**In-Memory OLTP Best Practices / Cheat Sheet**

**Understanding the requirements for In-Memory OLTP features**

* Follow [hardware and software for installing SQL Server](https://docs.microsoft.com/en-us/sql/sql-server/install/hardware-and-software-requirements-for-installing-sql-server?view=sql-server-ver15) documentation.
* Understanding additional requirements for Memory-Optimized tables
  + Available on SQL Server 64-bit versions
    - SQL Server 2016 SP1 ( or later ), and edition
    - SQL Server 2016 pre-sp1 ( Enterprise or Developer )
    - SQL Server 2014 ( Enterprise or Developer )
  + Enough memory to hold
    - data in-memory optimized tables and indexes
    - additional memory to support online workload
    - Ensure enough memory resources are available when running SQL Server in a VM
      * Memory reservation
      * Dynamic Memory, Minimum RAM
    - Enough free disk space ( 2 to 4 times the durable memory-optimized table)
    - Processor that supports instruction cmpxchg16b
    - [Requirements for using memory-optimized tables](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/requirements-for-using-memory-optimized-tables?view=sql-server-ver15)

**Usage Scenarios**

Determine what issue you are solving. Does it fit into to some of the examples provided? If yes, moving to the planning phase is the next step recommendation.

* High-throughput and low-latency transaction processing
  + Performance-Critical transaction use in-memory tables
  + Move as much business logic into native compiled stored proc for best performance
  + [Financial](https://customers.microsoft.com/story/because-a-second-is-too-long-to-wait-this-financial-services-firm-is-updating-its-trading-software) and [Gaming](https://customers.microsoft.com/story/when-an-online-gaming-company-doesnt-want-to-risk-its-future-it-bets-on-sql-server-2016) case studies
* Data ingestion, including IOT
  + Ingest large volumes of data from different sources
  + Common patterns
    - Including ingesting sensor reads and events
    - Managing batch updates from multiple sources, minimizing impact on concurrent read workload
  + [Implementation considerations](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/overview-and-usage-scenarios?view=sql-server-ver15#implementation-considerations-1) when using this pattern
  + [Improved batch performance](https://customers.microsoft.com/story/data-services-firm-gains-real-time-access-to-product-d) and [Ingestion](https://customers.microsoft.com/story/strengthening-data-security-and-creating-more-time-for) case studies
* Caching and session state
  + Really attractive for session state (ASP.NET application) and caching
  + [Gaming site](https://customers.microsoft.com/story/gaming-site-can-scale-to-250000-requests-per-second-an) and [mid-tier](https://blogs.msdn.microsoft.com/sqlcat/2016/10/26/how-bwin-is-using-sql-server-2016-in-memory-oltp-to-achieve-unprecedented-performance-and-scale/) use cases for in-memory OLTP
* [TempDB Object replacement](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/faster-temp-table-and-table-variable-by-using-memory-optimization?view=sql-server-ver15#a-basics-of-memory-optimized-table-variables)
  + Leverage to replace temp tables, table variables, and table variable functions
  + Utilize if you are looking to reduce temp object CPU and log IO.
  + Note [Memory-Optimized TempDB Metadata](https://docs.microsoft.com/en-us/sql/relational-databases/databases/tempdb-database?view=sql-server-ver15#memory-optimized-tempdb-metadata) can be leveraged to reduce TempDB metadata contention
  + [TVP for IOT ingestion](https://blogs.msdn.microsoft.com/sqlserverstorageengine/2016/04/07/a-technical-case-study-high-speed-iot-data-ingestion-using-in-memory-oltp-in-azure/) and [swapping tables in TempDB](https://customers.microsoft.com/story/sentryone-partner-professional-services-sql-server-azure) use cases
* ETL
  + Using non-durable memory-optimized tables for data staging
  + Use natively compiled stored procedures to do transformations on your staging tables

**Planning**

Utilize the transaction performance analysis reports to determine if the table and or stored procedure should be ported into In-Memory OLTP.

