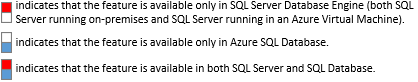
# PERFORMANCE

**Legend**



## Configuration Options for Performance

|  |  |
| --- | --- |
| **Disk configuration options** | security-center-sqlserver [Disk striping and RAID](https://technet.microsoft.com/library/ms190764(v=sql.105).aspx) |
| **Data and log file configuration options** | security-center-sqlserver [Place Data and Log Files on Separate Drives](https://docs.microsoft.com/en-us/sql/relational-databases/policy-based-management/place-data-and-log-files-on-separate-drives) security-center-sqlserver [View or Change the Default Locations for Data and Log Files (SQL Server Management Studio)](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/view-or-change-the-default-locations-for-data-and-log-files) |
| **TempDB configuration options** | security-center-sqlserver [Performance Improvements in TempDB](https://msdn.microsoft.com/library/ms190768.aspx#Anchor_1) security-center-sqlserver [Database Engine Configuration - TempDB](http://msdn.microsoft.com/library/7aabd304-f3c9-4c2d-bf9d-5479ee2498da) security-center-sqlserver [Using SSDs in Azure VMs to store SQL Server TempDB and Buffer Pool Extensions](http://blogs.technet.com/b/dataplatforminsider/archive/2014/09/25/using-ssds-in-azure-vms-to-store-sql-server-tempdb-and-buffer-pool-extensions.aspx) security-center-sqlserver [Disk and performance best practices for temporary disk for SQL Server in Azure Virtual Machines](https://azure.microsoft.com/documentation/articles/virtual-machines-sql-server-performance-best-practices/) |
| **Server Configuration Options** | * **Processor configuration options**   + security-center-sqlserver [affinity mask Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/affinity-mask-server-configuration-option)   + security-center-sqlserver [affinity Input-Output mask Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/affinity-input-output-mask-server-configuration-option)   + security-center-sqlserver [affinity64 mask Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/affinity64-mask-server-configuration-option)   + security-center-sqlserver [affinity64 Input-Output mask Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/affinity64-input-output-mask-server-configuration-option)   + security-center-sqlserver [Configure the max worker threads Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-max-worker-threads-server-configuration-option) * **Memory configuration options**   + security-center-sqlserver [Server Memory Server Configuration Options](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/server-memory-server-configuration-options) * **Index configuration options**   + security-center-sqlserver [Configure the fill factor Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-fill-factor-server-configuration-option) * **Query configuration options** * security-center-sqlserver [Configure the min memory per query Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-min-memory-per-query-server-configuration-option) * security-center-sqlserver [Configure the query governor cost limit Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-query-governor-cost-limit-server-configuration-option) * security-center-sqlserver [Configure the max degree of parallelism Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-max-degree-of-parallelism-server-configuration-option) * security-center-sqlserver [Configure the cost threshold for parallelism Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-cost-threshold-for-parallelism-server-configuration-option) * security-center-sqlserver [optimize for ad hoc workloads Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/optimize-for-ad-hoc-workloads-server-configuration-option) * **Backup configuration options** * security-center-sqlserver [View or Configure the backup compression default Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/view-or-configure-the-backup-compression-default-server-configuration-option) |
| **Database configuration optimization options** | security-center-sqlserver [Data Compression](https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression) security-center-both [View or Change the Compatibility Level of a Database](https://docs.microsoft.com/en-us/sql/relational-databases/databases/view-or-change-the-compatibility-level-of-a-database) security-center-both [ALTER DATABASE SCOPED CONFIGURATION (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-scoped-configuration-transact-sql) |
| **Table configuration optimization** | security-center-sqlserver [Partitioned Tables and Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/partitions/partitioned-tables-and-indexes) |
| **Database Engine Performance in an Azure Virtual Machine** | security-center-sqlserver [Quick check list](https://azure.microsoft.com/documentation/articles/virtual-machines-sql-server-performance-best-practices/) security-center-sqlserver [Virtual machine size and storage account considerations](https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/) security-center-sqlserver [Disks and performance considerations](https://azure.microsoft.com/documentation/articles/virtual-machines-sql-server-performance-best-practices/) security-center-sqlserver [I/O Performance Considerations](https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-sql-server-performance-best-practices/) security-center-sqlserver [Feature specific performance considerations](https://azure.microsoft.com/documentation/articles/virtual-machines-sql-server-performance-best-practices/) |

