Microsoft

Data Warehouse Fast Track

Reference Guide for SQL Server 2017

SQL Technical Article

December 2017, Document version 6.0

Authors
Jamie Reding
Henk van der Valk
Ralph Kemperdick

Contributors
Sadashivan Krishnamurthy
Sunil Agarwal
Matt Goswell

Contents

Copyright

Summary

This paper defines a reference architecture model known as Data Warehouse Fast Track, which uses a resource-balanced approach to implement a symmetric multiprocessor (SMP)–based SQL Server database system architecture with proven performance and scalability for data warehouse workloads.

The goal of a Data Warehouse Fast Track (DWFT) reference architecture is to help enterprises achieve an efficient resource balance between SQL Server data processing capability and realized component hardware throughput. Despite advances using in-memory technologies and the advent of solid-state drives (SSDs), storage I/O remains the slowest operation in a computing environment. The DWFT reference architecture compensates by offloading performance onto the storage subsystem to deliver a balance that works for data warehouse operations. As a result, enterprises can deploy a data warehouse with SQL Server that is optimized for data warehouse operations.

Enterprises can then also take advantage of additional SQL Server capabilities, which are also referenced in this document.

Introduction

Increasingly, enterprises realize that appropriate use of compute and storage resources can help them more successfully work with today's demanding data-driven solutions. With the rapid shift to memory-centric computing, the ability to balance performance between in-memory search algorithms and disk I/O is more important than ever.

The SQL Server Data Warehouse Fast Track (DWFT) program is designed to provide customers with standard and proven system architectures optimized for a range of enterprise data warehousing needs. DWFT is a joint effort between Microsoft and hardware partners. The goal is to help enterprise customers deploy data warehouse solutions with a recommended hardware configuration appropriate for the requirements of the workload with reduced risk, cost, and complexity.

Enterprises can purchase and build on reference implementations from participating system vendors or leverage the best practice guide provided through the program. The DWFT reference architecture program is continuously being improved to incorporate new SQL Server features and customer feedback.

- This document is for IT planners, architects, database administrators, and business intelligence (BI) users interested in standard, proven system architectures for DWFT-conforming SQL Server workloads running on-premises or in Azure.

- This document defines DWFT component architecture and methodology. The result is a set of SQL Server database system architectures and configurations—including software and hardware—required to achieve and maintain a set of baseline performance levels out-of-box for a range of data warehousing workloads.

When enterprises use the DWFT program to set up a data warehouse built on SQL Server, they lay the foundation for a complete Data Management Platform for Analytics. They can then take advantage of newer SQL Server features, including in-memory columnstore technologies that improve the performance of transactional and analytics workloads, as well as the ability of SQL Server to run on either Windows or Linux. They also gain support for both traditional structured relational data and for unstructured big data, such as Internet of Things (IoT) data stored in Hadoop, Spark, or an Azure Data Lake, all the while being able to query the data in languages such as T-SQL, Java, C/C++, C#/VB.NET, PHP, Node.js, Python and Ruby.

DWFT in-memory compression enhancements available in SQL Server

New hardware makes it more affordable to run SQL Server in memory, taking advantage of performance enhancements from memory-optimized tables and high in-memory data compression of columnstore technology.

The Data Warehouse Fast Track program, which uses randomly generated data for testing, shows 1:3.5 data compression. This ratio is what you can expect with randomly generated data, rowstore to columnstore.

In the field, however, you can expect better: The compression ratio with real customer data is likely to be closer to a 1:10 or even a 1:20 ratio. This means that 1 TB of uncompressed data, mapped into clustered columnstore tables, would consume approximately 50 to 100 GB. Typical query response time from in-memory columnstore data would be measured in milliseconds, rather than minutes or even hours.

By using PolyBase, a feature in SQL Server optimized for data warehouse workloads, enterprise customers can also merge big data into the SQL Server universe. PolyBase provides the ability to query both relational data and unstructured data, joining it together into a single result set without moving the data. The ability to connect disparate data sources that straddle on-premises and cloud means cost-effective, business-changing solutions are now possible.

SQL Server and its Microsoft Azure counterpart, Azure SQL Database, are part of a paradigm shift in the world of analytics. Traditionally, data was delivered to analytics routines and it was difficult to bring huge datasets to data scientists for analysis. Today, with Azure Machine Learning Studio, analytics are brought to the data. The ability to extract meaningful and actionable information from huge volumes of data results in more powerful operationalization of analytics models in applications and enables more companies to make sense of big data assets.

As with any database system, the physical I/O speed of the storage subsystem is typically the slowest performance component, even for servers using high-performance SSDs. For those reference architectures which rely on operational storage subsystems, if the storage subsystem isn't optimized, SQL Server cannot perform well. The DWFT reference architecture offloads performance onto the storage subsystem by constraining memory use, which helps ensure a good starting point for conventional operations. Anticipated in the near future will be reference architectures which rely on Persistent Memory, systems configured to use high speed memory as disks, nearly eliminating the need to use the storage subsystem in normal operations.

- After the system is in place, customers can modify, expand, and extend as needed, adding more memory, more storage, and more processing power as they begin to use the apps that are included with SQL Server—including SQL Server Analysis Services, SQL Server Integration Services, SQL Server Reporting Services, and Master Data Services.

- Enterprises can make further adjustments. Increasing the RAM on the server increases the amount of data that can be stored in memory, which takes the pressure off the storage subsystem. More RAM means more in-memory capability and more capability for hybrid transaction/analytical processing (HTAP), also known as real-time operational analytics. Memory access speed also increases as RAM sizes grow. At the far end of this spectrum, Persistent Memory systems keep all the data in memory at all times, writing to the storage only for backups and in case the system loses power.

Thus, enterprises can deploy a data warehouse with SQL Server that is validated to DWFT program standards and sets them up to take

advantage of additional Microsoft Data Management Platform capabilities, including Azure Analysis Services.

