

SQL Server 2014 Memory Optimized OLTP

Bob Beauchemin
bobb@SQLskills.com



Outline

- Pillars of In-Memory OLTP
- Memory-Optimized Tables details
- Compiled Stored Procedures details
- Multi-version concurrency control
- Programming differences
- Operational differences
- Migrating



2

© SQLskills. All rights reserved.
<http://www.SQLskills.com>



OLTP and OLAP

- **OLTP and OLAP are general categories for data applications**
 - OLTP applications
 - Read one or very few rows at a time
 - Read all or many of the columns in a row
 - OLAP applications
 - Reads many rows – scans or range scans
 - Reads very few columns from a row
- **SQL Server 2014 introduces technologies to speed up OLTP systems**
 - Called In-Memory OLTP
 - Formerly called Hekaton
 - Also known as XTP (Extreme Transaction Processing)
- **In-memory OLTP is an Enterprise Feature**

In-Memory OLTP Use Cases

- **High data insertion rate from multiple connections**
 - Eliminate contention and reduce logging
- **High-performance reads with periodic batch inserts/updates**
 - Low latency data retrieval, eliminate contention
- **Intensive business logic/computation in stored procedures**
 - Minimize code execution time, eliminate contention
- **Low latency business transactions**
- **Web server session state management**
 - Additional IO reduction when using non-durable tables

Pillars of In-Memory OLTP

- **Memory-optimized tables**
 - Separate storage engine
- **Compiled stored procedures**
 - Separate procedure and query processor
- **Lock and latch-free concurrency transaction processing**
 - Separate transaction management implementation
- **Built in to SQL Server**

Components

- **XTP storage engine**
 - Storage management
 - Hash and range indexes on tables
 - Transactional operations
 - Checkpoint
 - Recovery
 - High Availability
- **XTP compiler**
 - Compiles procedures into native code
 - Produces native access routines for in-memory objects
- **XTP runtime system**
 - Integration with SQL Server resources
 - Library of functions for compiled procedures

Getting Started

- **You must have 64 bit SQL Server & support cmpxchg16b instruction**
- **Database needs to have filegroup for memory-optimized data**
 - Only one filegroup for memory-optimized allowed
 - Filegroup can have multiple containers
 - Checkpoint data is stored in filestreams
- **Carefully consider collation**
 - Of your database
 - Of character-based columns
 - Learn the T-SQL COLLATE clause
- **Can consider**
 - MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT database option
 - Delayed durability
- **Enable instant file initialization on the instance**

Storage

- **Persistent memory-optimized objects stored as streams**
 - Also known as Checkpoint File Pairs (CFP)
 - Each CFP is one data file and one delta file
 - Limit of 8192 pairs per DB
- **Data Files**
 - 128 MB files, 16 MB on machines with <= 16 GB physical memory
 - 8 files initially allocated
 - Checkpoint-based sets of rows for all tables
- **Delta Files**
 - Stores IDs of deleted rows
- **Files are append-only**

Memory-Optimized Tables

- **Tables and table variables**
 - Table variables must be strongly typed
- **Durable and non-durable tables**
 - Table variables always non-durable
- **Durable tables persisted into filestream-like filegroup**
 - Must define a special filegroup
 - Multiple containers supported
- **Durable tables are re-loaded at startup time**
 - Must have enough memory
- **Non-durable are not persisted or logged**
 - For ETL staging or temp table replacement