Query Performance Options

|  |  |
| --- | --- |
| security-center-both [**Indexes**](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/indexes) | [Reorganize and Rebuild Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/reorganize-and-rebuild-indexes) [Specify Fill Factor for an Index](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/specify-fill-factor-for-an-index) [Configure Parallel Index Operations](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/configure-parallel-index-operations) [SORT\_IN\_TEMPDB Option For Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/sort-in-tempdb-option-for-indexes) [Improve the Performance of Full-Text Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/search/improve-the-performance-of-full-text-indexes) [Configure the min memory per query Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-min-memory-per-query-server-configuration-option) [Configure the index create memory Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-index-create-memory-server-configuration-option) |
| security-center-both [**Partitioned Tables and Indexes**](https://docs.microsoft.com/en-us/sql/relational-databases/partitions/partitioned-tables-and-indexes) | [Benefits of Partitioning](https://msdn.microsoft.com/library/ms190787.aspx#Anchor_0) |
| security-center-both [**Joins**](https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins) | [Join Fundamentals](https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins#fundamentals) [Nested Loops join](https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins#nested_loops) [Merge join](https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins#merge) [Hash join](https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins#hash) |
| security-center-both [**Subqueries**](https://docs.microsoft.com/en-us/sql/relational-databases/performance/subqueries) | [Subquery Fundamentals](https://docs.microsoft.com/en-us/sql/relational-databases/performance/subqueries#fundamentals) [Correlated subqueries](https://docs.microsoft.com/en-us/sql/relational-databases/performance/subqueries#correlated) [Subquery types](https://docs.microsoft.com/en-us/sql/relational-databases/performance/subqueries#types) |
| security-center-both [**Stored Procedures**](https://docs.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine) | [CREATE PROCEDURE (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql#best-practices) |
| security-center-both [**User-Defined Functions**](https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/user-defined-functions) | [CREATE FUNCTION (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql#best-practices) |
| security-center-both **Parallelism optimization** | [Configure the max worker threads Server Configuration Option](https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/configure-the-max-worker-threads-server-configuration-option) [ALTER DATABASE SCOPED CONFIGURATION (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-scoped-configuration-transact-sql) |
| security-center-both **Query optimizer optimization** | [ALTER DATABASE SCOPED CONFIGURATION (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-scoped-configuration-transact-sql) |
| security-center-both [**Statistics**](https://docs.microsoft.com/en-us/sql/relational-databases/statistics/statistics) | [When to Update Statistics](https://msdn.microsoft.com/library/ms190397.aspx#Anchor_3) [Update Statistics](https://docs.microsoft.com/en-us/sql/relational-databases/statistics/update-statistics) |
| security-center-both [**In-Memory OLTP (In-Memory Optimization)**](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/in-memory-oltp-in-memory-optimization) | [Memory-Optimized Tables](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/memory-optimized-tables) [Natively Compiled Stored Procedures](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/natively-compiled-stored-procedures) [Creating and Accessing Tables in TempDB from Natively Compiled Stored Procedures](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/create-and-access-tables-in-tempdb-from-stored-procedures) [Troubleshooting Common Performance Problems with Memory-Optimized Hash Indexes](http://msdn.microsoft.com/library/1954a997-7585-4713-81fd-76d429b8d095) [Demonstration: Performance Improvement of In-Memory OLTP](https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/demonstration-performance-improvement-of-in-memory-oltp) |

## Performance Center for SQL Server Database Engine and Azure SQL Database

## Configuring Storage Spaces with a NVDIMM-N write-back cache

Windows Server 2016 supports NVDIMM-N devices that allow for extremely fast input/output (I/O) operations. One attractive way of using such devices is as a write-back cache to achieve low write latencies.

## Adaptive query processing in SQL databases

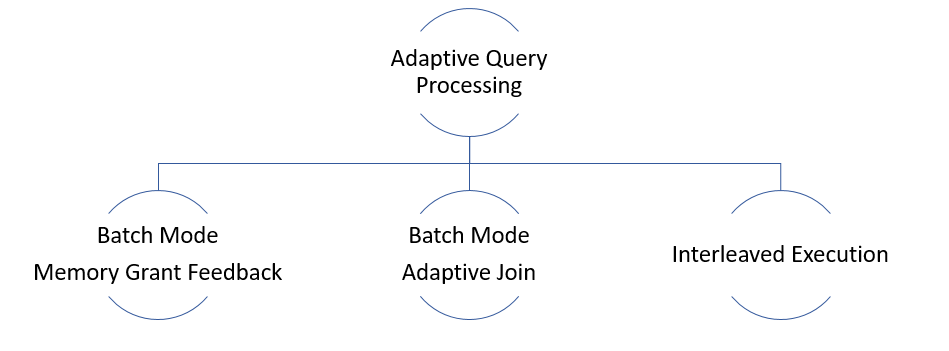
This article introduces these adaptive query processing features that you can use to improve query performance in SQL Server Automatic Tuning

* Batch mode memory grant feedback.
* Batch mode adaptive join.
* Interleaved execution.

