Database configuration

Enterprise applications

NetApp
January 12, 2026

Table of Contents

Database configuration . . . 1
- CPU configuration . . . 1
  - Hyper-threading . . . 1
  - Cores and licensing . . . 1
  - CPU affinity . . . 2
  - Max Degree of Parallelism (MAXDOP) . . . 3
  - Max worker threads . . . 3
- Memory configuration . . . 4
  - Max server memory . . . 5
  - Nonuniform memory access . . . 6
  - Index create memory . . . 6
  - Min memory per query . . . 6
- Shared instance versus dedicated instance . . . 7
- Tempdb files . . . 8

Database configuration

CPU configuration

SQL Server performance has multiple dependencies on the CPU and core configuration.

Hyper-threading

Hyper-threading is refers to either simultaneous multithreading (SMT) implementation, which improves parallelization of computations performed on x86 processors. SMT is available on both Intel and AMD processors.

Hyper-threading results in logical CPUs that appear as physical CPUs to the operating system. SQL Server then sees the those additional CPUs and uses them as if there are more cores than physically present. This can substantially improve performance by increasing parallelization.

The caveat here is that each SQL Server version has its own limitations on the compute power it can use. For more information, see Compute Capacity Limits by Edition of SQL Server.

Cores and licensing

There are two options for licensing SQL Server. The first is known as a server + client access license (CAL) model; the second is the per processor core model. Although you can access all the product features available in SQL Server with the server + CAL strategy, there is a hardware limit of 20 CPU cores per socket. Even if you have SQL Server Enterprise Edition + CAL for a server with more than 20 CPU cores per socket, the application cannot use all those cores at a time on that instance.

The picture below shows the SQL Server log message after startup indicating the enforcement of the core limit.

```
2017-01-11 07:16:30.71 Server      Microsoft SQL Server 2016
(RTM) - 13.0.1601.5 (X64)
     Apr 29 2016 23:23:58
     Copyright (c) Microsoft Corporation
     Enterprise Edition (64-bit) on Windows Server 2016
Datacenter 6.3 <X64> (Build 14393: )

2017-01-11 07:16:30.71 Server      UTC adjustment: -8:00
2017-01-11 07:16:30.71 Server      (c) Microsoft Corporation.
2017-01-11 07:16:30.71 Server      All rights reserved.
2017-01-11 07:16:30.71 Server      Server process ID is 10176.
2017-01-11 07:16:30.71 Server      System Manufacturer:
'FUJITSU', System Model: 'PRIMERGY RX2540 M1'.
2017-01-11 07:16:30.71 Server      Authentication mode is MIXED.
2017-01-11 07:16:30.71 Server      Logging SQL Server messages
in file 'C:\Program Files\Microsoft SQL Server
\MSSQL13.MSSQLSERVER\MSSQL\Log\ERRORLOG'.
2017-01-11 07:16:30.71 Server      The service account is 'SEA-
TM\FUJIA2R30$'. This is an informational message; no user action
is required.
2017-01-11 07:16:30.71 Server      Registry startup parameters:
     -d C:\Program Files\Microsoft SQL Server
\MSSQL13.MSSQLSERVER\MSSQL\DATA\master.mdf
     -e C:\Program Files\Microsoft SQL Server
\MSSQL13.MSSQLSERVER\MSSQL\Log\ERRORLOG
     -l C:\Program Files\Microsoft SQL Server
\MSSQL13.MSSQLSERVER\MSSQL\DATA\mastlog.ldf
     -T 3502
     -T 834
2017-01-11 07:16:30.71 Server      Command Line Startup
Parameters:
     -s "MSSQLSERVER"
2017-01-11 07:16:30.72 Server      SQL Server detected 2 sockets
with 18 cores per socket and 36 logical processors per socket,
72 total logical processors; using 40 logical processors based
on SQL Server licensing. This is an informational message; no
user action is required.
2017-01-11 07:16:30.72 Server      SQL Server is starting at
```

Therefore, to use all CPUs, you should use the per-processor core license. For detailed information about SQL Server licensing, see SQL Server 2022: Your modern data platform.

CPU affinity

You are unlikely to need to alter the processor affinity defaults unless you encounter performance problems, but it is still worth understanding what they are and how they work.