Data Warehouse Fast Track program

The DWFT program identifies component hardware configurations that conform to the principles of the DWFT reference architecture. Each reference architecture configuration is defined by a workload and a core set of configuration, validation, and database best practices. The following are key principles of the Fast Track program:

- Workload-specific benchmarks. System design and configuration are based on real concurrent query workloads.
- Detailed and validated hardware component specifications.
- Component architecture balance between database capability and key hardware resources.

The goal of this program is to achieve an efficient out-of-the-box balance between SQL Server data processing capability and hardware component resources. Ideally, the configuration will include minimum system hardware to satisfy storage and performance requirements for a data warehousing workload.

Value proposition

The following principles provide the foundation of the DWFT program:

- **Faster deployment:** The Fast Track program uses the core capabilities of SQL Server on Windows or Linux servers to deliver a balanced SMP data warehouse with optimized performance.
- **Out-of-the-box offerings:** Data Warehouse Fast Track is designed for data warehousing. Rather than being a one-size-fits-all approach to database configuration, the Data Warehouse Fast Track approach is optimized specifically for a data warehouse workload.
- **Reduced risk:** Predetermined balance across key system components minimizes the risk of overspending for CPU or storage resources that will never be realized at the application level.
- **Predictable out-of-the-box performance.** DWFT configurations are built to capacity that already matches the capabilities of the SQL Server application for a selected system and target workload.
- **Workload-centric approach.** Rather than being a one-size-fits-all approach to database configuration, the DWFT approach is aligned with a specific data warehouse workload.
- **Extensible foundation.** DWFT reference configurations deliver a fast-tracked implementation that enable adjustments based on

changing requirements. Configurations built on Windows Server 2016, for example, can be set up for limitless storage when customers adopt the Storage Spaces Direct converged (or disaggregated) approach.

- **Flexibility and choice:** Top industry hardware and software engineering are at the heart of the Fast Track solution. Choose from multiple partner configurations that are certified for Data Warehouse Fast Track use. Get the advanced capabilities of latest-generation servers and storage.

- **Choice of Platform:** Users can transform data into actionable insights by deploying a SQL Server Fast Track solution on both Linux and Windows, bringing SQL Server to where the data lives.

- **Value:** Fast Track solutions are prebuilt, eliminating the necessity of having to research and test multiple hardware components. Cost per QphH (query per hour, a price/performance measure) drops significantly when Fast Track is deployed with columnstore and in-memory technologies. Organizations can gain immediate value from latest-generation servers and storage provided with Fast Track solutions.

Data warehouse workload challenges

Organizations use data warehouses to aggregate data collected from operational systems and elsewhere and prepare data for analysis. A traditional data warehouse workload consists of:

- Periodic data load from operational data stores/applications, such as sales data and financial trades.

- Complex queries run by business analysts to get insight into the data for better decision making. Such queries aggregate large amounts of data across multiple tables, often running for long durations of time while consuming significant I/O bandwidth. For example, finance, marketing, operations, and research teams require access to large volumes of data. To speed up query performance, the data is pre-aggregated for the efficient execution of commonly occurring query patterns, such as sales data in a region aggregated by product categories.

New challenges face both designers and administrators managing mission-critical data warehouses.

Data growth

As the number of IoT devices increase, the data in data warehouses is growing exponentially. In this environment, it is important to use solutions that provide high data compression without compromising query performance, while reducing storage and I/O bandwidth.

Reducing data latency

Data latency refers to the time required to access data for analytics in a data warehouse. Data load and transformation can be a resource-intensive operation that interferes with the ongoing analytics workload. Typical steps are to export, or extract, the data from operational data stores into a staging table; transform the data; and then import, or load, the data into target tables to run analytics. To minimize the impact on business users, the extract, transform, and load (ETL) process typically takes place during off-peak hours. In today's global economy, however, there are no off-peak hours. Businesses are striving to reduce data latency by making data available for analytics within minutes or seconds of its arrival in operational data stores.

Increasingly, organizations want to remove complex ETL processes from the equation by merging operational data stores with analytical workloads. For example, a power generation and distribution company that monitors power consumption with online metering systems might want to adjust power generation and routing based on changing demand patterns. This requires loading incremental data into the data warehouse efficiently and quickly.

Faster query response

Customers require most complex analytic queries to be able to return results in seconds—to enable interactive data exploration at the speed of thought. To meet these challenges, organizations have resorted to using traditional relational database management system (RDBMS) optimization techniques such as building data warehouse–specific indexes and pre-aggregating data. The maintenance overhead associated with these approaches can overwhelm even generous batch windows. Many queries are ad hoc, which means that pre-aggregated data might not be leveraged for faster query response, despite the increasing need to run ad hoc analytics over large datasets.

Database optimization

The ability to optimize how data is stored and maintained within a system has been a complex task in the past. With modern architecture and a well-balanced system, new heights of efficiency are attainable.

- **Reducing I/O requirements by leveraging in-memory technology and Persistent Memory provides a new level of performance characteristic.** With the drop in RAM price and the near-RAM access speed achieved by SSD-based storage subsystems, in-memory and high-speed I/O capabilities are available at a reasonable price point today. Use of Persistent Memory means that most or all of a database can be placed into memory, creating an in-memory database. The combination of a high-speed infrastructure for database workloads and nearly limitless storage capabilities in the cloud allows new data architectures for operational analytics that have not been possible in the past.

- **Optimizing the I/O subsystem for scan performance vs IOPS**: I/O is a common bottleneck with a data warehouse workload. Traditionally, the solution to this challenge has been to simply add drives. It is not uncommon to see hundreds of disks supporting a relatively small data warehouse to overcome the I/O performance limitations of mapping a seek-based I/O infrastructure to a scan-based workload. This is frequently seen in large shared storage area network (SAN) environments that are traditionally optimized for seek (that is, equality search or short data-range scan). However, when database files and configurations are aligned with an efficient disk scan rather than seek access, the performance of individual disks can be many factors higher. The increase in the resulting per-disk performance reduces the number of disks needed to generate sufficient I/O throughput to satisfy the ability of SQL Server to process data for a given workload.

