SQL Server technical e-book series

Microsoft

SQL Server performance: faster querying with SQL Server

SQL Server performance: faster querying with SQL Server

Content

Who should read this e-book?

This e-book is for database architects, administrators, and developers looking to accelerate query processing capabilities to support their most demanding data-driven applications. By reading this e-book, you'll learn how to get the most out of SQL Server, taking advantage of advanced built-in processing capabilities including in-memory performance, security, analytics, and flexibility. This e-book covers tools and features like columnstore indexes and Adaptive Query Processing, with technical details on how to put these capabilities into practice.

How you handle data can be a difference-maker for your business.

With data factoring in an increasing amount of interactions everywhere, it's important to not only keep up with the volume of data but harness the power of it.

This need presents an opportunity for you to modernize your organization's applications and drive digital transformation with better built-in analytics. By using the most advanced business intelligence capabilities, you can make the most of the vast and varied data out there, accelerate your speed of doing business through smarter decision-making and faster execution, and gain a competitive advantage

Microsoft SQL Server can help you achieve this goal through the best and fastest platform available for your data and applications. SQL Server offers critical built-in capabilities, including:

- Industry-leading[1] in-memory performance
- Trusted security
- Game-changing in-database advanced analytics
- Flexibility to run your complete data estate on any environment with any data

[1] Gartner has rated Microsoft as a leader, with the most complete vision and highest ability to execute of any operational database management system, for three consecutive years. SQL Server Blog, Three years in a row—Microsoft is a leader in the ODBMS Magic Quadrant, November 3, 2017.

SQL Server combines higher speed with greater choice. By bringing the power of SQL Server to Linux, Linux-based containers, and Windows, Microsoft enables you to decide which development languages, data types, environments (on-premises or cloud), and operating systems work best for your unique situation.

IDC estimates that the amount of the global datasphere subject to data analysis will grow by a factor of

50 to 5.2 ZB in 2025.

Source: IDC, Data Age 2025, April 2017

SQL Server delivers built-in capabilities and features that accelerate analytics performance and query processing to keep your database application at peak speed.

SQL Server holds multiple top-performance benchmarks for transaction processing with leading business applications:

Hewlett Packard Enterprise (HPE) announced a new world-record TPC-H 10TB benchmark[1] result using SQL Server 2017 and Windows Server 2016, demonstrating the leadership of SQL Server in price and performance.

HPE also announced the first TPC-H 3TB result[2] and exhibited the power of SQL Server 2017 to handle analytic query workloads, including data warehouses.

SQL Server is a proven leader for online transaction processing (OLTP) workloads. Lenovo recently announced a new world-record TPC-E benchmark result[3] using SQL Server 2017 and Windows Server 2016. This is now the top TPC-E result in both performance and price/performance.

SQL Server holds a world-record 1TB TPC-H benchmark[4] result (non-clustered) for SQL Server on Red Hat Enterprise Linux. ■

[1] 10TB TPC-H non-clustered result as of November 9, 2017.
[2] 3TB TPC-H non-clustered result as of November 9, 2017.
[3] TPC-E benchmark result as of November 9, 2017.
[4] TPC-H benchmark result on RHEL as of April 2017.

Databases perform both simple and complex transactions.

Depending on the types and level of complexity involved in performing those transactions, the amount of time it takes a database to return results can increase significantly. When considering database optimization, it's important to know the types of transactions and the amount of throughput your database will need to sustain to be considered responsive enough.

SQL Server has several features designed to increase transactional throughput. In-memory data processing can save significant time with certain types of transactions. Columnstore indexes leverage column-based data storage to organize large amounts of analytics-ready information in a compressed format. This makes lookups extremely fast compared to B-tree index searches done in row-based data storage.

In-memory data

Broadly speaking, in-memory data processing technology is faster because it saves disk-seek and disk-read times. In-memory data processing also eliminates wait time due to lack of concurrency lockout, but without persistence it's vulnerable to the transient nature of RAM. That is, a sudden power failure can result in data loss.