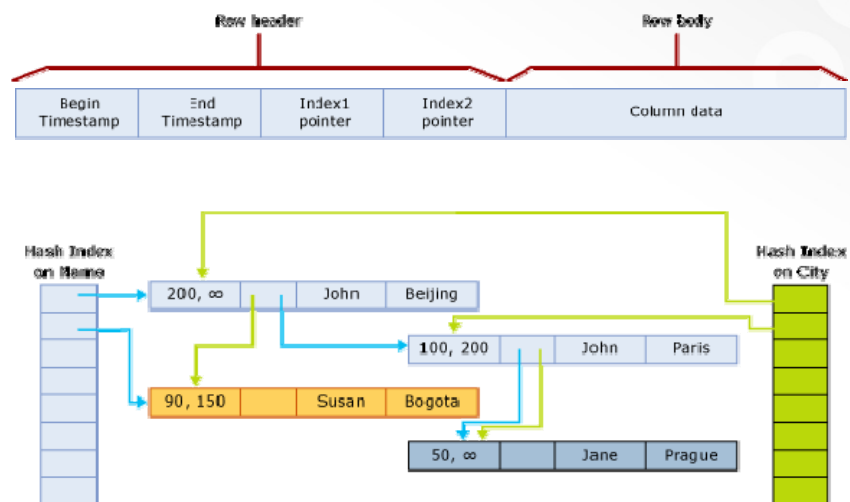
Memory Considerations

- **Memory-optimized tables' memory is non-pageable**
 - Different memory clerk than buffer pool
 - Covered by max_server_memory setting
- **~80% hard limit for memory-optimized tables**
 - Changes fail after limit hit
- **Can also limit through resource governor**
 - Limit is on total amount per-pool allocated to XTP
 - Resource governor per-workload group memory limits do not apply to XTP
- **Memory is not allocated in pages**
 - Linked lists over hash buckets
 - Tables still have 8060 byte limit

Table Layout and Versions

- Uses a multi-version engine
- Each row has a BeginTimestamp and EndTimestamp
- Inserts
 - Insert new rows with BeginTimestamp of tx timestamp, EndTime of infinity
- Updates
 - Insert new row - BeginTimestamp of tx timestamp, EndTimestamp of infinity
 - Set EndTimestamp of old row to tx timestamp
- Deletes
 - Sets EndTimestamp of old row to tx timestamp
- Row access is based on statement/transaction begin time
- Old rows are garbage collected
 - When no longer needed

Memory-Optimized Tables



Memory-Optimized Table Code

- **When a memory-optimized table is created a DLL is created**
 - One DLL per table with access methods
- **DLL is not stored in SQL Server**
 - In a subdirectory of ...\\Data
- **DLLs are recreated and reloaded at startup time**
 - For memory-optimized tables
- **Can be observed**
 - Creation and compilation with XEvent events
 - With sys.dm_os_loaded_modules

Indexing

- **Two types of indexes supported**
 - Hash index – useful for point queries
 - Range index – useful for range queries
- **Up to 8 indexes per table**
- **Tables must have at least one index**
 - Durable tables also need primary key
 - Index columns must be non-null, need not be unique
- **Indexes must be declared at CREATE TABLE-time**
 - No CREATE/ALTER INDEX or ALTER TABLE
 - Fragmentation and fill factor do not apply
- **Indexes are not stored on disk**
 - Created at load time
- **All indexes are covering**
 - Because they are pointer-based

Hash Indexes

- **Useful for point queries only**
 - Equality or IN predicates
- **Number of hash buckets must be declared with table**
 - 8-byte pointers
 - Should be 2x expected number of rows
 - Internally rounded up to the next power of two
 - Hard limit of 1,073,741,824 buckets
- **Hash collisions will chain rows, decrease performance**
 - Increase bucket_count
 - Don't use hash indexes if many duplicate index keys
- **Less useful for composite keys**
 - Whole key must match

Range Indexes

- **Range indexes are declared as NONCLUSTERED indexes**
 - Uses new BW-tree – lock and latch free
- **Range indexes are unidirectional**
 - Do not support backward scans
- **Pages are stored differently**
 - Not fixed size, but still maximum 8k
- **Range indexes handle updates differently than B-tree**
 - Index pages are never updated
 - They are replaced with a new page and mapping table is updated
 - Atomic replace uses compare-and-swap machine instruction
 - Each insert, update, delete to key value on that page
 - Produces a page w/delta record indicating the change that was made.

Compiled Procedures

- **Compiled at CREATE PROCEDURE-time**
 - T-SQL is translated to C code
 - Native DLL produced
- **Re-created on first use after system shutdown**
- **Query plan is created at CREATE PROCEDURE-time**
 - Statistics should be update before procedure creation
 - No auto-statistics update
 - No ALTER PROCEDURE, CREATE... WITH RECOMPILE, etc
- **Compiled procedures can only access memory-optimized tables**
 - Not disk-based ("traditional") tables
 - Not temporary tables, use memory-optimized table variables instead
- **Uses compiled code generated for memory-optimized tables**
 - Access routines
 - Supports a subset of plan iterators

CREATE PROCEDURE declarations

- **Compiled procedures must be declared with**
 - NATIVE_COMPILATION
 - SCHEMABINDING
 - EXECUTE AS OWNER/SELF/user (EXECUTE AS CALLER not allowed)
 - BEGIN ATOMIC
- **The BEGIN ATOMIC statement declares**
 - Transaction isolation level
 - Language
 - DATEFORMAT, DATEFIRST – optional
 - DELAYED_DURABILITY = ON | OFF
- **Can specify parameters and variables as NOT NULL**
 - Compiled procedures only