- **Minimal use of secondary indexes**: Adding non-clustered indexes generally adds performance to equality search lookups and short-range scans. However, nonclustered indexes are not suitable for queries returning large numbers of rows because the resulting increase in random disk seek operations can degrade overall system performance. Additionally, adding nonclustered indexes on a data warehouse table will cause significant data management overhead. This may create risk for the service-level agreement (SLA) and the ability to meet database load and maintenance windows. In contrast, sequential scan rates can be many factors faster (10x or more) than random access rates. A system that minimizes the use of random seeks in favor of large scans typically sees much higher average sustained I/O rates. This equates to more efficient use of storage I/O resources and more predictable performance for large scan-type queries.

- **Partitioning:** Separating data into segments, or partitions, is a common optimization pattern with data warehouses. For example, sales data can be partitioned with monthly or quarterly granularity, while customers can be partitioned across regions. Partitioning enables faster query response by reducing the dataset queried.

- **Data compression:** Compression can reduce storage and I/O bandwidth requirements significantly. The compressed clustered columnstore index technology, first introduced in SQL Server 2014, is designed to address the challenges posed by storage and I/O bandwidth in data warehouses. The current DWFT program, which is based on the TCP-H ad-hoc decision support benchmark, uses randomly generated data for testing. Randomly generated data will have a far lower compression ratio than real data. Real customer data typically delivers at least a 1:10 compression ratio (whereas the randomly generated data used for benchmarking gives much less compression benefits, in the 1:3.5 range).

- **Data growth:** The clustered columnstore index provides superior data compression of 10x or more and reduces the I/O, because only columns referenced in the query need to be brought into memory.

- **Reducing data latency**: The combination of memory-optimized tables for online transaction processing (OLTP) or staging types of workloads allows significant reduction in latency for data acquisition. Adding clustered columnstore via an automated temporal logic routine that was introduced in SQL Server 2016, allows data to be aged out of "hot," in-memory storage to a highly compressed data format that the clustered column index provides, significantly reducing overall maintenance by pushing the management of the paired tables to SQL Server. This approach significantly reduces the need for additional non-clustered indexes. The clustered column index in SQL Server simplifies the overhead on data loads, allowing customers to manage data and removing the maintenance windows which were needed in the past and might be the only index needed.

- **Query performance:** Getting data back from the database after the data is stored becomes, with the combination of memory-optimized tables and a clustered columnstore index, a matter of nanoseconds. Because the memory-optimized table is designed to take on the most challenging OLTP workloads in combination with an underlying clustered columnstore index, this provides the best of both worlds. Inserts, updates, and deletes hit the memory-optimized table, and the processing is done completely in memory. Aging data travels into the compressed columnstore table, which provides the capability to scan very large datasets by using partition elimination—that is, by reducing the need to read data that's outside the boundaries of the filter condition of a query. Also, features like predicate pushdown and batch mode processing within the SQL Server engine allow for queries to be significantly speeded up in this context. All of this makes executing ad hoc, complex analytical queries very efficient.

- **Cloud-scale storage:** DWFT reference architectures that include Windows Server also enable a broad selection of newer storage technologies. By choosing to implement the Storage Spaces Direct feature of Windows Server, for example, enterprises can scale storage and compute components independently to eliminate network connectivity bottlenecks and support massive scale out.

Reference implementations

A Data Warehouse Fast Track reference configuration is a specific combination of processors, memory, I/O media, and overall I/O bandwidth that has been benchmarked and certified to be sufficient to host a data warehouse of a specific size. The reference implementation is configured and benchmarked by a DWFT program partner using the DWFT benchmark kit. The DWFT benchmark kit is an executable that

helps partners create the test database, generate the test data, and then execute a series of multiple-stream query tests. These multiple concurrent workloads are designed to identify bottlenecks and establish key system performance metrics. The partner works closely with the Microsoft DWFT engineering team to ensure that they are using optimal configurations for the reference implementation. After the partner obtains the benchmark results, the Fast Track engineering team will do a final performance validation and certify the implementation.

The DWFT benchmark kit, at Version 5.4.5345 as of this writing, is derived from the TPC-H benchmark[1]. There are fundamental differences between a DWFT implementation and a standard TPC-H benchmark.

The DWFT benchmark kit uses a concurrent query model to better represent potential customer environments executing on the reference implementation. A full benchmark execution consists of two primary packages. A package is a group of preset SQL data warehouse queries. The DWFT benchmark kit contains an I/O saturation package and a CPU saturation package. Each of the packages runs with 12, 24, and 48 concurrent streams of queries against the rowstore, and then—in a second run—against a clustered columnstore configuration.

A full benchmark consists of a minimum of 12 performance runs, each running for one hour. Each stream is associated with one type of query. A stream consists of either simple, average, or complex queries. The ratio of the streams in a performance run is 1:2:3, meaning one stream running complex queries, two streams running average queries, and three streams running simple queries. For example, a 24-stream performance run will have 4 streams submitting complex queries, 8 streams submitting average queries, and 12 streams submitting simple queries. Performance data is collected and validated by the Microsoft DWFT engineering team.

Benchmark environment

For ease of benchmarking, the DWFT benchmark kit constrains database size and SQL Server maximum memory based on the number of sockets on the reference implementation under test. The intent is to purposely avoid pulling data into memory, thus forcing SQL Server to put extreme pressure on storage. Performance is off-loaded onto the storage subsystem, because if the storage subsystem is faulty, SQL Server will never perform optimally. The following table shows the testing constraints, which enable partners to produce benchmark results in a timely fashion and provide data to characterize the reference implementation and produce the certification metrics.

[1] The Microsoft DWFT workload is derived from the TPC-H benchmark and as such is not comparable with published TPC-H results.

