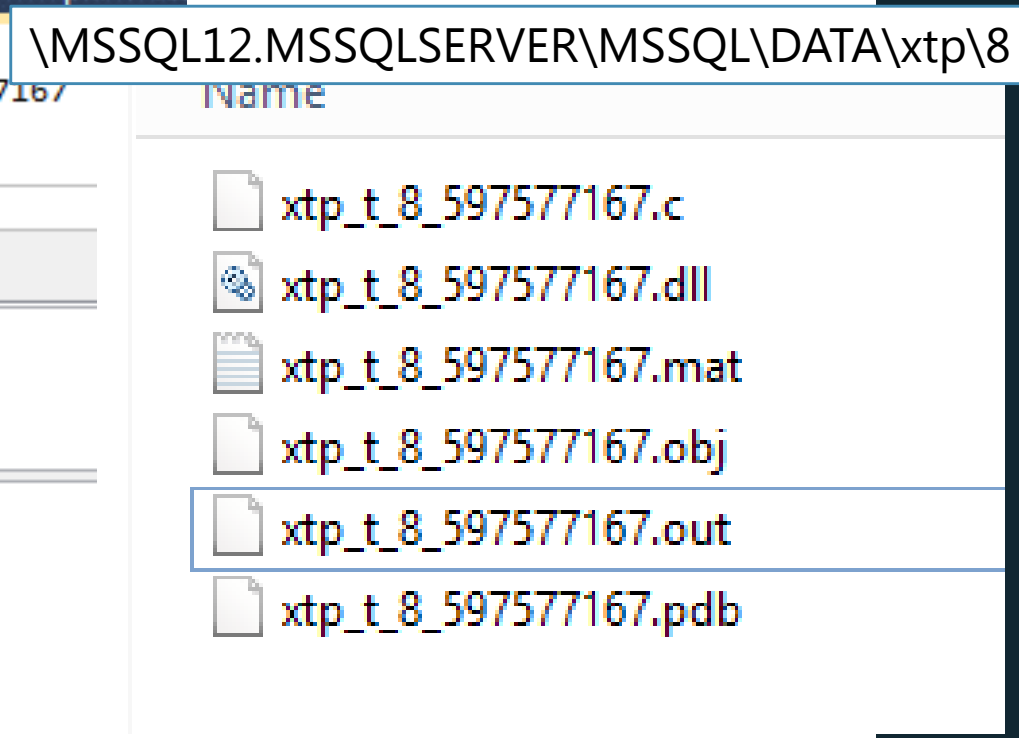
Multiversion Concurrency Control

- Sistema concettualmente simile allo SNAPSHOT Isolation, ma:
  - Nessun carico sul TempDB
  - Nessun blocco in UPDATE o DELETE
- Pienamente transazionale (ACID properties)
- Scritture nel Transaction Log drasticamente ridotte
  - Footprint minimo
  - Dati sporchi non vengono mai persistiti

Native Compilation

- Al momento della creazione le tabelle Hekaton viene generata una DLL per l'esecuzione di comandi DML sulla tabella





Native Compilation

- E' possibile anche creare delle stored procedure compilate in codice nativo
- Viene prodotta una DLL
- *Il piano di esecuzione viene generato al momento della creazione della stored procedure*
  - Quindi occhio ad avere dei dati realistici nella tabella
- Ci sono ancora parecchie limitazioni
- Performance boost notevole!

Native Compilation

- Le SP compilate in codice nativo hanno diversi limiti…i più importanti
  - Non possono accedere a tabelle non-hekaton
  - Non si possono usare tabelle temporanee
  - Non si possono usare subquery
  - L'unica collation utilizzabile è la BIN2
  - Si può usare solo l'operatore di INNER JOIN
  - Non si possono usare le funzioni ranking come ROW_NUMBER()
  - Ecc.
  - Ecc.
  - Ecc.
- Transact-SQL Constructs Not Supported by In-Memory OLTP
  http://msdn.microsoft.com/en-us/library/dn246937(v=sql.120).aspx

Native Compilation

- Nella creazione di una SP compilata nativamente, ci sono alcuni «obblighi»
  - E' necessario utilizzare il nuovo costrutto «BEGIN ATOMIC»
  - Bisogna specificare WITH SCHEMABINDING
  - Bisogna specificare EXECUTE AS