SQL Server supports processor affinity by two options:

- CPU affinity mask
- Affinity I/O mask

SQL Server uses all CPUs available from the operating system (if the per-processor core license is chosen). It also creates schedulers f0r each CPU to make best use of the resources for any given workload. When multitasking, the operating system or other applications on the server can switch process threads from one processor to another. SQL Server is a resource-intensive application, and performance can be affected when this occurs. To minimize the impact, you can configure the processors so that all of the SQL Server load is directed to a preselected group of processors. This is achieved by using the CPU affinity mask.

The affinity I/O mask option binds SQL Server disk I/O to a subset of CPUs. In SQL Server OLTP environments, this extension can significantly enhance the performance of SQL Server threads issuing I/O

operations.

Max Degree of Parallelism (MAXDOP)

By default, SQL Server uses all available CPUs during query execution if the per-processor core license chosen.

Although this is helpful for large queries, it can cause performance problems and limit concurrency. A better approach is to limit parallelism to the number of physical cores in a single CPU socket. For example, on a server with two physical CPU sockets with 12 cores per socket, regardless of hyper-threading, `MAXDOP` should be set to 12. `MAXDOP` cannot restrict or dictate which CPU is to be used. Instead, it restricts the number of CPUs that can be used by a single batch query.

NetApp recommends for DSS such as data warehouses, start with `MAXDOP` at 50 and explore tuning up or down if required. Make sure you measure the critical queries in your application when making changes.