Testing constraints

Number of Sockets	Target Database Size	SQL Server Memory
1	1 TB	118 GB
2	1 TB	118 GB
4	2 TB	236 GB
8	4 TB	472 GB

DWFT metrics

The certification process depends on the calculation of several primary and secondary metrics published with a reference implementation. See Appendix A for a sample certification template. Primary metrics are listed in the following table; secondary metrics in Appendix B.

Primary Metric	Description

Rated User Data Capacity (TB)	Calculated value based on the Row Store Relative Throughput, the Columnstore Relative Throughput, available storage, and the physical system memory. The calculations assume a compression ratio of 5:1.
	
	The value is the minimum of these computed values, rounded down to the nearest multiple of 10 for ratings over 100, a multiple of 5 for ratings between 40 and 100, and a multiple of 2 for ratings below 40.
	• Maximum User Data Capacity (TB) excluding the recommended free space
	• Memory-based limits
	  ○ One socket: 10 GB per terabyte of data
	  ○ Two sockets: 12 GB per terabyte of data
	  ○ Four sockets: 15 GB per terabyte of data
	  ○ Eight sockets: 18 GB per terabyte of data
	• Throughput-based limits factoring in rowstore and columnstore performance relative to the reference configuration
	(Row Store Relative Throughput + Columnstore Relative Throughput) × 0.125TB

Maximum User Data Capacity (TB)	Calculated value based on the total disk capacity of all disks allocated to primary data storage and assumes a compression ratio of 5:1. This metric is not limited by the relative throughput metrics.
Row Store Relative Throughput	Calculated as a ratio of the rowstore throughput to the overall maximum IO throughput of the DWFT reference configuration. The DWFT reference configuration is a representative two-socket system, which has a Rated User Data Capacity of 25 TB based on Fast Track Data Warehouse methodology. *
Columnstore Relative Throughput	Calculated as a ratio of the columnstore throughput to the rowstore throughput of the DWFT reference configuration *

** See Appendix-A for reference system metrics data*

Choosing a DWFT reference configuration

Expected DWFT compression ratio

Before you can select a DWFT reference configuration, you must have a good understanding of your current data warehouse environment. This includes the current size of your data warehouse and a sense of its future growth and the types of queries you generally run. Once you have quantified your environment, you can review and evaluate one or more of the published DWFT reference implementations that meet your current and future data warehouse environment.

To take full advantage of the information provided by the DWFT reference configurations, you must take into consideration some of the runtime environmental factors. For example, the DWFT compression ratio is set at 5:1, but your data may not provide the same level of compression. If you are not achieving 5:1 compression, you must verify that the reference configuration you are reviewing contains enough storage to accommodate your data. One way to determine this is by comparing the Rated User Data Capacity and the Max User Data Capacity. The difference between these two metrics indicates the "excess storage" that may be used to accommodate your data.

This DWFT process allows for a streamlined approach to choosing a database component architecture that ensures better out-of-the-box balance for your environment. This does assume a moderate degree of expertise in database system architecture and data warehouse deployment. DWFT partners and Microsoft technical sales resources should be involved during this process.

Example: Using the DWFT metrics to compare configurations

Using the three hypothetical DWFT reference configurations that follow, you can make several assumptions and decisions.

Example reference configurations

Configuration	Rated Capacity	Max User Data Capacity	Relative Row Store Throughput	Relative Columnstore Throughput
A	50 TB	80 TB	150	250
B	50 TB	55 TB	200	200
C	50 TB	55 TB	400	400

For this example, assume that Configurations A and B are similarly priced and Configuration C is more expensive. If your compression ratio is 3:1, versus 5:1, and you have approximately 40 TB of user data, then Configuration A would be a better option than Configuration B. Configuration A would easily accommodate your current data and allow for future growth. It would also provide better throughput for your columnstore operations.

If data capacity is not an issue and your solution is based on a rowstore configuration, then Configuration B would be a better option. It would provide higher performance for the rowstore throughput and accommodate up to 55 TB of user data. Although Configuration C would deliver a much higher rowstore or columnstore throughput, that level of performance would come with a price premium.

DWFT best practices

SQL Server provides multiple data compression technologies to meet your data warehouse challenges: page compression, memory-optimized tables, and columnstore indexes. These technologies can be combined to deliver significant data compression and superior query performance. This section provides an overview of best practices to meet data warehouse challenges, including use of data compression technologies, the Data Governor feature in SQL Server, and handling statistics. It is not intended to be a comprehensive guide.

Data loading

Loading large amounts of data requires advance consideration. Parallelization seems to be the ultimate approach in minimizing the time factor. This article discusses the approach within SQL Server. The improvements of Integration Services in conjunction with clustered columnstore indexes are available by setting the AutoAdjustBufferSize parameter setting to **True**. Microsoft also offers some performance considerations for data loading into clustered columnstore indexes.

Page compression

Introduced in SQL Server 2008, page compression allows data warehouse workloads to achieve a typical data compression of 2x to 5x while keeping the data in row structures (rowstores).

- Microsoft recommends page compression for tables, especially for small dimension tables (fewer than a few million rows) that require equality or limited range scans.

- Microsoft recommends using a clustered columnstore index when you create larger tables, such as fact tables.

Memory-optimized tables

In-Memory OLTP is the premier technology available in SQL Server and Azure SQL Database for optimizing OLTP transaction processing, data ingestion, data load, and transient data scenarios. Introduced in SQL Server 2014, In-Memory OLTP can provide great performance gains for the right workloads. It can be adapted and used in the data warehouse operational data store for ingesting data and to speed data throughput in the staging and ETL processes.

Although customers have seen up to a 30x performance gain in some cases, the real gain depends on the workload. In-Memory OLTP improves the performance of transaction processing by making data access and transaction execution more efficient and by removing "lock and latch"[2] contention between concurrently executing transactions. It is not fast because it is in-memory; it is fast because it is optimized around the data being in-memory. Data storage, access, and processing algorithms were redesigned from the ground up to take advantage of the latest enhancements in in-memory and high concurrency computing.