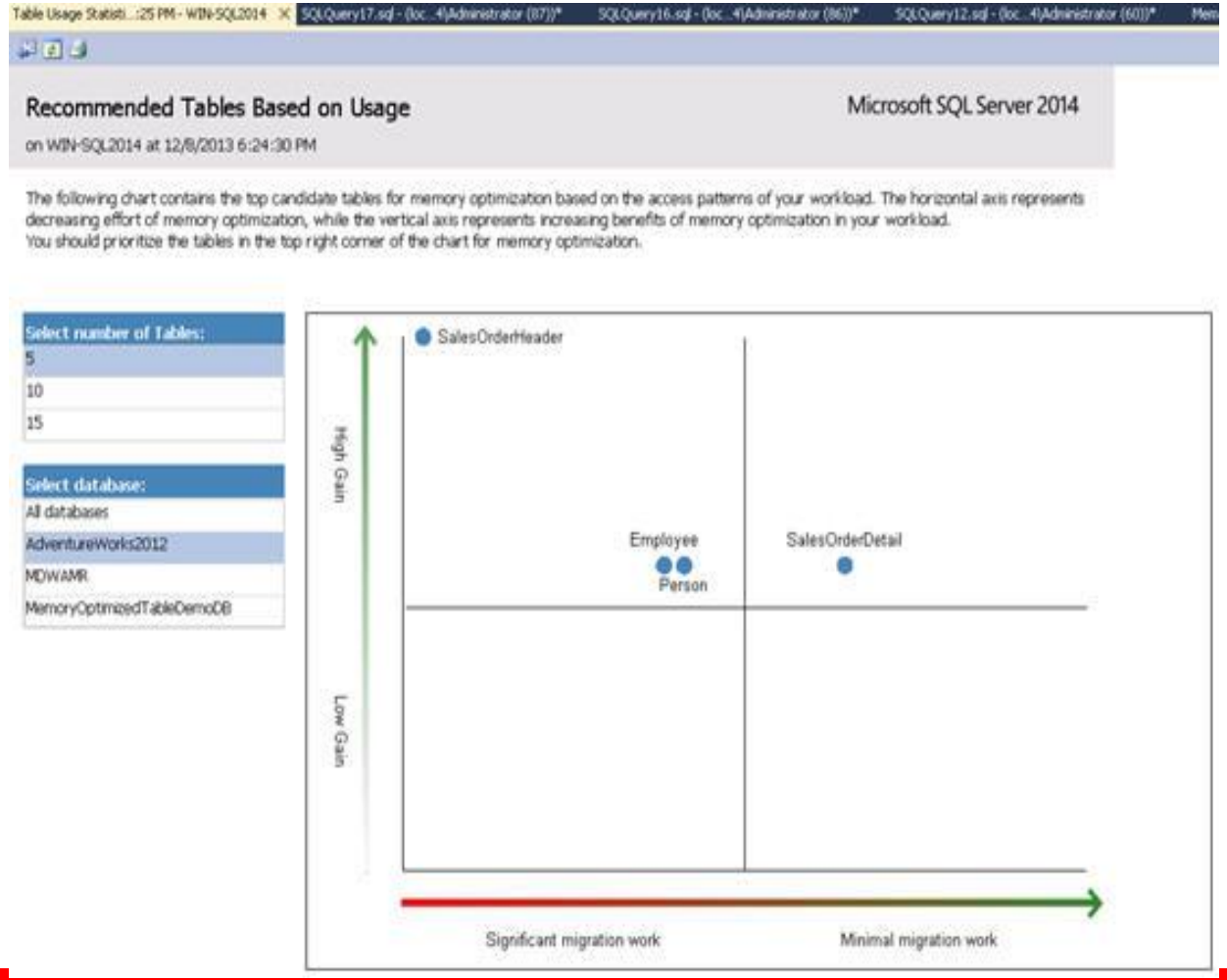
```
create procedure dbo.OrderInsert(@OrdNo integer, @CustCode nvarchar(5))
with native_compilation, schemabinding, execute as owner
as
begin atomic with
(transaction isolation level = snapshot,
language = N'English')

  declare @OrdDate datetime = getdate();
  insert into dbo.Ord (OrdNo, CustCode, OrdDate) values (@OrdNo, @CustCode, @OrdDate);
end
go
```