* Run Transaction Performance report against representative production workload to yield
  + Analyze workload determining hot spots where in-memory could potentially help
  + Helps plan and execute the migration to In-memory OLTP identifying incompatibilities and potential impact
* Review Transaction Management reports analysis
  + Tables
    - Scan statistics
      * Looking for high percent of total accesses. The higher the percentage indicates higher table utilization which could yield benefits from In-Memory OLTP.
      * Examine lookup and ranged scan statistics yields possible gains by converting to In-Memory OLTP due to its optimistic concurrency design.
    - Contention Statistics
      * Examine percent of total waits, latch statistics, and lock statistics
      * Tables with high percentage in any of these categories could yield significant performance gains by migrating to In-Memory OLTP tables.
    - Migration Difficulties
      * Provides a difficulty rating
      * Compare these ratings to the objects identified in the scan and contention statistics to help rank which tables should be ported to In-Memory.
    - [Should Table be Ported](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/determining-if-a-table-or-stored-procedure-should-be-ported-to-in-memory-oltp?view=sql-server-ver15#tables)
  + Stored Procedures
    - Execution Statistics
    - Table Refences
    - [Should a Stored Procedure be Ported?](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/determining-if-a-table-or-stored-procedure-should-be-ported-to-in-memory-oltp?view=sql-server-ver15%23stored-procedures)
  + Once artifacts are identified for the tables and stored procedure reports generate an In-Memory OLTP checklist.
    - Generates a checklist of any table or stored procedure not supported
    - [How to generate In-Memory OLTP Migration checklist](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/determining-if-a-table-or-stored-procedure-should-be-ported-to-in-memory-oltp?view=sql-server-ver15#generating-in-memory-oltp-migration-checklists)
* Additional In-Memory Planning considerations
  + Table must be offline to covert
  + Examine the exhaustive list for [Unsupported features](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/plan-your-adoption-of-in-memory-oltp-features-in-sql-server?view=sql-server-ver15#b-unsupported-features)
  + Database Snapshots are not supported
  + Cross database Transactions are not supported
  + Readpast table hint not supported
  + Rowversion, Sequence are not supported
  + [Details on additional In-memory planning and consideration](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/plan-your-adoption-of-in-memory-oltp-features-in-sql-server?view=sql-server-ver15)s

**Schema Durability**

* Schema Only
  + Durability SCHEMA\_ONLY
  + Deploy if you want to maintain only the schema of the in-memory table
  + Key to remember is this data is not maintained after a service restart or database goes offline
  + Use case examples
    - get around the use of temp tables
    - Store temporary data for ETL processes / staging tables for data warehouses
    - Logging tables
* Schema and Data
  + Durability SCHEMA\_AND\_DATA
  + Primary Key is required for Durable In-memory table
  + Deploy if your requirement is to maintain the schema and data of the in-memory table
  + Note this will impact/increase recovery time of the database ( recovery recommendations found in backup and recovery section )
  + Use case examples
    - Tables that require high throughput and low latency
* Starting in 2016 automatic update of statistics is supported for in-memory tables

