SQL Server 2014 In-Memory Tables (Extreme Transaction Processing)

Tony Rogerson, SQL Server MVP
@tonyrogerson
tonyrogerson@torver.net
http://dataidol.com/tonyrogerson

Who am I?

Freelance SQL Server professional and Data Specialist

Started out in 1986 – DB2, Oracle, SQL Server since 4.21a

SQL Server MVP since 97

Fellow BCS, MSc in BI, studying for Data Science PGCert

Allotment holder – onions and garlic were rubbish this year

Interested in Data Science especially commodity based distributed processing (NoSanMan!)

Agenda

What’s an in-Memory database?

SQL Server in-Memory technologies – OLAP and OLTP

Getting Started with Hekaton

Implementation ideas and moving forward

In-Memory Database/Table?

In-Memory Landscape (Nov 2013 – a small section!)

- Oracle – TimesTen (embedded into DB Cache, Exalytics)
- SAP – HANA
- IBM solidDB
- SQL Server xVelocity (next generation of Vertipaq) – Column Store
- SQL Server Hekaton (for OLTP – relational type workloads)

Database Cache OR proper in-Memory database

<u>Entire</u> DB in memory / <u>Selected tables</u> (mix with stable storage)

SQL Server in-Memory OLAP / OLTP

- Column Store (xVelocity - used to be called Vertipaq)
  - Column Store Indexing (read only)
  - New “CLUSTERED” column store index allowing ins/del/upd
  - Really in-memory or just compression?
- In-Memory Tables (Hekaton) – Extreme Transaction Processing (XTP)
  - Geared to new storage paradigm of Flash and multi-core machines
  - Memory and Flash optimised with new Bw-Tree (Buzz Word Tree) and Hash indexing
  - Natively Compiled Stored Procedures
  - Join in-memory data with legacy B-Tree/Heap on stable storage
  - Allows us to remove the need for Durability (hoorah!)
- StreamInsight – Complex Events Processing

Getting Started with Hekaton

Database composition

Hash Indexes – B+Tree or BW-Tree

Durability [or not] – Transaction Logging

Isolation

Interop or Native in-Memory

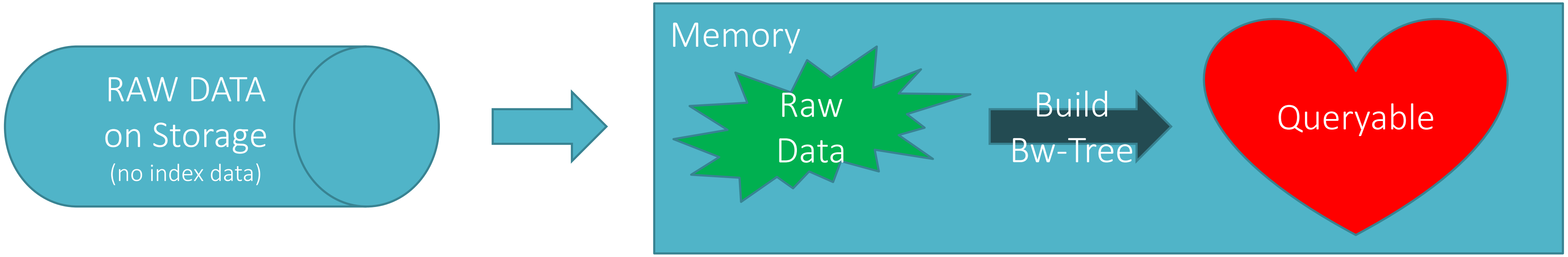
Native Compilation

Database Composition

```
CREATE DATABASE [AdventureWorks2014] ON
       PRIMARY
              ( NAME = N'AdventureWorks2012_Data',
                FILENAME = N'E:\SQLDATA\AdventureWorks2014_Data.mdf' ,
                ......... ),
       FILEGROUP [AdventureWorks2012_mod]
              CONTAINS MEMORY_OPTIMIZED_DATA  DEFAULT
                     ( NAME = N'AdventureWorks2012_mod',
                       FILENAME = N'E:\SQLDATA\inmem\AdventureWorks2012_mod' ,
                       MAXSIZE = UNLIMITED)
       LOG ON
              ( NAME = N'AdventureWorks2012_Log',
                FILENAME = N'E:\SQLDATA\AdventureWorks2014_log.ldf' ,
                .......... )
```