SQL Server has added functionality to build upon the best of in-memory data processing technology while mitigating the risks. This helps you to safely get more out of In-Memory OLTP, dramatically improving throughput.

Certain types of transactions are ideal candidates for In-Memory OLTP. Situations where transactions are short and plentiful will yield the most performance gain (especially if processing a high percentage of INSERT statements), including sales transaction recording, high-volume Internet of Things (IoT) or remote sensor reports, or ad clicks, just to name a few.

Several features of SQL Server's In-Memory OLTP can be combined to optimize performance depending on the type of data you're processing:

Memory-optimized tables. The key to In-Memory OLTP is the use of memory-optimized tables—a special type of table data structure which is created in memory and not on disk. Memory-optimized tables are said to take an optimistic approach to concurrent transaction processing because they rely on the fact that the chances of two UPDATE transactions affecting the same row of data is very low. Therefore, a row isn't locked before it's updated. Instead, the system performs some quick validation at the time of commit and flags any conflict that may occur, saving precious milliseconds of processing time.

Non-durable tables. In SQL Server, memory-optimized tables are persistent by default although they can be configured to be non-persistent or non-durable if the risk of data loss is acceptable. They are used as temporary storage for query results or for caching.

Memory-optimized table variables. This feature helps you with a variable that is declared as an in-memory table. These variables store query results in such a way that it's easy to pass them to other statements or procedures, such as natively compiled or interpreted stored procedures (the latter being used for disk-based tables).

Natively compiled stored procedures. A stored procedure has been compiled to native code and is able to access memory-optimized tables. Stored procedures are compiled upon creation, which gives them an advantage over interpreted stored procedures since error detection happens upon creation rather than at execution time. Higher performance gains are realized the more complex the logic and the more rows of data a natively compiled stored procedure processes. Read more about best practices around the use of natively compiled stored procedures.

Natively compiled scalar user-defined functions. Also called UDFs, these user-defined functions have been compiled to native code for faster execution on memory-optimized tables. UDFs defined in this way are only able to be executed on memory-optimized tables and not on traditional disk-based tables.

SQL Server improves the performance of In-Memory OLTP workloads by removing many of the limitations on tables and stored procedures found in earlier product versions. Features introduced in 2017 make it easier to migrate your applications and take advantage of the benefits of In-Memory OLTP. Additionally, memory-optimized tables now support even faster OLTP workloads with better throughput as a result of parallelized operations.

SQL Server also offers these benefits:

The limitation of eight indexes for memory-optimized tables has been eliminated. You can now create as many indexes on memory-optimized tables as you can create on disk-based tables. Any disk-based table in your database that you could not migrate previously due to this limitation can now be memory-optimized.

Transaction log redo of memory-optimized tables is now done in parallel. This bolsters faster recovery times and significantly increases the sustained throughput of Always On Availability Group configuration.

Performance of Bw-Tree (non-clustered) index rebuild for `MEMORY_OPTIMIZED` tables during database recovery has been significantly optimized. This improvement substantially reduces database recovery time when non-clustered indexes are used.

`sp_spaceused` is now supported for memory-optimized tables. It displays the number of rows, disk space reserved, and disk space used by a table, indexed view, or Service Broker queue in the current database. Alternatively, it displays the space reserved and used by the entire database.

`sp_rename` is now supported for memory-optimized tables and natively compiled T-SQL modules.

Memory-optimized filegroup files can now be stored on Azure Storage. Backup/restore of memory-optimized files on Azure Storage is supported.

Memory-optimized tables now support computed columns. Query surface area in native modules has been improved to include full support for JSON functions. Additional native support for query constructs, such as `CROSS APPLY`, `CASE`, and `TOP (N) WITH TIES`, is now available.