**Natively compiled Stored Procedures**

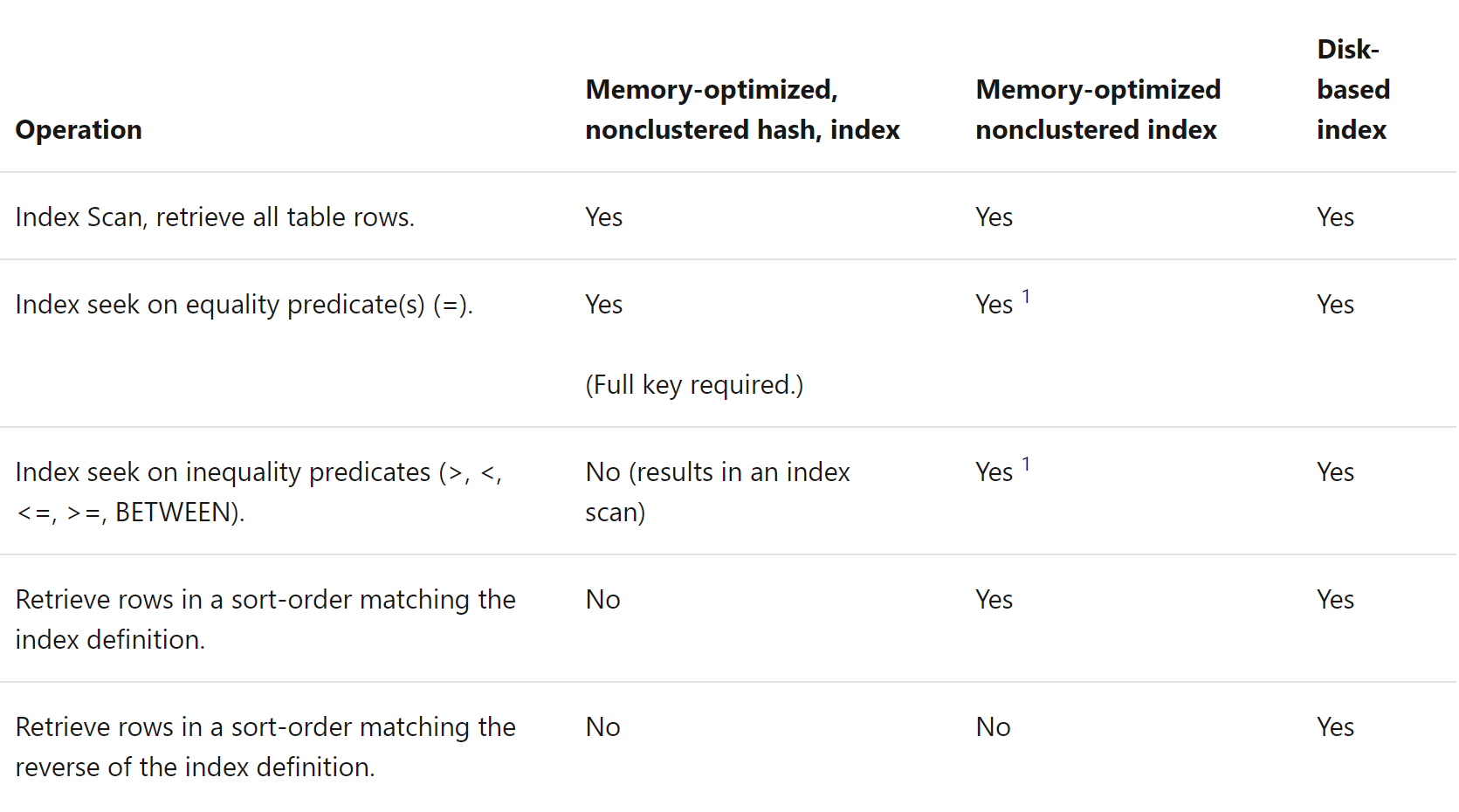
* Used for business-critical parts of application and frequently executed
* Frequently executed
* Compiled into machine code and stored as a DLL
* All parameters passed are considered to have UNKNON values ( uses statistical data for optimization )
  + Parameter sniffing is not used
  + Natively compiled stored procedures do support the [optimized for](https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver15) hint
  + Recommendation is to start with default optimization of unknown
* Natively compiled stored procs are not recompiled when statistics change
* The table should contain a representative set of data and statistics before the procedure Is created. [Guidelines for deploying tables and procedures](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/statistics-for-memory-optimized-tables?view=sql-server-ver15#guidelines-for-deploying-tables-and-procedures).
* In general, one can expect better performance from native compiled stored procs
  + As the number of rows increase
  + Use of the following should exhibit better performance
    - Aggregation, Nested-loop Joins
    - Multi-statement selects or DML operations
    - Complex expressions, procedural logic
* For the full list of [supported constructs](Supported%20Constructs%20in%20Natively%20Compiled%20Stored%20Procedures) in natively compiled stored procedures
* For the list of [constructs not supported](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/transact-sql-constructs-not-supported-by-in-memory-oltp?view=sql-server-2014&redirectedfrom=MSDN#natively-compiled-stored-procedures) by In-memory OLTP

**Indexing**

* Start with nonclustered / ranged indexes then examine Hash Indexes for seeks
* Exist in Memory Only and are not persisted to disk

\* Exception to this are cluster columnstore indexes, to speed up databases recovery they are stored in checkpoint files