1 File Group,
1 or More files

Storage (Database Start Up)



Uses File-Stream

Data loaded, indexes built on Database recovery

Make sure the IO Sub-system can handle the load – it will swamp the IO's

Offline CHECKPOINT saves tables SCHEMA_AND_DATA durability to File-Stream (storage)

File Stream

Normal tables:

- Bulked up IO through lazywriter
- {over}-Writes are to set database structure
- Random IO causes latency issues
- Blocked IO causes Flash wear

XTP:

- Writes append to multiple 128MiB files
- No need to write Index changes or internal structures
- Just need the delta's for recovery

Use multiple files on the in-memory file group for performance

Balance – Buffer Pool v XTP

XTP competes with the Buffer Pool

Use Resource Pool to limit XTP

Base resource pool memory on the minimum requirement

Versioning needs to be considered – multiple versions of the row!

DEMO #1 - DB

CREATEDB.sql

New Index types

HASH index – for expressions of equality

- Hash of the index (~~key~~) columns – no DRI
- If it can’t seek it will scan – scan everything!

Range index – for everything else

- Bw(Buzz Word)-Tree
- Derivative of the B+Tree but
  - *Elastic page sizes*
  - *No update – updates through versioning and processing the delta’s on commit*

Hash indexing

Array [Mapping Table] of cells [Hash Buckets]
  - Eg. {CHECKSUM(“Jen Stirrup”), CHECKSUM(“Anthony Saxby”)}

Hash collision forms a chain of rows (1 bucket has a row chain – rows sharing same hash)

Reduce hash collisions by increasing the BUCKET_COUNT

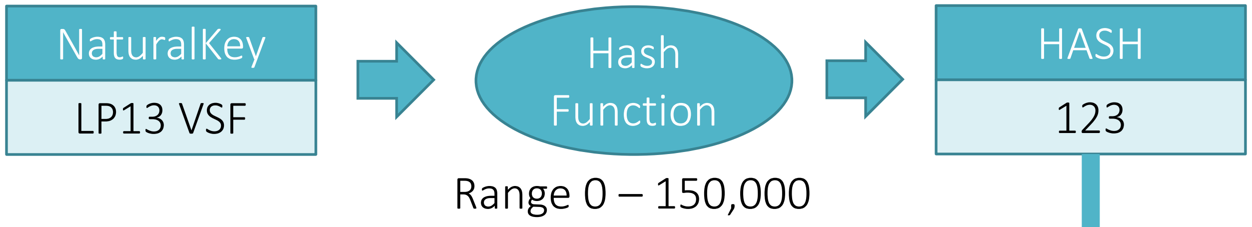
Ideal is BUCKET_COUNT = number of unique values x2 (depends on hash collisions too)

Buckets use memory

Careful on composite keys – the complete key is hashed making part lookups unable to use the index.