demo

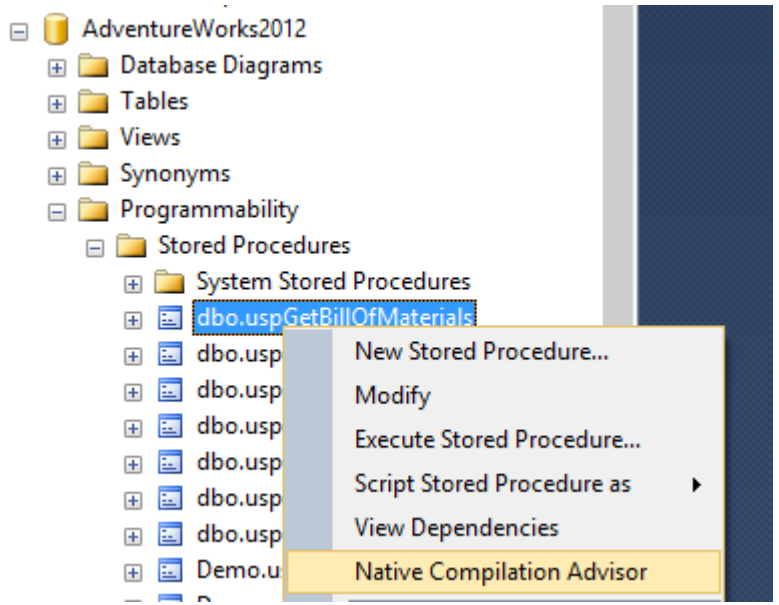
Native Compilation

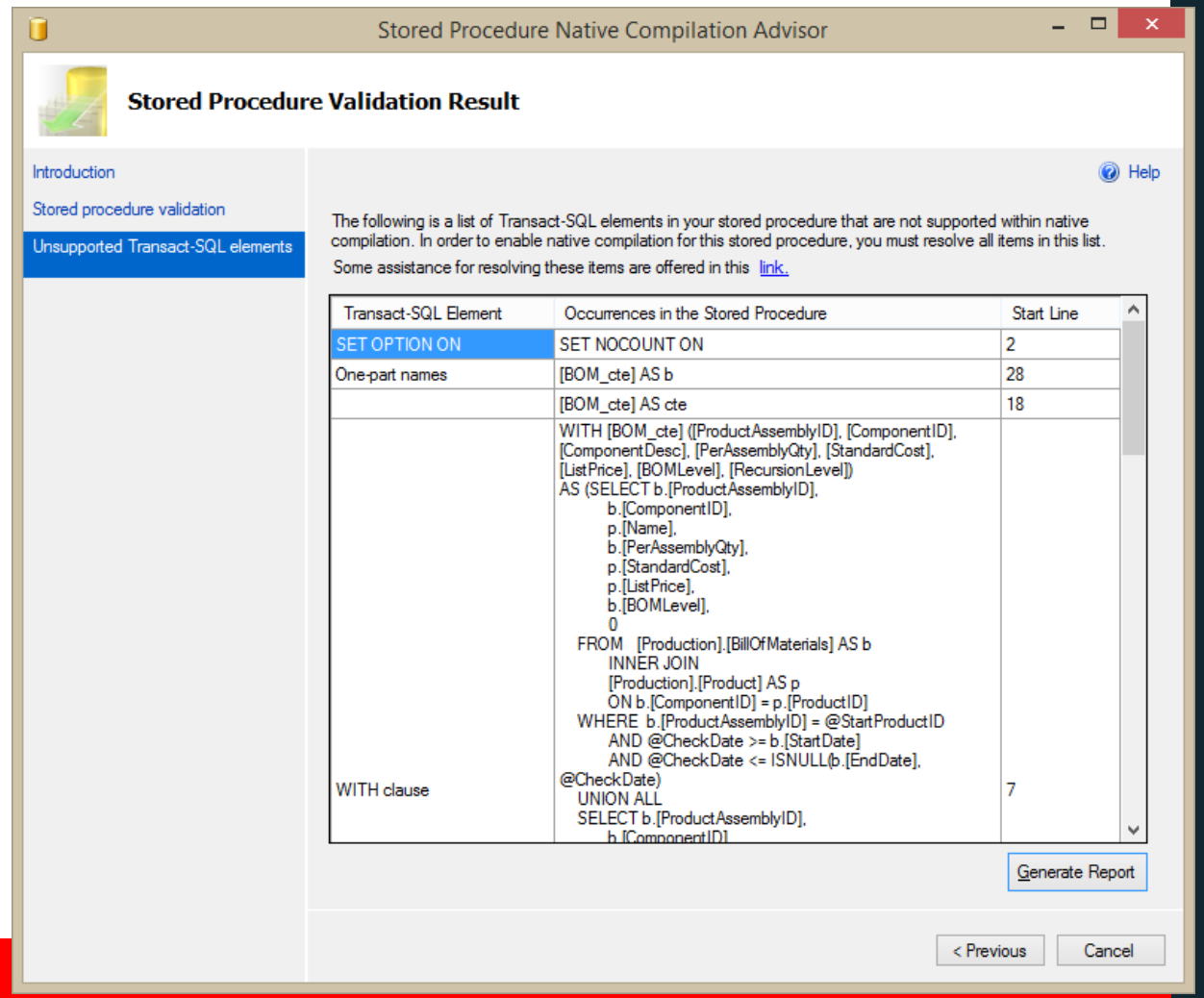
AMR – Analysis Migration & Reporting



- Basato sul Management DWH
- Raccoglie informazioni sull'uso
- Analizza
  - Tabelle
  - SP
- Consiglia quali migrare su Hekaton

Native Compilation Advisor





Recap

- TANTA carne al fuoco…ed abbiamo visto solo un'introduzione
- Numerose altre novità in SQL Server 2014
  - Sul canale VIMEO di UGISS (www.ugiss.org) saranno pubblicati video-pills su tutte le novità
- Se volete approfondire il discorso: www.sqlconference.it

Q&A

Tutto il materiale di questa sessione su

http://www.communitydays.it/

Lascia il feedback su questa sessione,

potrai essere estratto per i nostri premi!

Seguici su

Twitter @CommunityDaysIT

Facebook http://facebook.com/cdaysit

#CDays14