Atomic Blocks

- **Each compiled procedure is an atomic block**
 - Cannot begin or end transactions in compiled procedures
- **It's transactional standalone or composed in user transactions**
- **Composition with user transactions**
 - Accomplished with savepoints, not autonomous transactions
- **Errors during procedure execution will rollback procedure code**
- **TRY-CATCH to limit rollback**
 - But no deadlocks occur
- **Transactions, procedures should be kept short**

Compiled Procedure Use Cases

- **The more procedural code (and more singleton rows) that are processed, the bigger the benefit from compiled procedures**
 - Aggregation
 - Nested-loop joins
 - Multi-statement select/insert/update/delete
 - Complex expressions
 - Procedural logic, like conditionals and loops

Troubleshooting

- **Code is never auto-recompiled**
 - Update statistics before CREATE
 - Must drop and recreate to update plan
- **Code consists of**
 - Table IO from per-table DDLs
 - Built-in functions code
- **Optional DMV information**
 - Turn on query_stats and procedure_stats info on demand
 - Query_stats can be per-procedure, procedure_stats is global
 - Only timings, execution counts, last compiled/executed
- **No T-SQL debugging**

Transactions and Isolation

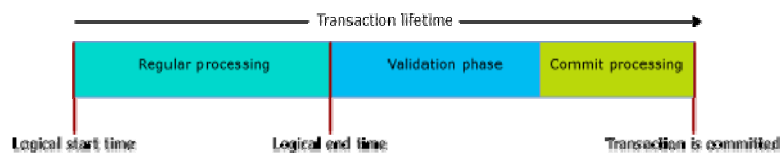
- **In-Memory OLTP uses multi-version engine**
 - MVCC = Multi-version concurrency control
 - Multiple versions of rows in memory
 - Only one version is valid at a given time
 - Which one is read depends transaction's begin time
- **For access from interpreted SQL**
 - MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT on CREATE DATABASE
 - READ COMMITTED supported for single statements
 - Otherwise statements need isolation level hints
- **Optimistic concurrency used**
 - No locks and no latches
 - Assumes other transactions will commit (doesn't wait)
- **Error-handling and retry logic required**

Isolation Levels

- **SNAPSHOT**
 - Read-consistency as of beginning of transaction
- **REPEATABLE READ**
 - Row that are read cannot change from beginning to end of tx
- **SERIALIZABLE**
 - All scans are repeatable too
 - No phantoms
- **Write-write conflicts cause rollback**
 - Immediately
- **Violations of isolation level cause rollback at commit time**
 - In prepare phase

Transaction Phases

- **Transactions consist of three phases**
- **Transaction created – begin timestamp**
 - Normal Processing
 - Preparation
 - Gets end timestamp
 - Must resolve transaction dependencies
 - Checks for concurrency violations
 - Postprocessing
 - Writes end timestamp
- **Transaction terminated**



Isolation Level Implementation

- **Each transaction that updates data keeps WRITE_SET**
 - Used to update timestamps quickly when transaction commits
- **Repeatable read transactions use READ_SET**
 - Used to check that rows have not changed between transaction logical begin time and logical end time
 - Rows reread at validation time
- **Serializable transactions use READ_SET and SCAN_SET**
 - Validates reads like repeatable read
 - Validates that no phantom inserts occurred between transaction logical begin time and logical end time
 - Ranges re-scanned at validation time

Speculative Reads

- **If another transaction is in progress**
 - Readers assume other transaction will commit
 - Can do speculative reads
 - Can speculate and ignore versions
- **This causes transaction dependencies**
- **Must be resolved before proceeding**
 - Can cause waiting here

Garbage Collection

- **Two kinds of garbage collection**
 - Garbage collection of rows in memory
 - Garbage collection of checkpoint data files (merge)
- **Garbage collection of rows in memory is cooperative**
 - User threads can collect old versions during scans
 - System GC process runs once per minute (or more)
 - Assigns GC queue items to user threads
- **Merge operation merges data files**
 - If two consecutive data files < 50% full
 - Merge can also be forced by system stored procedure

But there are limits...