Character data-type required to be Latin1_General_100_BIN2
  - @ Database level
  - @ Column Level e.g. avarchar varchar(50) COLLATE Latin1_General_100_BIN2 not null

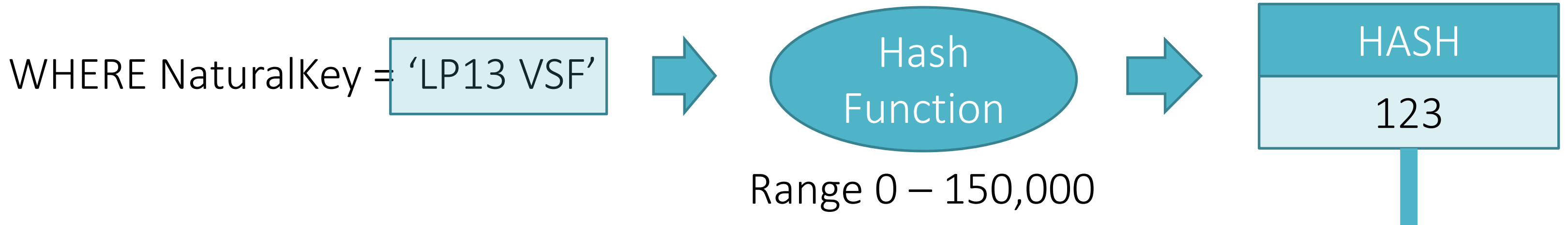
Hash Index



Hash Bucket #	Pointer (64bit)
0	Null
1	Null
//	//
122	Null
123	12312
124	Null
//	
150000	Null

NaturalKey	DateRegistered	Make	Colour	Memory Location
AA13 SAE	2013-01-01	Ford	Red	0
LP13 VSF	2013-05-01	Volkswagen	Blue	12312

Hash Index - Usage



Hash Bucket #	Pointer (64bit)
0	Null
1	Null
//	//
122	Null
123	12312
124	Null
//	
150000	Null

NaturalKey	DateRegistered	Make	Colour	Memory Location
AA13 SAE	2013-01-01	Ford	Red	0
LP13 VSF	2013-05-01	Volkswagen	Blue	12312

Hash Index – Range expressions

Hash [seek] is an equality / inequality only operation

WHERE NaturalKey > ‘LP13 VSF’ makes no sense

WHERE Hash(NaturalKey) > Hash(‘LP13 VSF’)

WHERE 123 > 123

```
print checksum( 'a' )
print checksum( 'b' )
print checksum( 'c' )
print checksum( 'd' )
print checksum( 'e' )
```

```
114
121
122
106
81
```

WHERE HASH(NaturalKey [a..e]) > HASH(‘a’)

Hash Index (reality)

Hash Bucket #	Pointer (64bit)
0	Null
1	Null
//	//
122	Null
123	12312
124	Null
//	
150000	Null

NaturalKey	DateRegistered	Make	Colour	Memory Location	Begin/End Timestamp	IDX Ptr
LP13 VSF	2013-05-01	Volkswagen	Blue	12312		
KK10 SED	2010-05-01	Ford	Green	12355		
LP13 VSF	2013-05-01	Volkswagen	Blue	12360		
KK10 SED	2010-05-01	Ford	Green	1290		

All Natural Key
Index Values
hash down
to bucket 123

Bw-Tree (Latch-Free, Log-Structured)

Designed for Multi-Core, Main Memory and Flash based Storage

Avoids thread latch blocking – move towards MVCC and writer-writer fail

Record updates, splits done as “delta’s” – updates combined on write out

Exploits fast sequential writes (reducing random writes) – increases flash life

Mapping table maps Logical Page ID’s to Flash offset or memory pointer
- Physical Location of Nodes allow to change without propagation to root
- Tree Nodes are logical (why they aren’t stored)
- Page size is elastic – split when convenient rather than on size (e.g. at 8KiB)

Latch-Free
- Use Compare and Swap (CAS) instructions
- Persistence of Thread exec helps preserve processor core cache

http://sites.computer.org/debull/A13june/bwtree1.pdf
http://research.microsoft.com/en-us/um/people/justinle/papers/ICDE2013_bwtree.pptx
http://research.microsoft.com/pubs/178758/bw-tree-icde2013-final.pdf

DEMO

CREATEDATA.sql

Key Concepts - Durability

What is Durability? ACI<u>D</u>

Tables:

- SCHEMA_ONLY

Setting: DELAYED_DURABILITY

- Database {Disabled, Allowed, Forced}
- Procedure option, Transaction option (on COMMIT)

Reduced Logging

- No UNDO records * No need – everything fits in memory, no spill required
- No Index records * No need – re-hash on loading data – take the hit on db start

DEMO

DURABILITY.sql

Key Concepts – MVCC #1

READ COMMITTED (Writer block reader)

MVCC (Multi-Version Concurrency Control)

- MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT
- WITH (SNAPSHOT), SET TRANSACTION ISOLATION LEVEL SNAPSHOT

Use WITH (SNAPSHOT) on FROM, COMMIT TRAN, UPDATE {table} etc.

Transaction preserves a “point in time” through versions

A	1	1	2							
B		1	1	1	1					
C			2	2	2	2				

T

Key Concepts – MVCC #2 [in-memory]

READ_COMMITTED only for single statement transactions (not transaction nested)

If there is a transaction in-mem use SNAPSHOT

Watch out for MVCC – more versions = more memory used

Write-Write conflict – application behaviour change?

Applications dependent on locking - rethink

DEMO

MVCC.sql

Key Concepts – Native (Compiled)

Compiled Stored Proc stored in file system (..\mssql\data\xtp\)

Loads of restrictions

Incredible performance boost

Plan created and fixed when CREATE PROC ran

Can you OPTIMISE FOR

Make sure you have data in your tables – of production size (statistics!)

DEMO

NATIVECOMPILED.sql

Design considerations for native compiled stored procedures

Benefits

Efficient business-logic processing

In-Memory OLTP Tech Pillars

T-SQL compiled to machine code

- T-SQL compiled to machine code via C code generator and Visual C compiler
- Invoking a procedure is just a DLL entry-point
- Aggressive optimizations at compile-time

Drivers

Hardware trends

Stalling CPU clock rate

	Native compiled stored procedures	Non-native compilation
Performance	**High**. Significantly less instructions to go through	No different than T-SQL calls in SQL Server today
Migration strategy	Application changes; development overhead	Easier app migration as can still access memory-optimized tables
Access to objects	Can only interact with memory-optimized tables	All objects; access for transactions across memory optimized and B-tree tables
Support for T-SQL constructs	Limited	T-SQL surface area (limit on memory-optimized interaction)
Optimization, stats, and query plan	Statistics utilized at CREATE -> Compile time	Statistics updates can be used to modify plan at runtime
Flexibility	Limited (no ALTER procedure, compile-time isolation level)	Ad-hoc query patterns

Ref: SQL Server 2014 Mission Critical Performance Level 300 Deck.pdf

Key Concepts – Querying

“Status Quo” in the most part

Mix in-memory tables with on-storage with some restrictions

If in a transaction you must use WITH(SNAPSHOT)

Migration path towards full blown performance of Hekaton

DEMO

QUERYING.sql

HA / DR

Works with Availability Groups

Works with Native SQL Server database and log backups

Does not replicate data from SCHEMA_ONLY tables

Implementation Ideas

AMR (Analysis Migration and Reporting) over your existing production box

Consider how to apply in-memory tables given how hash/bw-tree indexes work

Durability: SCHEMA_ONLY

- ETL
- Session State
- Real-time Analytics calculations (from raw data stored in NoSQL)
- Part of a LAMBDA archiecture

Durability: SCHEMA_AND_DATA with DELAYED_DURABILITY

- High insert activity that you don’t mind losing some of in case of failure

Durable

- Latch contention

References

SQL Server 2014 CTP2 Product Guide

http://www.microsoft.com/en-us/download/confirmation.aspx?id=39269

BW-Tree

http://research.microsoft.com/apps/pubs/default.aspx?id=178758

Microsoft Research Main Memory Databases Project page

http://research.microsoft.com/en-us/projects/main-memory_dbs