SQL Server configuration recommendations

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Description | Date |
| 1.0 | Mark Herasimovich | First version | 4/29/2015 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[SQL Server settings 4](#_Toc418081834)

[TempDB settings 5](#_Toc418081835)

[Production database settings 6](#_Toc418081836)

[Backups 6](#_Toc418081837)

[Index maintenance 6](#_Toc418081838)

# SQL Server settings

You can manage and optimize SQL Server resources through configuration options by using SQL Server Management Studio or the sp\_configure system stored procedure. The most commonly used server configuration options are available through SQL Server Management Studio; all configuration options are accessible through sp\_configure. Consider the effects on your system carefully before setting these options. For more information see [Server configuration options](https://msdn.microsoft.com/en-us/library/ms189631(v=sql.110).aspx).

Below you can find SQL Server settings which you have to optimize for your requirements:

* [Backup compression default](https://technet.microsoft.com/en-us/library/bb677250(v=sql.105).aspx). By default, compression significantly increases CPU usage, and the additional CPU consumed by the compression process might adversely impact concurrent operations. But it could extremely decrease the time of database restoring.
* [Cost threshold for parallelism](https://technet.microsoft.com/en-us/library/ms188603(v=sql.105).aspx). Use the **cost threshold for parallelism** option to specify the threshold at which Microsoft SQL Server creates and runs parallel plans for queries. SQL Server creates and runs a parallel plan for a query only when the estimated cost to run a serial plan for the same query is higher than the value set in **cost threshold for parallelism**. The cost refers to an estimated elapsed time in seconds required to run the serial plan on a specific hardware configuration. Only set **cost threshold for parallelism** on symmetric multiprocessors.

Longer queries usually benefit from parallel plans; the performance advantage negates the additional time required to initialize, synchronize, and terminate parallel plans. The **cost threshold for parallelism option** is actively used when a mix of short and longer queries is run. The short queries run serial plans, whereas the longer queries use parallel plans. The value of **cost threshold for parallelism** determines which queries are considered short, and they should therefore be run using serial plans.

In certain cases, a parallel plan may be chosen even though the query's cost plan is less than the current **cost threshold for parallelism** value. This can happen because the decision to use a parallel or serial plan is based on a cost estimate provided before the full optimization is complete.

* [Max degree of parallelism](https://technet.microsoft.com/en-us/library/ms181007(v=sql.105).aspx). When SQL Server runs on a computer with more than one microprocessor or CPU, it detects the best degree of parallelism, that is, the number of processors employed to run a single statement, for each parallel plan execution. You can use the max degree of parallelism option to limit the number of processors to use in parallel plan execution. To enable the server to determine the maximum degree of parallelism, set this option to 0, the default value. Setting maximum degree of parallelism to 0 allows SQL Server to use all the available processors up to 64 processors. To suppress parallel plan generation, set max degree of parallelism to 1. Set the value to a number greater than 1 to restrict the maximum number of processors used by a single query execution. The maximum value for the degree of parallelism setting is controlled by the edition of SQL Server, CPU type, and operating system. If a value greater than the number of available processors is specified, the actual number of available processors is used. If the computer has only one processor, the max degree of parallelism value is ignored.

For example, you could use **MAXDOP calculator.sql** script (attached) for finding theoretical optimal value of max degree of parallelism. But you should use it at own risk, because script designed for average SQL Server. For OLTP, DWH and hybrid servers the value will be different.

* [Max server memory](https://msdn.microsoft.com/en-us/library/ms178067(v=sql.110).aspx). Use the two server memory options, min server memory and max server memory, to reconfigure the amount of memory (in megabytes) that is managed by the SQL Server Memory Manager for a SQL Server process used by an instance of SQL Server.

The default setting for min server memory is 0, and the default setting for max server memory is 2147483647 MB. By default, SQL Server can change its memory requirements dynamically based on available system resources.

When SQL Server is using memory dynamically, it queries the system periodically to determine the amount of free memory. Maintaining this free memory prevents the operating system (OS) from paging. If less memory is free, SQL Server releases memory to the OS. If more memory is free, SQL Server may allocate more memory. SQL Server adds memory only when its workload requires more memory; a server at rest does not increase the size of its virtual address space.

For example, you could use **OptimalMaxServerMemory.sql** script (attached) for finding theoretical optimal value for max server memory. But you should use it at own risk, because script designed for average SQL Server. If the server used not only for MS SQL but for another tasks, you have to take into account how much memory uses these tasks.