At a general level, SQL Server executes a query as follows:

1. The query optimization process generates a set of feasible execution plans for a specific query. During this time, the cost of plan options is estimated and the plan with the lowest estimated cost is used.
2. The query execution process takes the plan chosen by the query optimizer and uses it for execution.

Sometimes the plan chosen by the query optimizer is not optimal for a variety of reasons. For example, the estimated number of rows flowing through the query plan may be incorrect. The estimated costs help determine which plan gets selected for use in execution. If cardinality estimates are incorrect, the original plan is still used despite the poor original assumptions.



### How to enable adaptive query processing

You can make workloads automatically eligible for adaptive query processing by enabling compatibility level 140 for the database.

ALTER DATABASE [WideWorldImportersDW] SET COMPATIBILITY\_LEVEL = 140;

#### Batch mode memory grant feedback

A query’s post-execution plan in SQL Server includes the minimum required memory needed for execution and the ideal memory grant size to have all rows fit in memory. Performance suffers when memory grant sizes are incorrectly sized. Excessive grants result in wasted memory and reduced concurrency. Insufficient memory grants cause expensive spills to disk. By addressing repeating workloads, batch mode memory grant feedback recalculates the actual memory required for a query and then updates the grant value for the cached plan.

##### **Memory grant feedback sizing**

**For excessive grants**, if the granted memory is more than two times the size of the actual used memory, memory grant feedback will recalculate the memory grant 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 of the memory grant.

##### **Memory grant feedback and parameter sensitive scenarios**

Different parameter values may also require different query plans in order to remain optimal. This type of query is defined as “parameter-sensitive.” For parameter-sensitive plans, memory grant feedback will disable itself on a query if it has unstable memory requirements. The plan is disabled after several repeated runs of the query and this can be observed by monitoring the memory\_grant\_feedback\_loop\_disabled XEvent.

##### **Memory grant feedback caching**

##### **Tracking memory grant feedback activity**

You can track memory grant feedback events using the memory\_grant\_updated\_by\_feedback XEvent event.

##### **Memory grant feedback, resource governor and query hints**

The actual memory granted honors the query memory limit determined by the resource governor or query hint.

### Batch mode adaptive joins

The batch mode adaptive joins feature enables the choice of a hash join or nested loop join method to be deferred 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. Your plan can therefore dynamically switch to a better join strategy during execution. Here’s how it works:

* If the row count of the build join input is small enough that a nested loop join would be more optimal than a hash join, your plan switches to a nested loop algorithm.
* If the build join input exceeds a specific row count threshold, no switch occurs and your plan continues with a hash join.

### Interleaved execution

### Interleaved execution for multi-statement table valued functions

Interleaved execution changes the unidirectional boundary between the optimization and execution phases for a single-query execution and enables plans to adapt based on the revised cardinality estimates. During optimization if we encounter a candidate for interleaved execution, which is currently **multi-statement table valued functions (MSTVFs)**, we will pause optimization, execute the applicable subtree, capture accurate cardinality estimates, and then resume optimization for downstream operations. MSTVFs have a fixed cardinality guess of “100” in SQL Server 2014 and SQL Server 2016, and “1” for earlier versions. Interleaved execution helps workload performance issues that are due to these fixed cardinality estimates associated with multi-statement table valued functions.

## Automatic tuning

Automatic tuning is a database feature in SQL server 2017 that provides insight into potential query performance problems, recommend solutions, and automatically fix identified problems.

Automatic tuning in SQL Server 2017 notifies you whenever a potential performance issue is detected, and lets you apply corrective actions, or lets the Database Engine automatically fix performance problems. Automatic tuning in SQL Server 2017 enables you to identify and fix performance issues caused by **SQL plan choice regressions**. Automatic tuning in Azure SQL Database creates necessary indexes and drops unused indexes.

There are two automatic tuning features that are available:

* **Automatic plan correction** (available in SQL Server 2017 and Azure SQL Database) that identifies problematic query execution plans and fixes SQL plan performance problems.
* **Automatic index management** (available only in Azure SQL Database) that identifies indexes that should be added in your database, and indexes that should be removed.