Data stored as memory-optimized tables is guaranteed to have atomicity, consistency, isolation, and durability (ACID) properties. By default, all transactions are fully durable, meaning that there is the same no-data-loss guarantee as for any other table in SQL Server. As part of transaction commit, all changes are written to the transaction log on disk. If there is a failure at any time after the transaction commits, the data is there when the database comes back online. In addition, In-Memory OLTP works with all high availability and disaster recovery capabilities of SQL Server, such as Always On availability groups, backup, and restore.

So how does a technology that was initially created for OLTP map to a data warehouse? In most data warehouse implementations, data staging and ETL are a big part of the process. Data from external sources, including transactional databases and data streams from the Internet of Things, flows into the data staging area, is processed, and finally pushed into the data warehouse.

Introduced with SQL Server 2016, a columnstore index can be created on top of an in-memory table. Using memory-optimized tables with the addition of a columnstore index allows staging and ETL systems to be scaled in such a way that existing facilities can handle many times the amount of data throughput (10x or even 20x) than they otherwise could.

In the staging area, memory-optimized tables used as the target repository for data pulled from the transactional system benefit from the features inherent in an in-memory table.

- **Nondurable tables** are used for transient data, either for caching or for intermediate result sets (replacing traditional temp tables). A *nondurable table* is a memory-optimized table that is declared by using DURABILITY=SCHEMA_ONLY, meaning that changes to these tables do not incur any I/O. This avoids consuming log I/O resources for cases where durability is not a concern.

[2] Latches are internal to the SQL engine and are used to provide memory consistency, whereas locks are used by SQL Server to provide logical transactional consistency.

- **Memory-optimized table types** are used for table-valued parameters (TVPs), as well as intermediate result sets in stored procedures. These can be used instead of traditional table types. Table variables and TVPs that are declared by using a memory-optimized table type inherit the benefits of nondurable memory-optimized tables: efficient data access and no I/O.

- **Natively compiled Transact-SQL modules** are used to further reduce the time taken for an individual transaction by reducing CPU cycles required to process the operations. A Transact-SQL module can be declared to be natively compiled at create time. Stored procedures, triggers, and scalar user-defined functions can be natively compiled.

These features help ensure that data can be quickly and efficiently processed, and then piped into the data warehouse.

At present, memory-optimized tables are a supplement to piping the data from the source systems directly into the data warehouse with the clustered columnstore indexes configured. No benchmarks have been established, nor have any tests been run to calculate efficiencies.

Read more about In-Memory OLTP:

- In-Memory OLTP (In-Memory Optimization)
- In-Memory OLTP in Azure SQL Database
- SQL Database options and performance: Understand what's available in each service tier: Single database service tiers and performance levels

Columnstore

Columnstore indexes store data in columnar structures. By storing data in columns rather than rows, SQL Server can more precisely access the data it needs to answer a query, resulting in increased query performance compared to scanning and discarding unwanted data in rows. Columnstore data can be compressed much more efficiently than rowstore data, often reaching 10x compression while speeding up batch-mode query performance. This technology, which was introduced in SQL Server 2012 with nonclustered columnstore indexes, is significantly improved with the introduction of the updatable clustered columnstore index.

This section focuses on columnstore index best practices. Some guidance will be given for the use of nonclustered columnstore indexes, because in some cases it may not be possible to move an entire table to a clustered columnstore.

As mentioned previously, columnstore tables provide significant compression and query performance over row-oriented tables. Typical

compression savings are of the order of 10x. Data is physically stored in columns and managed as a collection of rows called *rowgroups*. Each rowgroup typically contains 100k (102,400) to 1 million (1,048,576) rows, but under some conditions the rowgroup can be smaller. See Clustered Column Store: Factors that impact size of a RowGroup for details. Within each rowgroup, each column of data is stored independently as a column segment. The storage for a column segment leverages the large object storage format. A column segment is a unit of I/O. There is no requirement that all column segments reside in memory. SQL Server loads the column segments on an as-needed basis. The segments are paged out under memory pressure. See Columnstore indexes - overview for an overview of the technology.

Using columnar storage, analytics queries can also be speeded up significantly for two reasons:

- Reduced I/O due to significant data compression. Columns often have similar data, which results in high compression rates. High compression rates improve query performance by using a smaller in-memory footprint. In turn, query performance can improve because SQL Server can perform more query and data operations in-memory.

- Reduced I/O due to the fact that only columns referenced by the query need to be brought into memory. This is inherent in columnar storage because each column is stored separately.

A new query execution mechanism called batch mode execution has been added to SQL Server. Batch mode execution reduces CPU usage significantly. It is closely integrated with, and optimized around, the columnstore storage format. Batch mode execution provides 3x to 4x speedup of query execution. You can read more about batch mode execution at Columnstore Index Performance: Batch Mode Execution.

Clustered columnstore index

SQL Server 2014 introduced the updateable clustered columnstore index. Columnstore indexes can provide 10x query performance gains over traditional row-oriented storage and up to 10x data compression over the uncompressed data size. Subsequent versions of SQL Server have built on the characteristics of the SQL Server 2014 clustered columnstore index:

- Introduced with SQL Server 2017, a clustered columnstore index now supports non-persisted computed columns. Persisted columns are not supported, and you cannot create a nonclustered index on a columnstore index that has a computed column on it.

- A clustered columnstore index can now have one or more nonclustered b-tree rowstore indexes built on it. Previously, the columnstore index did not support nonclustered indexes. SQL Server

automatically maintains the nonclustered indexes for data manipulation language (DML) operations.

  - Note: MERGE is disabled when a b-tree index is defined on a clustered columnstore index.

- There is now support for primary keys and foreign keys by using a b-tree index to enforce these constraints on a clustered columnstore index.

  - Clustered columnstore indexes support read committed snapshot isolation (RCSI) level and snapshot isolation (SI). This enables transactional consistent analytics queries with no locks.