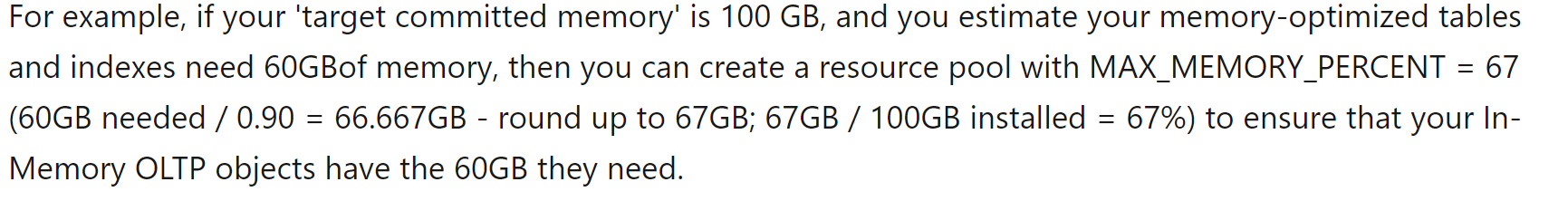
* Memory-optimized tables must have at least one index
* Primary Key is required for Durable In-memory table
* Are inherently covering, includes all columns and are pointers contain a porting to the row in the table data structure
* Fragmentation and fillfactor do not apply to in-memory indexes
* Nonclustered hash indexes can yield performance benefits for point lookups
  + Always fixed sizes
  + Values returned are not sorted
* Nonclustered indexes can yield performance benefits for range and ordered scans
* Index count considerations
  + Up to SQL Server 2016 Memory-optimized table could have up to 8 indexes. SQL Server 2017 and above has removed this constraint.
  + Prior to SQL Server 2016 Indexes need to be added at the time of the table creation
  + Apply only the indexes if they will be frequently used
  + Old row versions on rarely used indexes can impact garbage collection performance
* Performance can be impact if index key has many duplicate values. Performance can be improved by adding an additional column to the nonclustered index
* [**Operations on memory-optimized and disk-based indexes**](https://docs.microsoft.com/en-us/sql/database-engine/guidelines-for-using-indexes-on-memory-optimized-tables?view=sql-server-2014%23operations-on-memory-optimized-and-disk-based-indexes)

****

* [Full guidelines for indexes on Memory-Optimized Tables](https://docs.microsoft.com/en-us/sql/database-engine/guidelines-for-using-indexes-on-memory-optimized-tables?view=sql-server-2014%23operations-on-memory-optimized-and-disk-based-indexes)

**Memory Management**

* Safe recommendation is to start with 2 times the memory
  + Based off expected size of memory optimized tables and indexes
  + Monitoring memory requirements for in-memory objects
* Memory-optimized size corresponds to size, data, and some overhead for row headers
* In-memory indexes tend to be smaller than disk-based indexes
  + Nonclustered index size is in order of the [primary key size] \* [rowcount]
  + Hash indexes are [bucket count] \* 8 bytes
* Bind database with in-memory tables to a [resource pool](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/bind-a-database-with-memory-optimized-tables-to-a-resource-pool?view=sql-server-ver15) ( Requires Enterprise Edition )
  + Prevents memory pool buffer pressure that could affect disk-based table performance
  + Must estimate how much physical memory required for in-memory tables
  + Identify [max memory percent](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/bind-a-database-with-memory-optimized-tables-to-a-resource-pool?view=sql-server-ver15#bkmk_PercentAvailable) for the resource pool which the database containing the in-memory tables is bound
  + Provided is an example from the link provided above



* + sys.dm\_resource\_governor\_resource\_pools will help provide accurate monitoring of the memory optimized tables if bound to a resource pool.
* Detailed guidance around [estimate requirements for memory-optimized tables](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/estimate-memory-requirements-for-memory-optimized-tables?view=sql-server-ver15#basic-guidance-for-estimating-memory-requirements)