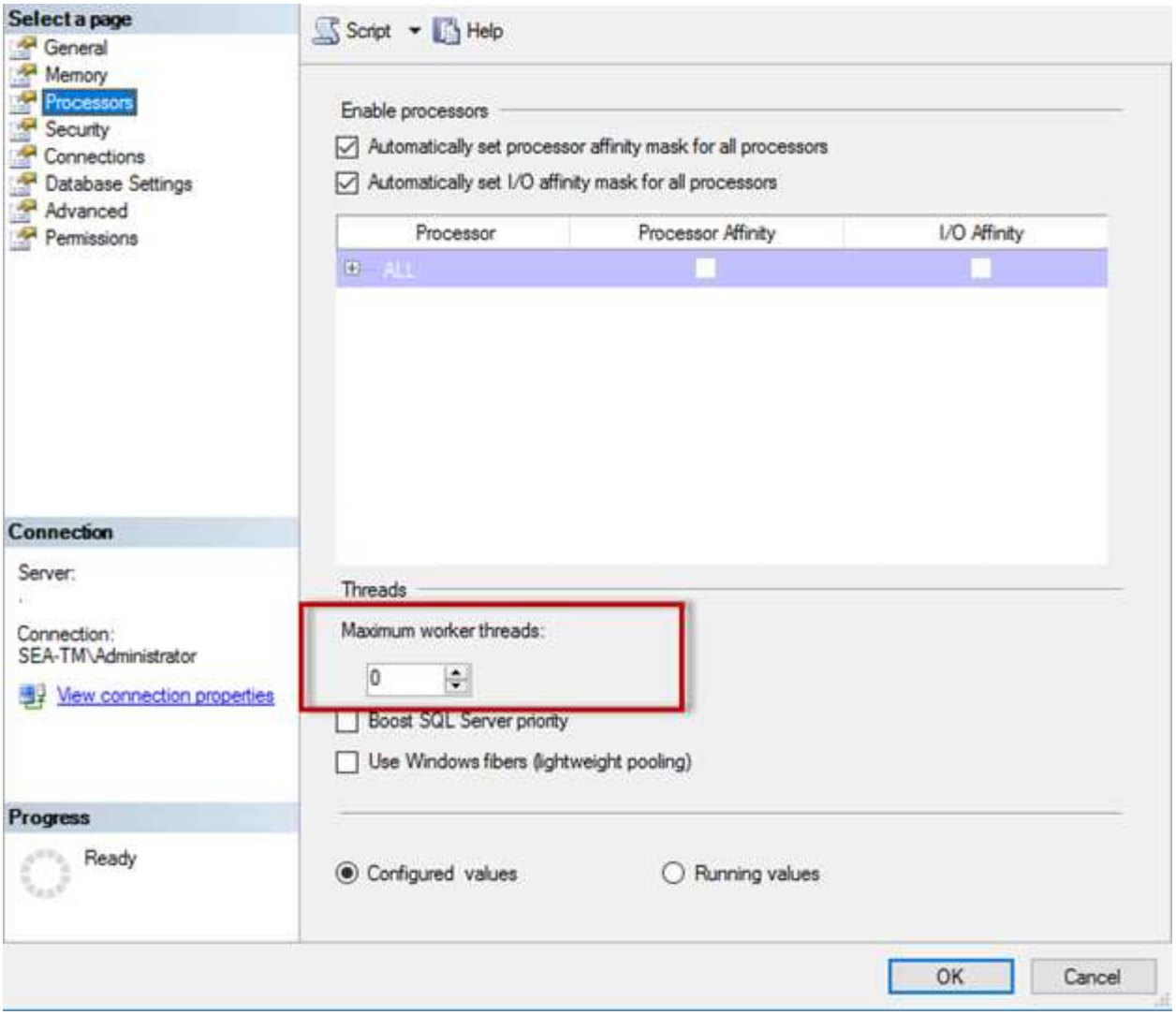
Max worker threads

The max worker threads option helps to optimize performance when large numbers of clients are connected to SQL Server.

Normally, a separate operating system thread is created for each query. If hundreds of simultaneous connections are made to SQL Server, then one-thread-per-query configuration can consume excessive system resources. The `max worker threads` option helps improve performance by enabling SQL Server to create a pool of worker threads that can collectively service a larger number of query requests.

The default value is 0, which allows SQL Server to automatically configure the number of worker threads at startup. This works for most systems. Max worker threads is an advanced option and should not be altered without assistance from an experienced database administrator (DBA).

When should you configure SQL Server to use more worker threads? If the average work queue length for each scheduler is above 1, you might benefit from adding more threads to the system, but only if the load is not CPU-bound or experiencing any other heavy waits. If either of those is happening, adding more threads does not help because they end up waiting for other system bottlenecks. For more information about max worker threads, see Configure the max worker threads server configuration option.



Configuring max worker threads using SQL Server Management Studio.

The following example shows how to configure the max work threads option using T-SQL.

```
EXEC sp_configure 'show advanced options', 1;
GO
RECONFIGURE ;
GO
EXEC sp_configure 'max worker threads', 900 ;
GO
RECONFIGURE;
GO
```

Memory configuration

The following section explains the SQL Server memory settings required to optimize database performance.