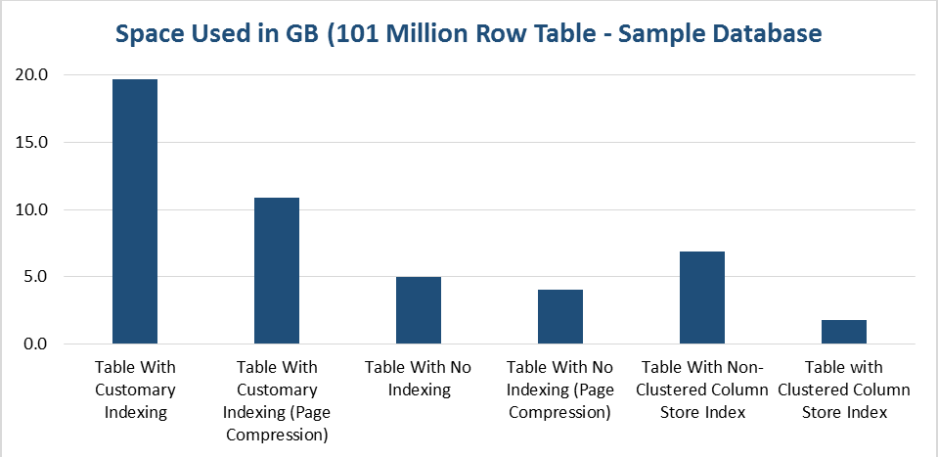
  - Columnstore supports index defragmentation by removing deleted rows without the need to explicitly rebuild the index. The ALTER INDEX (Transact-SQL) REORGANIZE operation will remove deleted rows—based on an internally defined policy—from the columnstore as an online operation.

  - Columnstore indexes can access an Always On readable secondary replica. Performance for operational analytics can be improved by offloading analytics queries to an Always On secondary replica.

  - Aggregate pushdown is supported with or without the Group By clause for both clustered columnstore indexes and nonclustered columnstore indexes.

- Columnstore can be used as the primary storage method for the entire table. This enables great data compression savings as shown in the following graphic. A table with page compression applied produces approximately 2x storage savings. A table with a nonclustered columnstore index takes less space than data that is page compressed, because indexes other than nonclustered columnstore index are not needed for data warehouse queries. However, SQL Server still requires the underlying table be saved in a rowstore storage format, even for the nonclustered columnstore index structure. The best storage savings will be achieved with the clustered columnstore index. Compression is even greater with a clustered columnstore index because there is no rowstore required, as shown in the last bar in the graphic.

Example results by compression type



The clustered columnstore index is best suited for tables that have large numbers (think millions) of rows, such as FACT tables. If there are fewer than 1 million rows, we don't recommend using a columnstore index. Columnstore compression and performance may be affected by a having a small number of rows in each column segment. See Columnstore indexes – what's new for additional details about columnstore index technology.

Nonclustered columnstore index

Available since SQL Server 2012, the nonclustered columnstore index can be leveraged as an additional index on a table to cover specific columns. Significant enhancements have been made to the nonclustered columnstore index since its introduction:

- A rowstore table can have one updateable nonclustered columnstore index. Previously, the nonclustered columnstore index was read-only.

- The nonclustered columnstore index definition supports using a filtered condition. Use this feature to create a nonclustered columnstore index on only the cold data of an operational workload. By doing this, the performance impact of having a columnstore index on an OLTP table will be minimal.

- An in-memory table can have one columnstore index. You can create it when the table is created or add it later by using ALTER TABLE (Transact-SQL). Previously, only a disk-based table could have a columnstore index. Note the following:

    - For memory-optimized tables, a columnstore index must include all the columns; the columnstore index cannot have a filtered condition.

    - For memory-optimized tables, queries on columnstore indexes run only in InterOP mode, and not in the in-memory native mode. Parallel execution is supported.

- Columnstore indexes offer a compression delay option that minimizes the impact the transactional workload can have on real-time operational analytics. This option allows for frequently changing rows to stabilize before compressing them into the columnstore. For details, see CREATE COLUMNSTORE INDEX (Transact-SQL) and Get started with Columnstore for real time operational analytics.

- Nonclustered columnstore indexes support both read committed snapshot isolation (RCSI) and "simple" snapshot isolation (SI). This enables transactional consistent analytics queries with no locks.

    - Note: SI requires possible changes to the application layer; RCSI does not, but will put extra stress on TempDB.

- Columnstore supports index defragmentation by removing deleted rows without the need to explicitly rebuild the index. The ALTER INDEX (Transact-SQL) REORGANIZE operation will remove deleted rows—based on an internally defined policy—from the columnstore as an online operation.

- To improve performance, SQL Server computes the aggregate functions MIN, MAX, SUM, COUNT, and AVG during table scans when the data type uses no more than eight bytes and is not of a string type. Aggregate pushdown is supported with or without the Group By clause for both clustered columnstore indexes and nonclustered columnstore indexes.

- Predicate pushdown speeds up queries that compare strings of type [v]char or n[v]char. This applies to the common comparison operators and includes operators such as LIKE that use bitmap filters. This works with all collations that SQL Server supports.

- Batch mode execution now supports queries using any of these operations:

    - SORT

    - Aggregates with multiple distinct functions COUNT/COUNT, AVG/SUM, CHECKSUM_AGG, STDEV/STDEVP

    - Window aggregate functions COUNT, COUNT_BIG, SUM, AVG, MIN, MAX, and CLR

    - Window user-defined aggregates CHECKSUM_AGG, STDEV, STDEVP, VAR, VARP, and GROUPING

    - Window aggregate analytic functions LAG< LEAD, FIRST_VALUE, LAST_VALUE, PERCENTILE_CONT, PERCENTILE_DISC, CUME_DIST, and PERCENT_RANK

- Single-threaded queries running under MAXDOP 1 or with a serial query plan execute in batch mode. Previously, only multi-threaded queries ran with batch execution.