**Performance Considerations**

* Plan and Optimize your Hash Index
  + If you are expecting single value lookups
  + If include more than one column in the key equality comparison must include all columns in the key
  + Must define a bucket count
    - OK to default high, but don’t overestimate
    - Higher cardinality is a better candidate for a Hash Index
    - Lower cardinality will create row chains which could lead to performance issues
  + High confidence in the number of rows allows good estimated bucket count
    - Bucket count for the primary keys is total number of rows
    - Bucket count for non-unique and composite hash index is the total distinct keys
  + Correct [bucket count](https://docs.microsoft.com/en-us/sql/database-engine/determining-the-correct-bucket-count-for-hash-indexes?view=sql-server-2014) for hash indexes
* Storage considerations for In-memory OLTP
  + Ensure that transaction logs can support additional IOPs for memory-optimized tables
  + Each container should be typically mapped to its own storage device
  + Ensure storage device can support sequential IOPs up to 3 times the sustained transaction log throughput
  + Ensure storage supports recovery time for durable in-memory tables
  + Maintain equal distribution of checkpoint files
    - SQL Server 2014 – odd number of containers is required
    - SQL Server 2016 and above – odd and even number of leads to uniform distribution
  + Deeper insight into [In-Memory Storage Configuration](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/configuring-storage-for-memory-optimized-tables?view=sql-server-ver15)
* Process to improve load times during recovery to improve Recovery Time Objective
  + Create delta map filters
    - Creating multiple containers could help load performance
    - One thread per container reads the data files and creates a delta map filter
  + Streaming of the data files
    - Data files are read by as many threads as logical CPUs
    - This process could become CPU bound during recovery
  + Advantage for Always On Availability groups modify memory-optimized data on the secondary replica. In the event of a failover data does not need to be re-streamed
  + Failover Cluster Instances must restream memory-optimized data on a failover
  + [Process for improving load times](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/restore-and-recovery-of-memory-optimized-tables?view=sql-server-ver15%23process-for-improving-load-time)
* Garbage collection has issues with memory-optimized table variables. Row versions will only get cleaned out when the variable goes out of scope.

**Backup and Recovery**

* In-Memory filegroups can’t currently be deleted in any version of SQL Server. Defining a well-defined Recovery Point Objective(RPO) is key to preventing data loss in the event the database is marked suspect due to In-Memory objects. It is important to note that a well-defined RPO should be part of any database deployment.
* Database recovery is like disk-based tables, however memory-optimized tables must be loaded into memory before database is available
* In-memory tables adds an additional step to database recovery redo phase
  + Data from data and delta pairs are loaded into memory
  + Data is then updated with active transaction log from last durable checkpoint
  + Factors that could impact the load times
    - Amount of data
    - Sequential I/O bandwidth
    - Degree of parallelism
    - Log records in the active portion of the log
* Database recovery will fail and become suspect if there is not enough memory
* Considering adding containers to improve database recovery
  + Having delta maps can improve performance when reading data files and creating delta map filters
  + Data file thread reads will use as many logical CPUs available which could become CPU bound
* [Database recovery with In-Memory Tables](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/restore-and-recovery-of-memory-optimized-tables?view=sql-server-ver15)

**Monitoring**

* Monitor In-Memory OLTP memory usage
  + Best practice is to bind the database with in-memory optimized tables to a resource pool
  + SSMS has provides a standard report to look at real time
  + Programmatically track in-memory usage by using [DMVs](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/monitor-and-troubleshoot-memory-usage?view=sql-server-ver15#bkmk_UsingDMVs)
    - sys.dm\_db\_xtp\_table\_memory\_stats - consumption for all user tables, indexes, and system objects
    - sys.dm\_os\_memory\_clerks – monitor all MEMORYCLERK\_XTP account across the In-Memory OLTP engine
  + Track In-Memory usage historically using the DMVs
    - Allowing for postmortem analysis if performance issues were experienced
    - Track In-Memory OLTP growth ensuring the resource pool has enough memory allocated preventing outages
* XEvent natively\_compiled\_proc\_slow\_parameter\_passing can be used to find native compiled stored proc inefficiencies
  + Mismatched types: reason=parameter\_conversion
  + Named parameters: reason=named\_parameters
  + DEFAULT values: reason=default
* Monitor disk space allocated to the In-Memory filegroup