Dive deeper: In-Memory OLTP

To enable an application to use In-Memory OLTP, you can create a memory-optimized table:

```
 CREATE TABLE SupportEvent
(
    SupportEventId   int NOT NULL
        PRIMARY KEY NONCLUSTERED,
    ...
)      WITH (
        MEMORY_OPTIMIZED = ON,
        DURABILITY = SCHEMA_AND_DATA);
```

The `MEMORY_OPTIMIZED = ON` clause identifies a table as memory-optimized and `SCHEMA_AND_DATA`, specifying that all changes to the table will be logged and the table data is stored in memory. Every single memory-optimized table must contain at least one index.

➔ **For more in-depth information** about memory-optimized tables, see Memory-Optimized Tables.

You can also create natively compiled stored procedures to access data in memory-optimized tables. Here's an example syntax:

```
CREATE PROCEDURE dbo.usp_add_kitchen @dept_id int, @kitchen_
count int NOT NULL
WITH EXECUTE AS OWNER, SCHEMABINDING, NATIVE_COMPILATION
AS
BEGIN ATOMIC WITH (
TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'us_
english')

  UPDATE dbo.Departments
  SET kitchen_count = ISNULL(kitchen_count, 0) +
@kitchen_count
  WHERE id = @dept_id
END;
GO
```

A procedure created without `NATIVE_COMPILATION` cannot be altered to a natively compiled stored procedure.

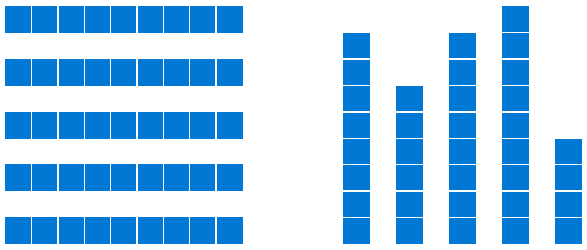
➔ **For a discussion** of programmability in natively compiled stored procedures, supported query surface area, and operators, see Supported Features for Natively Compiled T-SQL Modules.

Columnstore indexes

A columnstore index is a technology for storing and querying large amounts of data in a columnar format. This is one of the most powerful features of SQL Server for high-speed analytic queries and large databases. Columnstore indexes boost performance by compressing columnar data to reduce memory and disk footprint, filtering scans automatically through rowgroup elimination, and processing queries in batches. With SQL Server columnstore indexes, you can enable operational analytics—the ability to run real-time analytics on a transactional workload.

SQL Server offers several capabilities for columnstore indexes, including:

- Non-clustered columnstore index (NCCI) online rebuild.
- Large objects (LOBs) support for columnstore indexes.
- Computed columns for clustered column index (CCI).
- Query optimizer features, like Machine Learning Services.



Data stored as rows Data stored as columns

Dive deeper: Columnstore indexes

A columnstore index is either clustered or non-clustered. A clustered columnstore index (CCI) is the physical storage for the entire table. Use a CCI to store fact tables and large dimension tables for data warehousing workloads. A non-clustered columnstore index (NCCI) is a secondary index created on a rowstore table. Use an NCCI to perform analysis in real time on an OLTP workload. A clustered columnstore index can have one or more non-clustered B-tree indexes.

➔ **To learn more** about CCI and NCCI, go to Columnstore indexes – Overview.

Dive deeper: Columnstore indexes for data warehousing

CCIs are best suited for analytics queries, since analytics queries tend to perform operations on large ranges of values rather than looking up specific values. When you create a table with the `CREATE TABLE` statement, you can designate the table as a CCI by creating a `CLUSTERED COLUMNSTORE INDEX` option:

```
--Create the table
CREATE TABLE t_account (
    AccountKey int NOT NULL,
    AccountDescription nvarchar (50),
    AccountType nvarchar(50),
    UnitSold int
);
GO
--Store the table as a columnstore.
CREATE CLUSTERED COLUMNSTORE INDEX taccount_cci ON t_account;
GO
```