### What is SQL plan choice regression?

SQL Server Database Engine may use different SQL plans to execute the Transact-SQL queries. Query plans depend on the statistics, indexes, and other factors. The optimal plan that should be used to execute some Transact-SQL query might be changed over time. In some cases, the new plan might not be better than the previous one, and the new plan might cause a performance regression.

Whenever you notice the plan choice regression, you should find some previous good plan and force it instead of the current one using sp\_query\_store\_force\_plan procedure. Database Engine in SQL Server 2017 provides information about regressed plans and recommended corrective actions. Additionally, Database Engine enables you to fully automate this process and let Database Engine fix any problem found related to the plan changes.

#### Automatic plan choice correction

Database Engine can automatically switch to the last known good plan whenever the plan choice regression is detected.

ALTER DATABASE current

SET AUTOMATIC\_TUNING ( FORCE\_LAST\_GOOD\_PLAN = ON );

### Automatic index management

In Azure SQL Database, index management is easy because Azure SQL Database learns about your workload and ensures that your data is always optimally indexed. Proper index design is crucial for optimal performance of your workload, and automatic index management can help you optimize your indexes. Automatic index management can either fix performance issues in incorrectly indexed databases, or maintain and improve indexes on the existing database schema. Automatic tuning in Azure SQL Database performs the following actions:

* Identifies indexes that could improve performance of your T-SQL queries that read data from the tables.
* Identifies the redundant indexes or indexes that were not used in longer period of time that could be removed. Removing unnecessary indexes improves performance of the queries that update data in tables.

## Cardinality Estimation

Most systems benefit from the latest cardinality estimation (CE) because it is the most accurate. The CE predicts how many rows your query will likely return. The cardinality prediction is used by the Query Optimizer to generate the optimal query plan. With more accurate estimations, the Query Optimizer can usually do a better job of producing a more optimal query plan.

**Query store:** Starting with SQL Server 2016, the query store is a handy tool for examining the performance of your queries. In Management Studio, in the **Object Explorer** under your database node, a **Query Store** node is displayed when the query store is enabled.

ALTER DATABASE <yourDatabase> SET QUERY\_STORE = ON;

ALTER DATABASE <yourDatabase> SET QUERY\_STORE CLEAR;

Another option for tracking the cardinality estimation process is to use the extended event named **query\_optimizer\_estimate\_cardinality**.

## Monitor and Tune for Performance

The goal of monitoring databases is to assess how a server is performing. Effective monitoring involves taking periodic snapshots of current performance to isolate processes that are causing problems, and gathering data continuously over time to track performance trends.

Ongoing evaluation of the database performance helps you minimize response times and maximize throughput, yielding optimal performance. Efficient network traffic, disk I/O, and CPU usage are key to peak performance.

### Monitoring and tuning databases for performance

* Determine whether you can improve performance. For example, by monitoring the response times for frequently used queries, you can determine whether changes to the query or indexes on the tables are required.
* Evaluate user activity. For example, by monitoring users trying to connect to an instance of SQL Server, you can determine whether security is set up adequately and test applications or development systems. For example, by monitoring SQL queries as they are executed, you can determine whether they are written correctly and producing the expected results.
* Troubleshoot problems or debug application components, such as stored procedures

### Monitor SQL Server Components

#### Select the Appropriate Tool

The Windows operating system and SQL Server provide a complete set of tools to monitor servers in transaction-intensive environments.

Windows provides the following tools for monitoring applications that are running on a server:

* System Monitor, which lets you collect and view real-time data about activities such as memory, disk, and processor usage
* Performance logs and alerts
* Task Manager

SQL Server provides the following tools for monitoring components of SQL Server:

* SQL Trace
* SQL Server Profiler
* Distributed Replay Utility
* SQL Server Management Studio Activity Monitor
* SQL Server Management Studio Graphical Showplan
* Stored procedures
* Database Console Commands (DBCC)
* Built-in functions
* Trace flags