- Memory-optimized table queries can have parallel plans in SQL InterOp mode when accessing data in a rowstore or in a columnstore index.

Bulk import

A clustered columnstore index allows parallel data loads using bulk import commands. The bulk import loads the data directly into a compressed rowgroup if the batch size is greater than or equal to 102,400 rows. This minimizes logging overhead, because the directly compressed data does not go through a delta rowgroup for processing.

Smaller groups of rows, such as those that are encountered at the end of a bulk load operation, are written first into compressed delta store tables. When the maximum number of rows for a rowgroup (1 million) is reached, the rowgroup is marked as closed, and a background process compresses the delta rowgroups into columnar storage and moves them into final compressed rowgroup segments. This process is slower than directly loading into a compressed rowgroup because each row of data is touched twice—once for insertion into the delta store table, and again when the data in the delta store table is moved into the compressed rowgroup. See Clustered Column Store Index: Bulk Loading the Data for additional information about bulk-loading data into a columnstore index, and also see Columnstore indexes – overview.

One general best practice is to load the data, if possible, in the order of the column that will be used for filtering rows in the data warehouse query workload. This enables SQL Server to eliminate rowgroups that won't qualify based on the minimum and maximum values kept for each column in the rowgroup.

Parallel or concurrent bulk imports may result in multiple delta rowgroups, because the last batch of each may not have 100k rows. It's important to monitor the number of delta rowgroups, because they can negatively affect the performance of queries. It's a good idea to schedule bulk imports when there is least load on the server, if possible.

Data fragmentation

When a row is updated in a clustered columnstore index, the operation is a DELETE followed by an INSERT. The row is simply marked for deletion but is not actually removed; the updated version of the row is inserted into a delta rowgroup. The rowgroup is compressed and migrated into the columnstore when it contains 1,048,576 rows. The delta rowgroup is then marked as closed. A background process, called the *tuple-mover*, finds each closed rowgroup and compresses it into the columnstore.

Over time, the number of deleted rows in the clustered columnstore can grow to a point where they affect storage efficiency and query performance. To defragment the index and force delta rowgroups into the columnstore, use an ALTER INDEX command to rebuild or reorganize the index. For detailed information about how to defragment a columnstore index, see Columnstore indexes – defragmentation.

Partitioning

The concept of partitioning is the same in a conventional clustered index, a heap, or a columnstore index. Partitioning a table divides the table into smaller groups of rows based on a range of column values. It is often used for managing the data. For example, you could create a partition for each year of data, and then use partition switching to archive data to less expensive storage. Partition switching works on columnstore indexes and makes it easy to move a partition of data to another location.

Rowgroups are always defined within a table partition. When a columnstore index is partitioned, each partition has its own compressed rowgroups and delta rowgroups.

Each partition can have more than one delta rowgroup. When the columnstore index needs to add data to a delta rowgroup and the delta rowgroup is locked, the columnstore index tries to obtain a lock on a different delta rowgroup. If there are no delta rowgroups available, the columnstore index creates a new delta rowgroup. For example, a table with 10 partitions could easily have 20 or more delta rowgroups.

When partitioning columnstore indexes, we recommend deploying a broader range in the partition strategy to increase the number of rows. For example, if there are fewer than a million rows in daily partitions, you may want to consider using weekly, monthly, or yearly partitions. A partition strategy should match data update and delete requirements. Data partitioning may reduce the amount of work and downtime needed for an index rebuild to alleviate a fragmented columnstore index.

For detailed information about columnstore partitioning, see Columnstore indexes – architecture.

Resource Governor

Data warehousing workloads typically include a good proportion of complex queries operating on large volumes of data. These queries often consume large amounts of memory, and they can spill to disk where memory is constrained. This has specific implications in terms of resource management. You can use the Resource Governor technology to manage resource usage. Maximum memory settings are of importance in this context.

In default settings for SQL Server, Resource Governor is providing a maximum of 25 percent of SQL Server memory resources to each

session. This means that, at worst, three queries heavy enough to consume at least 25 percent of available memory will block any other memory-intensive query. In this state, any additional queries that require a large memory grant to run will queue until resources become available.

You can use Resource Governor to reduce the maximum memory consumed per query. However, as a result, concurrent queries that would otherwise consume large amounts of memory utilize **tempdb** instead, introducing more random I/O, which can reduce overall throughput. While it can be beneficial for many data warehouse workloads to limit the amount of system resources available to an individual session, this is best measured through analysis of concurrency benchmarks and workload requirements. For more information about how to use Resource Governor, see Managing SQL Server Workloads with Resource Governor in SQL Server Books Online. Vendor specific guidance and practices for DWFT solutions should also be reviewed. Larger 4-socket and 8-socket DWFT solutions may rely on specific Resource Governor settings to achieve optimal performance.

In summary, there is a trade-off between lowering constraints that offer higher performance for individual queries and more stringent constraints that guarantee the number of queries that can run concurrently. Setting MAXDOP at a value other than the maximum will also have an impact – in effect, it further segments the resources available, and it constrains I/O throughput to an individual query.

Database statistics best practices

- Use the AUTO CREATE and AUTO UPDATE options for statistics (sync is the system default in SQL Server). Use of this technique will minimize the need to run statistics manually.

- If you must gather statistics manually, statistics ideally should be gathered on all columns in a table. If it is not possible to run statistics for all columns, you should at least gather statistics on all columns that are used in a WHERE or HAVING clause and on join keys. Index creation builds statistics on the index key, so you don't have to do that explicitly.

- Composite (multicolumn) statistics are critical for many join scenarios. Fact-dimension joins that involve composite join keys may induce suboptimal nested loop optimization plans in the absence of composite statistics. Auto-statistics will not create, refresh, or replace composite statistics.