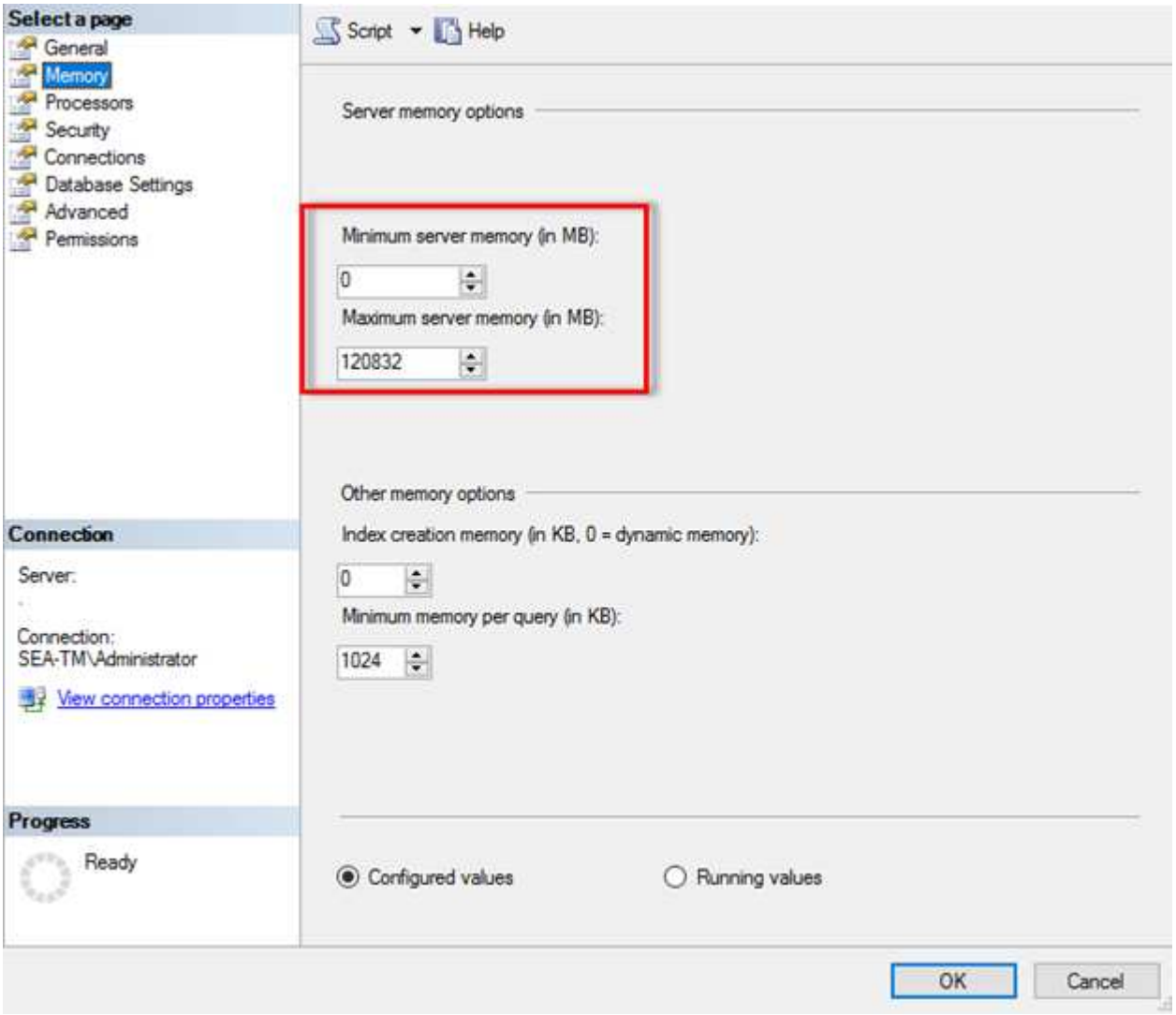
Max server memory

The max server memory option sets the maximum amount of memory that the SQL Server instance can use. It is generally used if multiple applications are running on the same server where SQL Server is running and you want to guarantee that these applications have sufficient memory to function properly.

Some applications only use whatever memory is available when they start and do not request additional memory, even if they are under memory pressure. That is where the max server memory setting comes into play.

On a SQL Server cluster with several SQL Server instances, each instance could be competing for resources. Setting a memory limit for each SQL Server instance can help guarantee best performance for each instance.

NetApp recommends leaving at least 4GB to 6GB of RAM for the operating system to avoid performance issues.



Adjusting minimum and maximum server memory using SQL Server Management Studio.

Using SQL Server Management Studio to adjust minimum or maximum server memory requires a restart of the SQL Server service. You can also adjust server memory using transact SQL (T-SQL) using this code:

```
EXECUTE sp_configure 'show advanced options', 1
GO
EXECUTE sp_configure 'min server memory (MB)', 2048
GO
EXEC sp_configure 'max server memory (MB)', 120832
GO
RECONFIGURE WITH OVERRIDE
```

Nonuniform memory access

Nonuniform memory access (NUMA) is a memory-access optimization technology that helps avoid extra the load on the processor bus.

If NUMA is configured on a server where SQL Server is installed, no additional configuration is required because SQL Server is NUMA-aware and performs well on NUMA hardware.