# TempDB settings

The size and physical placement of the **tempdb** database can affect the performance of a system. For example, if the size that is defined for tempdb is too small, part of the system-processing load may be taken up with autogrowing tempdb to the size required to support the workload every time you restart the instance of SQL Server. You can avoid this overhead by increasing the sizes of the **tempdb** data and log file. If it possible for your environment – always place **tempdb** database on different physical disk. It will increase performance of SQL Server and your databases.

To achieve optimal tempdb performance, msdn recommends the following configuration for **tempdb** in a production environment:

* Set the recovery model of tempdb to SIMPLE. This model automatically reclaims log space to keep space requirements small.
* For more information, see [ALTER DATABASE (Transact-SQL)](https://technet.microsoft.com/en-us/library/ms174269) or [How to: View or Change the Recovery Model of a Database (SQL Server Management Studio)](https://technet.microsoft.com/en-us/library/ms189272).
* Allow for tempdb files to automatically grow as required. This allows for the file to grow until the disk is full.
* Set the file growth increment to a reasonable size to avoid the tempdb database files from growing by too small a value. If the file growth is too small, compared to the amount of data that is being written to tempdb, tempdb may have to constantly expand. This will affect performance. We recommend the following general guidelines for setting the FILEGROWTH increment for tempdb files.

|  |  |
| --- | --- |
| tempdb file size | FILEGROWTH increment |
| 0 to 100 MB | 10 MB |
| 100 to 200 MB | 20 MB |
| 200 MB or more | 10%\* |

\* You may have to adjust this percentage based on the speed of the I/O subsystem on which the tempdb files are located. To avoid potential latch time-outs, we recommend limiting the autogrow operation to approximately two minutes. For example, if the I/O subsystem can initialize a file at 50 MB per second, the FILEGROWTH increment should be set to a maximum of 6 GB, regardless of the tempdb file size. If possible, use [instant database file initialization](https://technet.microsoft.com/en-us/library/ms175935) to improve the performance of autogrow operations.

* Preallocate space for all tempdb files by setting the file size to a value large enough to accommodate the typical workload in the environment. This prevents tempdb from expanding too frequently, which can affect performance. The tempdb database should be set to autogrow, but this should be used to increase disk space for unplanned exceptions.
* Create as many files as needed to maximize disk bandwidth. Using multiple files reduces tempdb storage contention and yields significantly better scalability. However, do not create too many files because this can reduce performance and increase management overhead. As a general guideline, create one data file for each CPU on the server (accounting for any [affinity mask](https://technet.microsoft.com/en-us/library/ms187104) settings) and then adjust the number of files up or down as necessary. Note that a dual-core CPU is considered to be two CPUs.
* Make each data file the same size; this allows for optimal proportional-fill performance.
* Put the tempdb database on a fast I/O subsystem. Use disk striping if there are many directly attached disks.
* Put the tempdb database on disks that differ from those that are used by user databases.

For more information see [Optimizing tempdb Performance](https://technet.microsoft.com/en-us/library/ms175527).

# Production database settings

Good practices for production database settings

* Production database should be stored on different with tempdb and system PHYSICAL hard drives.
* If it possible – store database files (log, data, indexes) on different PHYSICAL hard drives. It could improve database performance.
* If it possible – try to split database between filegroup on different physical hard drives. For example: create different filegroup for each type of data:
  + PRIMARY
  + Fast tables
  + Slow tables
  + Fast indexes
  + Fast growing indexes
  + Read only
  + Default
  + Temporary
* Set up Filegrowth parameter for production database: for data file it should be 512-1024 Mb and for Log file it should be 2048-4096 Mb.

# Backups

Be sure that you have backup plan for your databases, including system databases like msdb, tempdb, etc.

# Index maintenance

Be sure that you have index maintenance plan for your database. Try to rebuild/reorganize indexes as often as your data is changes. Sometimes it will be OK, when index maintenance is weekly scheduled. But sometimes data changing too fast, and you have to think about more often index maintenance.

The same recommendations for statistics.

You could use free scripts like this: <https://ola.hallengren.com/sql-server-index-and-statistics-maintenance.html> or create your own script.

# Configure Instant File Initialization for SQL Server

More information here: <http://www.brentozar.com/archive/2013/09/five-sql-server-settings-to-change/>