### Performance Monitoring and Tuning Tools

|  |  |
| --- | --- |
| **Tool** | **Description** |
| [sp\_trace\_setfilter](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-trace-setfilter-transact-sql) | SQL Server Profiler tracks engine process events, such as the start of a batch or a transaction, enabling you to monitor server and database activity (for example, deadlocks, fatal errors, or login activity). You can capture SQL Server Profiler data to a SQL Server table or a file for later analysis, and you can also replay the events captured on SQL Server step by step, to see exactly what happened. |
| [SQL Server Distributed Replay](https://docs.microsoft.com/en-us/sql/tools/distributed-replay/sql-server-distributed-replay) | Microsoft SQL Server Distributed Replay can use multiple computers to replay trace data, simulating a mission-critical workload. |
| [Monitor Resource Usage](https://docs.microsoft.com/en-us/sql/relational-databases/performance-monitor/monitor-resource-usage-system-monitor) | System Monitor primarily tracks resource usage, such as the number of buffer manager page requests in use, enabling you to monitor server performance and activity using predefined objects and counters or user-defined counters to monitor events. |
| [Open Activity Monitor](https://docs.microsoft.com/en-us/sql/relational-databases/performance-monitor/open-activity-monitor-sql-server-management-studio) | The Activity Monitor in SQL Server Management Studio is useful for ad hoc views of current activity and graphically displays information about: Processes running on an instance of SQL Server. **Blocked processes / Locks / User activity.** |
| [Live Query Statistics](https://docs.microsoft.com/en-us/sql/relational-databases/performance/live-query-statistics) | Displays real-time statistics about query execution steps. Because this data is available while the query is executing, these execution statistics are extremely useful for debugging query performance issues. |
| Error Logs | The Windows application event log provides an overall picture of events occurring on the Windows Server and Windows operating systems as a whole, as well as events in SQL Server, SQL Server Agent, and full-text search. |
| [System Stored Procedures](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/system-stored-procedures-transact-sql) | The following SQL Server system stored procedures , DMVS provide a powerful alternative for many monitoring tasks: **sp\_who** : Reports snapshot information about current SQL Server users and processes, including the currently executing statement and whether the statement is blocked. **sp\_lock** : Reports snapshot information about locks, including the object ID, index ID, type of lock, and type or resource to which the lock applies. **sp\_spaceused** : Displays an estimate of the current amount of disk space used by a table (or a whole database). sp\_monitor : Displays statistics, including CPU usage, I/O usage, and the amount of time idle since sp\_monitor was last executed. |
| [DBCC (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-transact-sql) | DBCC (Database Console Command) statements enable you to check performance statistics and the logical and physical consistency of a database. |
| [Built-in Functions (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/functions/functions) | Built-in functions display snapshot statistics about SQL Server activity since the server was started; these statistics are stored in predefined SQL Server counters. For example, **@@CPU\_BUSY** contains the amount of time the CPU has been executing SQL Server code; **@@CONNECTIONS**contains the number of SQL Server connections or attempted connections; and **@@PACKET\_ERRORS** contains the number of network packets occurring on SQL Server connections. |
| [Trace Flags (Transact-SQL)](https://docs.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-traceon-trace-flags-transact-sql) | Trace flags display information about a specific activity within the server and are used to diagnose problems or performance issues (for example, deadlock chains). |
| [Database Engine Tuning Advisor](https://docs.microsoft.com/en-us/sql/relational-databases/performance/database-engine-tuning-advisor) | Database Engine Tuning Advisor analyzes the performance effects of Transact-SQL statements executed against databases you want to tune. Database Engine Tuning Advisor provides recommendations to add, remove, or modify indexes, indexed views, and partitioning. |

### Establish a Performance Baseline

To establish a server performance baseline. Compare each new set of measurements with those taken earlier.

The following areas affect the performance of SQL Server:

* System resources (hardware)
* Network architecture
* The operating system
* Database applications
* Client applications

At a minimum, use baseline measurements to determine:

* Peak and off-peak hours of operation.
* Production-query or batch-command response times.
* Database backup and restore completion times.

### Isolate Performance Problems

It is often more effective to use several Microsoft SQL Server or Microsoft Windows tools together to isolate database performance problems than to use one tool at a time. For example, the graphical Execution Plan feature, also called Showplan, helps you quickly recognize deadlocks in a single query. However, you can recognize some other performance problems more easily if you use the monitoring features of SQL Server and Windows together.

SQL Server Profiler can be used to monitor and troubleshoot Transact-SQL and application-related problems. System Monitor can be used to monitor hardware and other system-related problems.

You can monitor the following areas to troubleshoot problems:

* SQL Server stored procedures or batches of Transact-SQL statements submitted by user applications.
* User activity, such as blocking locks or deadlocks.
* Hardware activity, such as disk usage.

Problems can include:

* Application development errors involving incorrectly written Transact-SQL statements.
* Hardware errors, such as disk- or network-related errors.
* Excessive blocking due to an incorrectly designed database.