- **Limits on memory-optimized tables**
 - Only some data types
 - BIN2 collations on columns with indexes
 - 8060 byte row limit
 - No constraints (except NULL), triggers
- **Limits on T-SQL statements**
 - TRUNCATE TABLE unsupported
- **Limits on T-SQL constructs usable in compiled procedures**
 - Compiled procedures can only access memory-optimized tables
 - Compiled procedures can't call other procedures
 - Some constructs (e.g. OUTER JOIN) unavailable
 - Many functions unavailable
 - ORDER BY and comparison – BIN2 collations only
- **Limits on transaction isolation levels**
 - And limits on cross-container transactions

Programming Differences

- **Uniqueidentifier is a useful hash key**
 - No worries about index fragmentation
 - Sequence object can be a bottleneck
- **You must account for errors**
 - Transactions more likely to roll back
- **No parallelism for query referencing memory-optimized tables**
- **Deleting a range of rows with concurrent insert/update will likely fail**
- **All parameters for native procs assumed to have unknown values**
 - OPTIMIZE FOR hint is useful more often

Programming Differences (2)

- **Design changes**
 - No constraints or triggers – integrity redesign
 - No large data types – table redesign
 - Hash index only used for point queries – index redesign
- **Smaller function library and query surface area**
 - You may need to write your own equivalents
 - Some pre-written examples

Management Differences

- **No index rebuilds needed**
 - Indexes rebuilt at startup
- **No automatic statistics updates and query plan rebuilds**
 - Must manually rebuild stats and compiled sprocs
 - Must specify NORECOMPUTE and FULLSCAN
- **No schema changes without dropping existing**
 - No ALTER TABLE
 - No ALTER PROCEDURE
 - Compiled sprocs must be declared WITH SCHEMABINDING
- **Availability groups supported**
 - Not replication or database mirroring
 - Non-durable tables don't get replicated in AGs

Management Differences (2)

- **Must have sufficient memory**
 - Must have strategy to deal with out-of-memory
 - XTP competes with the buffer pool
- **Table population is parallelized per-filegroup**
 - Multiple memory-optimized filegroups help
- **Log speed is the most limiting resource**
 - Some use cases for delayed durability
 - Logging is more efficient in general
- **Limit 256 GB – limit applies to durable tables**
- **Cannot drop memory-optimized filegroup**
- **Statement-level audit not supported in compiled sprocs**

Instrumentation

- **Extended Events Categories**
 - Checkpoint
 - Deploy
 - GC
 - Transaction
 - XTP
- **DMVs**
 - Instance-wide – sys.dm_xtp...
 - Database-specific – sys.dm_db_xtp..
- **Perfmon Counters**
 - XTP series
- **Metadata**
 - sys.heap_indexes
 - New fields in sys.tables, sys.table_types, sys.procedures, others

Migrating to In-Memory OLTP

- **Run AMR (Analysis, migrate, and support) Tool**
 - Set up Management Data Warehouse
 - Enable Data Collection Sets
 - Table Usage Analysis
 - Stored Procedure Usage Analysis
 - Run workload
 - Generated AMR Reports (MDW database/Reports)
 - Doesn't add much overhead to production server
 - Requires 2014 SSMS
- **Memory Optimization Advisor and Native Compilation Advisor**
- **Migrate Tables to Memory-Optimized Tables**
 - Workarounds for some common limitation in BOL
- **Migrate Procedures to Compiled Stored Procedures**
 - When possible

Migration Best Practices

- **Get a baseline**
- **Test a realistic workload**
 - Not just batches of insert statements
 - Right transaction mix to test locking vs lock-free
 - Use distributed replay with test suites
- **It doesn't fix everything**
 - More than 60 cores may affect performance

Summary

- **In-Memory OLTP feature consists of**
 - Memory-optimized tables
 - Multi-version engine
 - Hash and range indexes
 - New, pointer-based layout (no pages)
 - Persistent schema_and_data or just schema persistent
 - DLL-per-table for data access
 - Compiled stored procedures
 - T-SQL compiled into C code, compiled into DLLs
 - Atomic blocks
 - Error-based rollback
 - Declarative isolation level and other settings
 - Lock and latch-free access
 - Multi-version engine
 - Multiple isolation levels supported
- **Part of SQL Server**
 - Not a separate add-on

Resources

- **Books Online – In Memory OLTP section**
- **Whitepapers**
 - SQL Server In-Memory OLTP Internals Overview
 - In-Memory OLTP – Common Workload Patterns and Migration Considerations
 - Hekaton: SQL Server's In-Memory OLTP Engine (Sigmod 2013)
 - High-Performance Concurrency Control Mechanisms For Main-Memory Databases
 - The BW-Tree: A B-tree for New Hardware Platforms
- **Blogs**
 - SQL Server Team Blog
 - My blog
 - Tony Rogerson's Rambling on Data
 - Blakhani – Help: SQL Server - A-Z of In-Memory OLTP