- Statistics that involve an increasing key value (such as a date on a fact table) should be updated manually after each incremental load operation. In all other cases, statistics can be updated less frequently. If you determine that the AUTO_UPDATE_STATISTICS option is not sufficient for you, run statistics on a scheduled basis.

Appendix A. DWFT certification template with reference system data

DWFT Certification #xxxx-xxx DWFT Rev. 5.x	DWFT Reference Architecture		Report Date MM/DD/YYYY
System Provider	**System Name**	**Processor Type**	**Memory**
Partner logo	DWFT Reference System	Intel Xeon E7-8890 v4 2.2 GHz (4/96/192)	3072 GB
Operating System		**SQL Server Edition**	
Windows Server 2016		SQL Server 2017 Enterprise Edition	
Storage Provider	**Storage Information**		
Storage Provider Logo	2x 300 GB SAS HDDs for OS (RAID 1) 8x SanDisk ioMemory3 (6.4TB) Enterprise Flash adapters for data and tempdb 4x 960GB SSDs for log (RAID 10)		

Primary Metrics			
Rated User Data Capacity[1] (TB)	Row Store Relative Throughput[2]	Column Store Relative Throughput[3]	Maximum User Data Capacity[1] (TB)
145	459	734	187

Row Store					
Relative Throughput[2]	Measured Throughput (Queries/Hr/TB)	Measured Scan Rate Physical (MB/Sec)	Measured Scan Rate Logical (MB/Sec)	Measured I/O Throughput (MB/Sec)	Measured CPU (Avg.) (%)
459	486	12,200	14,832	13,516	87

Column Store					
Relative Throughput[2]	Measured Throughput (Queries/Hr/TB)	Measured Scan Rate Physical (MB/Sec)	Measured Scan Rate Logical (MB/Sec)	Measured I/O Throughput (MB/Sec)	Measured CPU (Avg.) (%)
734	4,774	2,798	N/A	N/A	88

The reference configuration is a 2 socket system rated for 25TB using the DWFT V4 methodology

[1] Assumes a data compression ratio of 5:1

[2] Percent ratio of the throughput to the row store throughput of the reference configuration.

[3] Percent ratio of the throughput to the column store throughput of the reference configuration.

Reported metrics are based on the qualification configuration which specifies database size and SQL Server memory.

Appendix B. Data Warehouse Fast Track metrics

Metric	Description
Rated User Data Capacity (TB) [Primary Metric]	Calculated value based on the Row Store Relative Throughput, the Columnstore Relative Throughput, available storage and the physical system memory. The calculations assume a compression ratio of 5:1.
Maximum User Data Capacity (TB) [Primary Metric]	Calculated value based on the total disk capacity of all disks allocated to primary data storage and assumes a compression ratio of 5:1. This metric is not limited by the relative throughput metrics.
Row Store Relative Throughput [Primary Metric]	Calculated as a ratio of the Row Store throughput to the Row Store throughput of the DWFT reference configuration.
Columnstore Relative Throughput [Primary Metric]	Calculated as a ratio of the Columnstore throughput to the Columnstore throughput of the DWFT reference configuration.
Row Store Measured Throughput (Queries/Hr/TB)	Calculated from the measured number of rowstore benchmark queries completed during the measurement interval, normalized to a 1-TB database and expressed in Queries/Hr/TB.
Row Store Measured Scan Rate Physical (MB/Sec)	Measured physical I/O reads from disk during the measurement interval of the benchmark and expressed in MB/Sec.
Row Store Measured Scan Rate Logical (MB/Sec)	Measured user query throughput which includes reads from RAM/buffer cache and expressed in MB/Sec.
Row Store Measured I/O Throughput (MB/Sec)	Calculated midpoint of the measured Scan Rate Physical and the measured Scan Rate Logical and expressed in MB/Sec.
Row Store Measured CPU Utilization (%)	Average CPU utilization measured during the measurement interval of the benchmark and expressed in percentage of total CPU.
Columnstore Measured Throughput (Queries/Hr/TB)	Calculated from the measured number of columnstore benchmark queries completed during the measurement interval, normalized to a 1TB database and expressed in Queries/Hr/TB.
Columnstore Measured Scan Rate Physical (MB/Sec)	Measured physical I/O reads from disk during the measurement interval of the benchmark and expressed in MB/Sec.
Columnstore Measured CPU Utilization (%)	Average CPU utilization measured during the measurement interval of the benchmark and expressed in percentage of total CPU.

Appendix C. Data Warehouse Fast Track query examples

The Data Warehouse Fast Track (DWFT) benchmark kit uses queries defined as simple, average, and complex to measure the performance of a system. Examples of these queries are:

Simple Query

```
SELECT  SUM(l_extendedprice * l_discount) AS revenue
FROM    lineitem
WHERE   l_discount BETWEEN 0.04 - 0.01 AND 0.04 + 0.01 AND
l_quantity < 25
```

Average Query

```
SELECT  l_returnflag,
        l_linestatus,
        SUM(l_quantity) AS sum_qty,
        SUM(l_extendedprice) AS sum_base_price,
        SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
        SUM(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
        AVG(l_quantity_ AS avg_qty,
        AVG(l_extendedprice) AS avg_price,
        AVG(l_discount) AS avg_disc,
        COUNT_BIG(*) AS count_order
FROM    lineitem
WHERE   l_shipdate <= DATEADD(dd, -90, '1998-12-01')
GROUP   BY  l_returnflag,
            l_linestatus
ORDER   BY  l_returnflag,
            l_linestatus
```

Complex Query

```
SELECT  100.00 * SUM(CASE
                       WHEN p_type LIKE 'PROMO%'
                       THEN l_extendedprice * (1 - l_discount)
                       ELSE 0
                     END) / SUM(l_extendedprice * (1 - l_discount)) AS
promo_revenue
FROM    lineitem,
        part
WHERE   l_partkey   =  p_partkey AND
        l_shipdate  >= '1995-09-01' AND
        l_shipdate  <  DATEADD(mm, 1, '1995-09-01')
```