### Identify Bottlenecks

### Server Performance and Activity Monitoring

* Start System Monitor (Windows)
* Set Up a SQL Server Database Alert (Windows)
* View the Windows Application Log (Windows)
* View the SQL Server Error Log (SQL Server Management Studio)
* Save Deadlock Graphs (SQL Server Profiler)
* Open, View, and Print a Deadlock File (SQL Server Management Studio)
* Save Showplan XML Events Separately (SQL Server Profiler)
* Save Showplan XML Statistics Profile Events Separately (SQL Server Profiler)
* Display and Save Execution Plans
* Display the Estimated Execution Plan
* Display an Actual Execution Plan
* Save an Execution Plan in XML Format

### Live Query Statistics

**This Feature Starts From SQL server 2016**

SQL Server Management Studio provides the ability to view the live execution plan of an active query. This live query plan provides real-time insights into the query execution process as the controls flow from one query plan operator to another. The live query plan displays the overall query progress and operator-level run-time execution statistics such as the number of rows produced, elapsed time, operator progress, etc. These execution statistics are extremely useful for debugging query performance issues.

Use this information to understand the overall query execution process and to debug long running queries, queries that run indefinitely, queries that cause tempdb overflow, and timeout issues.

#### Limitations of Live Query Statistics

There are currently a few limitations when working with Live Query Statistics:

* Columnstore indexes are not supported
* Memory-optimized tables are not supported
* Natively compiled stored procedures are not supported

Warning

This feature is primarily intended for troubleshooting purposes. Using this feature can moderately slow the overall query performance.

### Monitoring Performance By Using the Query Store

The SQL Server Query Store feature helps you to track query plans, runtime statistics and queries/plans history. It simplifies performance troubleshooting by helping you quickly find performance differences caused by query plan changes. Query Store automatically captures a history of queries, plans, and runtime statistics, and retains these for your review. You can quickly find new queries with multiple plans, identify un-efficient plans and force a better plan.

### ALTER DATABASE AdventureWorks2012 SET QUERY\_STORE = ON;

The query store contains three stores:

* a **plan store** for persisting the execution plan information.
* a **runtime stats store** for persisting the execution statistics information.
* a **wait stats store** for persisting wait statistics information.

### Using the Query Store with In-Memory OLTP

**All** – Captures all queries. **This is the default option**.

**Auto** – Infrequent queries and queries with insignificant compile and execution duration are ignored. Thresholds for execution count, compile and runtime duration are internally determined.

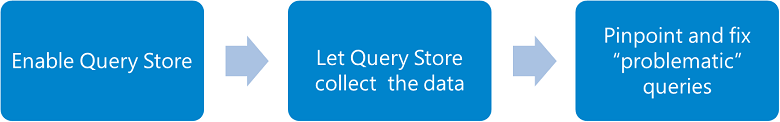
**None** – Query Store stops capturing new queries.

1. [Use Query Performance Insight in Azure SQL Database](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#Insight)
2. [Using Query Store with Elastic Pool Databases](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#using-query-store-with-elastic-pool-databases)
3. [Keep Query Store adjusted to your workload](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#Configure)
4. [How to start with query performance troubleshooting](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#how-to-start-with-query-performance-troubleshooting)
   1. [Verify Query Store is collecting query data continuously](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#Verify)
   2. [Set the optimal query capture mode](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#set-the-optimal-query-capture-mode)
   3. [Keep the most relevant data in Query Store](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#keep-the-most-relevant-data-in-query-store)
5. [**Avoid using non-parameterized queries**](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#Parameterize)
6. [Avoid a DROP and CREATE pattern when maintaining containing objects for the queries](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#Drop)
7. [Check the status of Forced Plans regularly](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#CheckForced)
8. [Avoid renaming databases if you have queries with Forced Plans](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#Renaming)
9. [Use traceflags on mission critical servers to improve recovery from disaster](https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store#Recovery)

### Best Practice with the Query Store

#### How to start with query performance troubleshooting

Troubleshooting workflow with Query Store is simple, as shown on the following diagram:



### Query Store Usage Scenarios

Query Store can be used in wide set of scenarios when tracking and ensuring predictable workload performance is critical. Here are some examples you can consider:

* Pinpoint and fix queries with plan choice regressions
* Identify and tune top resource consuming queries
* A/B testing
* Keep performance stability during the upgrade to newer SQL Server
* Identify and improve ad-hoc workloads

#### Pinpoint and fix queries with plan choice regressions

With the Query Store you can quickly:

* Identify all queries which execution metrics have been degraded in the period of time of interest (last hour, day, week, etc.). Use **Regressed Queries** in SQL Server Management Studio to speed up your analysis.
* Among the regressed queries it’s very easy to find those that had multiple plans and which degraded because of the bad plan choice. Use **Plan Summary** pane in **Regressed Queries** to visualize all plans for a regressed query and their query performance over time.
* Force the previous plan from the history if it proved to be better. Use **Force Plan** button in **Regressed Queries** to force selected plan for the query.