You can create non-clustered B-tree indexes as secondary indexes on a CCI. To optimize table seeks in a data warehouse, you can create an NCCI designed to run queries that perform best with these searches:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX taccount_nc1 ON t_
account (AccountKey);
```

Dive deeper: Columnstore for a hybrid transactional/analytical processing (HTAP)

Hybrid transactional/analytical processing (HTAP) uses a columnstore index on a rowstore table that you're able to update. The columnstore index keeps a copy of the data, so the OLTP and analytics workloads run against isolated copies of the data. This enables real-time analytical processing over transactional data workloads without reduced performance. For each table, drop all B-tree indexes that are chiefly designed to accelerate existing analytics on your OLTP workload. Replace them with a solitary columnstore index. See the below example to create a non-clustered columnstore on the OLTP table with a filtered condition:

```
CREATE TABLE t_account (
    accountkey int PRIMARY KEY,
    accountdescription nvarchar (50),
    accounttype nvarchar(50),
    unitsold int
);
--Create the columnstore index with a filtered condition
CREATE NONCLUSTERED COLUMNSTORE INDEX account_NCCI
ON t_account (accountkey, accountdescription, unitsold)   ;
```

The columnstore index on an in-memory table enables you to use operational analytics by integrating In-Memory OLTP and in-memory columnstore technologies, delivering high performance for both these workloads. The columnstore index on an in-memory table must include all columns. This example creates a memory-optimized table with a columnstore index:

```
CREATE TABLE t_account (
    accountkey int NOT NULL PRIMARY KEY NONCLUSTERED,
    Accountdescription nvarchar (50),
    accounttype nvarchar(50),
    unitsold int,
    INDEX t_account_cci CLUSTERED COLUMNSTORE
)
    WITH (MEMORY_OPTIMIZED =
ON );
GO
```

With this index creation, you can implement HTAP without making any changes to your application. Analytics queries will run against the columnstore index and OLTP operations will keep running against your OLTP B-tree indexes.

Non-clustered columnstore online rebuild

Columnstore indexes are an important component in keeping your queries performant. To maintain performance, these indexes need to be rebuilt periodically which—for very large indexes—can take time. For any online business, taking an application down for an hour means losing real money and possibly even worse. To keep your application up and running, SQL Server supports online backups, consistency checks, and index rebuilds.

In SQL Server, you can pause an index build and resume it at any time, even after a failure. You can rebuild indexes while they are still in use, and you can pause and resume those rebuilds, picking up exactly where the rebuild left off. You can enjoy the benefits of using less log space than the index rebuild operations of previous releases. ■

Poorly written queries can degrade application performance and hinder the availability of critical business information.

They can also lead to inefficient use of resources like CPU, memory, and network. Regressions in query execution plans can also greatly impact performance. These can occur if there have been application changes, stale database statistics, or inaccurate row count estimates. Even if your database server runs on the most powerful hardware available, its performance can be negatively affected by a handful of misbehaving queries. In fact, even one bad query can cause serious performance issues for your database.

Query performance depends on many factors, one of which is the query plan. When tuning and optimizing a poor query, a DBA usually starts by looking at the execution plan of that query and evaluating it to determine the best and most efficient plan based on data estimation. To help this process, SQL Server provides query processing and performance features that change the way query plans work:

Query Store improvements enable you to track wait statistics summary information, helping to reduce the time spent troubleshooting.

Adaptive Query Processing (AQP) is a way of optimizing the SQL Server execution plan by mitigating errors in the query plan and adapting the execution plan based on run results.

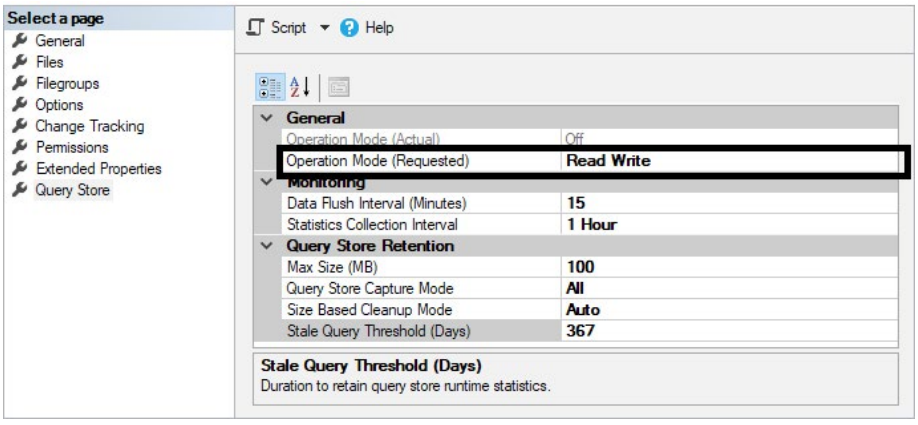
Query Store

Query Store gathers telemetry on compilation-time and execution-time statistics. It also captures query and plan history for your review. Data is separated by time windows, so you can see database usage patterns and understand when query plan changes happened on the server.

Wait statistics are another source of information that help you troubleshoot performance issues in SQL Server. In the past, wait statistics were available only at the instance level, which made it difficult to backtrack details to the actual query. Query Store helps you track wait statistics more efficiently by providing summary information. Wait statistics are tied to a query plan and taken over time, just like runtime statistics. This provides more insight into workload performance and bottlenecks, while preserving key Query Store advantages.

Dive deeper: Using SQL Server Management Studio or Transact-SQL Syntax for Query Store

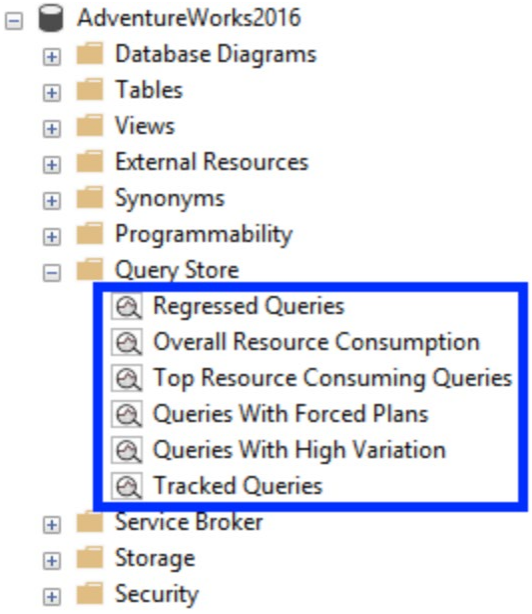
SQL Server Management Studio (SSMS) hosts a set of user interfaces designed for configuring Query Store as well as for consuming collected data about your workloads. In Object Explorer in SSMS, you can enable Query Store by selecting the **Operation Mode (Requested)** box:



You can also use the `ALTER DATABASE` statement to implement the Query Store. For example:

```
ALTER DATABASE AdventureWorks2012 SET QUERY_STORE = ON;
```

Once you've moved forward with either of these options, refresh the database portion of the Object Explorer pane to add the **Query Store** section. You'll be able to see Query Store reports:



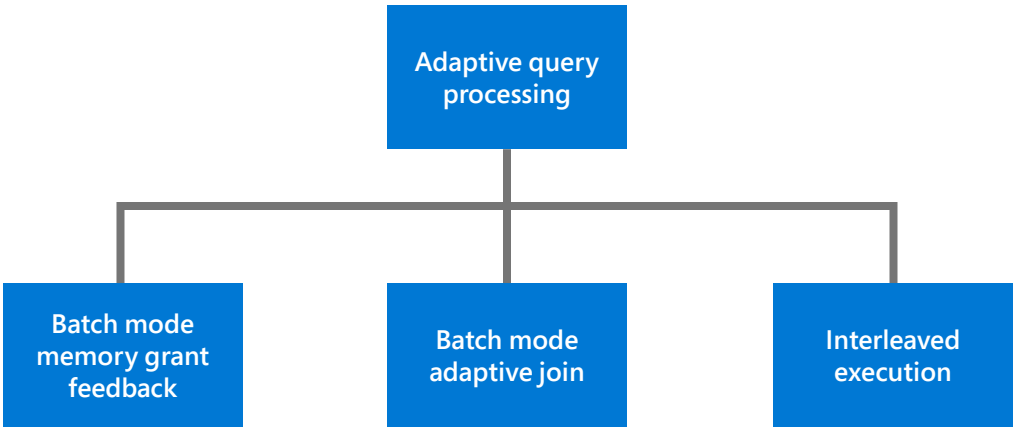
➔ **For more information** about how to monitor performance by using the Query Store, see Microsoft documentation.

Adaptive Query Processing

During query processing and optimization, the cardinality estimation (CE) process is responsible for approximating the number of rows processed at each step in an execution plan. Inaccurate estimations can result in slow query response time, unnecessary resource utilization (memory, CPU, IO), and reduced throughput and concurrency.

To improve the CE process, SQL Server offers a feature family called Adaptive Query Processing (AQP). AQP makes SQL Server significantly faster at processing workloads by allowing the query processor to adjust query plan choices based on runtime characteristics. AQP breaks the barrier between query plan and actual execution. Optimization can be done while the query is executing or even after execution is complete, which benefits subsequent query executions. AQP offers three techniques for adapting to application workload characteristics:

- Batch mode memory grant feedback.
- Batch mode adaptive joins.
- Interleaved execution for multistatement table-valued functions.



Batch mode memory grant feedback

For a query's post-execution plan, SQL Server takes a cardinality estimate for a given T-SQL batch and estimates the minimum memory grant needed for execution as well as the ideal memory grant needed to hold all rows of the batch in memory. If there are problems with the CE, performance suffers and available memory is constrained. Excessive memory grants result in wasted memory and reduced concurrency. Insufficient memory grants cause expensive spills to disk.

With batch mode memory grant feedback, SQL Server recalculates the actual memory required for a query and then updates the grant value for the cached plan. When an identical query statement is executed, the query uses the revised memory grant size. Performance is improved because there are fewer spills to tempdb and, because memory grants to batches are more accurate, additional memory can be provided to the batches that need it most.

For **excessive** grants, if the granted memory is more than two times the size of the used memory, memory grant feedback will recalculate and update the cached plan. Plans with memory grants under 1 MB will not be recalculated for overages. For **insufficiently sized** memory grants that result in a spill to disk for batch mode operators, memory grant feedback will trigger a recalculation. Spill events are reported to memory grant feedback. This event returns the Node ID from the plan and the spilled data size of that node.

Batch mode adaptive joins

SQL Server typically chooses among three types of physical join operators: nested loop joins, merge joins, and hash joins. Each type of join has strengths and weaknesses, depending on the characteristics of the data and query patterns. Which algorithm is best to use in each query depends on the cardinality estimates of the join inputs. Inaccurate input CEs can result in the selection of an inappropriate join algorithm.

With the batch mode adaptive joins feature, SQL Server enables you to defer the selection of a hash join or nested loop join method until **after** the first input has been scanned. The adaptive join operator defines a threshold that is used to decide when to switch to a nested loop plan. Consequenlty, a plan can dynamically switch to a better join strategy during execution.

Interleaved execution for multistatement table-valued functions

Multistatement table-valued functions (MSTVFs) are popular among developers although their initial execution can cause performance slowdowns. With the help of AQP, SQL Server resolves this issue through the interleaved execution for MSTVFs feature. This feature changes the unidirectional boundary between the optimization and execution phases for a single-query execution, and it enables plans to adapt based on the revised cardinality estimates. With interleaved execution, the actual row counts from the MSTVF are used to make plan optimizations downstream from the MSTVF references. The result is a better-informed plan based on actual workload characteristics and, ultimately, better query performance.

When an MSTVF is encountered, the query optimizer will take the following actions:

- Pause optimization.
- Execute the MSTVF subtree to get an accurate CE.
- Continue processing subsequent operations with an accurate set of assumptions.

Based on the execution results (estimated number of rows), the query optimizer can consider a better plan and execute the query with the modified plan.

In general, the higher the skew between the estimated and actual number of rows—coupled with the number of downstream plan operations—the greater the performance impact. **Interleaved execution benefits queries where both of the following are true:**

- There is a large skew between the estimated and actual number of rows for the intermediate result set (in this case, the MSTVF).
- The overall query is sensitive to a change in the size of the intermediate result. This typically happens when there is a complex tree above the subtree in the query plan. A simple "`SELECT *`" from an MSTVF will not benefit from interleaved execution.

Dive deeper: Enabling AQP

You can make workloads automatically eligible for AQP by enabling compatibility level 140 for the database. Here's an example of how you can set this using T-SQL:

```
ALTER DATABASE [YourDatabaseName]
SET COMPATIBILITY_LEVEL = 140;
```

➔ **For more information** about how to use these features, see Adaptive query processing in SQL databases. ■

There are also other tools and features available and supported by SQL Server for monitoring and optimizing performance, including feature configuration options and monitoring and tuning features.

Feature configuration options for performance

SQL Server offers various configuration options for disk, server, table, and query at the database engine level for further improving the SQL Server performance.

Disk configuration. You can set redundant array of independent disks (RAID) levels 0, 1, and 5 with SQL Server. Configure RAID level 0 for disk stripping, 1 for disk mirroring, and 5 striping with parity to be used with SQL Server.

Tempdb configuration. For optimizing tempdb performance, you can use options such as database instant file initialization, autogrow, and disk striping to keep tempdb in your local drive instead of the shared network drive. For further details, see Optimizing tempdb performance in SQL Server.

Server configuration. You can configure settings for processor, memory, index, backup, and query to enhance the server performance with SQL Server. These settings include options for a maximum degree of parallelism like MAXDOP, max server memory, optimize for ad hoc workloads, and nested triggers. For further details, see Configuration Options for Performance.

Database configuration. Set row and page compression using the data compression function for rowstore and columnstore tables and indexes to optimize the database performance. You can also change the compatibility level of a database based on your requirements.

Table configuration. You can use partitioned tables and indexes for improving table performance.

Query performance options. For enhancing query level performance, you can use indexes, partitions, stored procedures, UDFs, and statistics. Use previously discussed features like memory-optimized tables and natively compiled stored procedures to improve In-Memory OLTP performance. See the Query Performance Options for further details.

Monitoring and tuning features for performance

Tools like Query Store, execution plans, live query statistics, and Database Engine Tuning Advisor can help you monitor SQL Server events. You can also use various T-SQL commands like sp_trace_setfilter to tracks engine process events or DBCC TRACEON, a command to enable trace flags. Additionally, you can also establish a performance baseline using sp_configure to determine whether your SQL Server system is performing optimally or not. For more information, see Performance Monitoring and Tuning Tools.

Resource Governor

Resource Governor is a tool in SQL Server that helps you manage and specify limits on the system resource consumptions. You can simply define the resource limit on CPU, physical I/O, and memory that incoming application requests can use. Using Resource Governor, you can continually observe resource usage patterns and adjust the system settings accordingly for maximize effectiveness. To dive deeper into its usage and how it works, see Resource Governor documentation. ■

SQL Server 2017 delivers faster processing capabilities to support your most demanding data-driven applications. It includes a unique set of features built on industry-leading, advanced performance capabilities.

In-Memory OLTP provides faster online transaction processing workloads and better throughput. Columnstore indexes improve database performance and boost high-speed analytics. Plus, features like Adaptive Query Processing enable DBAs to further optimize their query processing capabilities. SQL Server 2017 provides the speed, features, and scalability that organizations need to keep their work up and running at the pace their users demand. ■

See how to run SQL Server on your favorite platform.

Learn about the latest features in SQL Server.

View SQL industry benchmarks and performance.

Download SQL Server.