#### Identify and tune top resource consuming queries

When you identify a query with sub-optimal performance, your action depends on the nature of the problem:

1. If the query was executed with multiple plans and the last plan is significantly worse than previous plan, you can use the plan forcing mechanism to ensure SQL Server will use the optimal plan for future executions
2. Check if the optimizer is suggesting any missing indexes in XML plan. If yes, create the missing index and use the Query Store to evaluate query performance after the index creation
3. Make sure that the statistics are up-to-date for the underlying tables used by the query.
4. Make sure that indexes used by the query are defragmented.
5. Consider rewriting expensive query. For example, take advantages of query parameterization and reduce usage of dynamic SQL.

#### A/B testing

Use Query Store to compare workload performance before and after the application change you plan to introduce. The following list contains several examples where you can use Query Store to assess impact of the environment or application change to the workload performance:

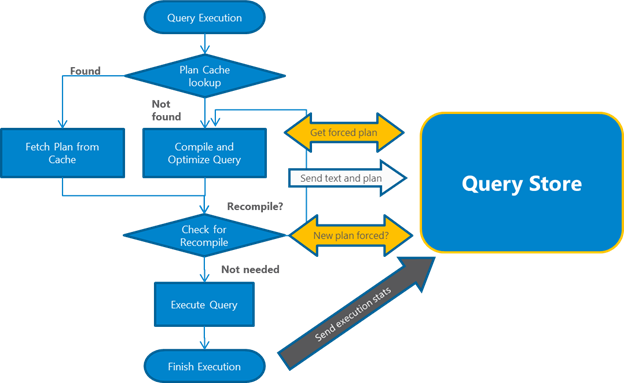
* Rolling out new application version.
* Adding new hardware to the server.
* Creating missing indexes on tables referenced by expensive queries.
* Applying filtering policy for row-level security
* Adding temporal system-versioning to tables that are frequently modified by your OLTP applications.

In any of these scenarios apply the following workflow:

1. Run your workload with the Query Store before the planned change to generate performance baseline.
2. Apply application change at the controlled moment in time.
3. Continue running the workload long enough to generate performance image of the system after the change
4. Compare results from #1 and #3.
   1. Open **Overall Database Consumption** to determine impact to the entire database.
   2. Open **Top Resource Consuming Queries** (or run your own analysis using Transact-SQL) to analyze impact of the change to the most important queries.
5. Decide whether to keep the change or perform roll back in case when new performance is unacceptable.

### How Query Store Collects Data

|  |  |
| --- | --- |
| **View** | **Description** |
| **sys.query\_store\_query\_text** | Presents unique query texts executed against the database. Comments and spaces before and after the query text are ignored. Comments and spaces inside text are not ignored. Every statement in the batch generates a separate query text entry. |
| **sys.query\_context\_settings** | Presents unique combinations of plan affecting settings under which queries are executed. The same query text executed with different plan affecting settings produces separate query entry in the Query Store because context\_settings\_id is part of the query key. |
| **sys.query\_store\_query** | Query entries that are tracked and forced separately in the Query Store. A single query text can produce multiple query entries if it is executed under different context settings or if it is executed outside vs. inside of different Transact-SQL modules (stored procedures, triggers, etc.). |
| **sys.query\_store\_plan** | Presents estimated plan for the query with the compile time statistics. Stored plan is equivalent to one that you would get by using SET SHOWPLAN\_XML ON. |
| **sys.**  **query\_store\_runtime\_stats\_interval** | [Query Store divides time into automatically generated time windows (intervals) and stores aggregated statistics on that interval for every executed plan. The size of the interval is controlled by the configuration option Statistics Collection Interval (in Management Studio) or INTERVAL\_LENGTH\_MINUTES using ALTER DATABASE SET Options (Transact-SQL).](https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-database-transact-sql-set-options) |
| **sys.**  **query\_store\_runtime\_stats** | Aggregated runtime statistics for executed plans. All captured metrics are expressed in form of 4 statistic functions: Average, Minimum, Maximum, and Standard Deviation. |



## Database Tuning Advisor

The Microsoft Database Engine Tuning Advisor (DTA) analyzes databases and makes recommendations that you can use to optimize query performance. You can use the Database Engine Tuning Advisor to select and create an optimal set of indexes, indexed views, or table partitions without having an expert understanding of the database structure or the internals of SQL Server. Using the DTA, you can perform the following tasks.

* Troubleshoot the performance of a specific problem query
* Tune a large set of queries across one or more databases
* Perform an exploratory what-if analysis of potential physical design changes
* Manage storage space

### Database Engine Tuning Advisor Benefits

* Recommend the best mix of rowstore and [columnstore](https://docs.microsoft.com/en-us/sql/relational-databases/performance/columnstore-index-recommendations-in-database-engine-tuning-advisor-dta) indexes for databases by using the query optimizer to analyze queries in a workload.
* Recommend aligned or non-aligned partitions for databases referenced in a workload.
* Recommend indexed views for databases referenced in a workload.
* Analyze the effects of the proposed changes, including index usage, query distribution among tables, and query performance in the workload.
* Recommend ways to tune the database for a small set of problem queries.
* Allow you to customize the recommendation by specifying advanced options such as disk space constraints.
* Provide reports that summarize the effects of implementing the recommendations for a given workload.
* Consider alternatives in which you supply possible design choices in the form of hypothetical configurations for Database Engine Tuning Advisor to evaluate.
* Tune workloads from a variety of sources including SQL Server Query Store, Plan Cache, SQL Server Profiler Trace file or table, or a .SQL file.

The Database Engine Tuning Advisor is designed to handle the following types of query workloads.

* Online transaction processing (OLTP) queries only
* Online analytical processing (OLAP) queries only
* Mixed OLTP and OLAP queries
* Query-heavy workloads (more queries than data modifications)
* Update-heavy workloads (more data modifications than queries)

### Limitations and Restrictions

The Database Engine Tuning Advisor has the following limitations and restrictions.

* It cannot add or drop unique indexes or indexes that enforce PRIMARY KEY or UNIQUE constraints.
* It cannot analyze a database that is set to single-user mode.
* Database Engine Tuning Advisor might not make recommendations under the following circumstances:
  + The table being tuned contains less than 10 data pages.
  + The recommended indexes would not offer enough improvement in query performance over the current physical database design.
  + The user who runs Database Engine Tuning Advisor is not a member of the **db\_owner** database role or the **sysadmin** fixed server role. The queries in the workload are analyzed in the security context of the user who runs the Database Engine Tuning Advisor. The user must be a member of the **db\_owner** database role.
* Database Engine Tuning Advisor stores tuning session data and other information in the **msdb** database. If changes are made to the **msdb** database you may risk losing tuning session data. To eliminate this risk, implement an appropriate backup strategy for the **msdb**database.

## Plan Guides

Plan guides let you optimize the performance of queries when you cannot or do not want to directly change the text of the actual query in SQL Server 2017. Plan guides influence the optimization of queries by attaching query hints or a fixed query plan to them. Plan guides can be useful when a small subset of queries in a database application provided by a third-party vendor are not performing as expected. In the plan guide, you specify the Transact-SQL statement that you want optimized and either an OPTION clause that contains the query hints you want to use or a specific query plan you want to use to optimize the query. When the query executes, SQL Server matches the Transact-SQL statement to the plan guide and attaches the OPTION clause to the query at run time or uses the specified query plan.

#### Types of Plan Guides

##### OBJECT plan guide

##### SQL plan guide

sp\_create\_plan\_guide

@name = N'Guide2',

@stmt = N'SELECT TOP 1 \* FROM Sales.SalesOrderHeader ORDER BY OrderDate DESC',

@type = N'SQL',

@module\_or\_batch = NULL,

@params = NULL,

@hints = N'OPTION (MAXDOP 1)';

##### TEMPLATE plan guide

### Create a New Plan Guide

### Create a Plan Guide for Parameterized Queries

### Specify Query Parameterization Behavior by Using Plan Guides

### Apply a Fixed Query Plan to a Plan Guide

### Attach Query Hints to a Plan Guide

### View Plan Guide Properties

### Use SQL Server Profiler to Create and Test Plan Guides

### Validate Plan Guides After Upgrade

### Delete a Plan Guide

### Enable or Disable a Plan Guide

## Monitor Resource Usage (System Monitor)