Index create memory

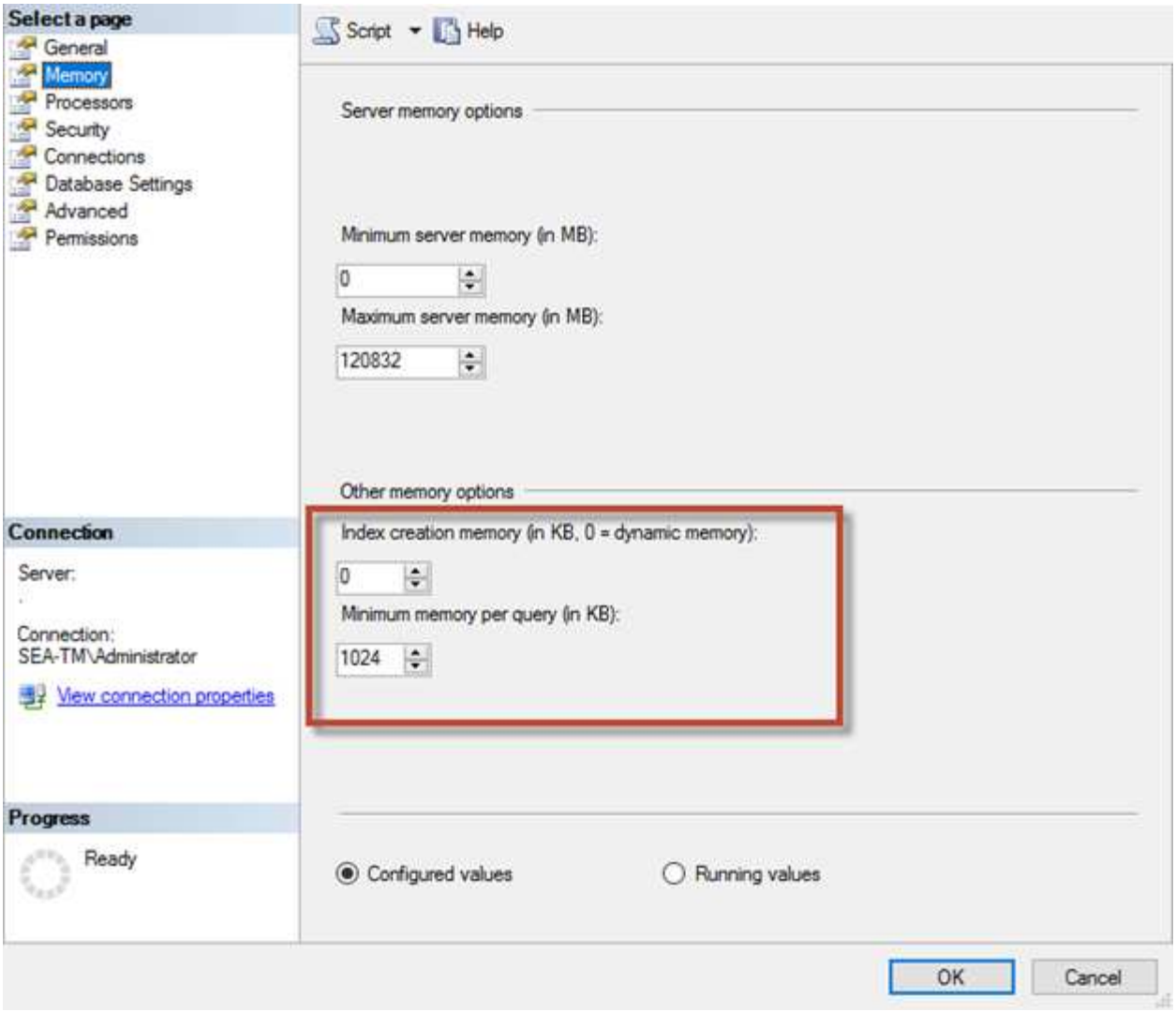
The index create memory option is another advanced option that should not normally need to be changed from defaults.

It controls the maximum amount of RAM initially allocated for creating indexes. The default value for this option is 0, which means that it is managed by SQL Server automatically. However, if you encounter difficulties creating indexes, consider increasing the value of this option.

Min memory per query

When a query is run, SQL Server tries to allocate the optimum amount of memory to run efficiently.

By default, the min memory per query setting allocates >= to 1024KB for each query to run. It is a best practice is to leave this setting at the default value in order to allow SQL Server to dynamically manage the amount of memory allocated for index creation operations. However, if SQL Server has more RAM than it needs to run efficiently, the performance of some queries can be boosted if you increase this setting. Therefore, as long as memory is available on the server that is not being used by SQL Server, any other applications, or the operating system, then boosting this setting can help overall SQL Server performance. If no free memory is available, increasing this setting might hurt overall performance.



Shared instance versus dedicated instance

SQL Server can be configured as a single instance per server or as multiple instances. The right decision usually depends on factors such as whether the server is to be used for production or development, whether the instance is considered critical to business operations and performance goals.

Shared instance configurations may be initially easier to configure, but it can lead to problems where resources become divided or locked, which in turn causes performance issues for other applications that have databases hosted on the shared SQL Server instance.

Troubleshooting performance issues can be complicated because you must figure out which instance is the root cause. This question is weighed against the costs of operating system licenses and SQL Server licenses. If application performance is paramount, then a dedicated instance is highly recommended.

Microsoft licenses SQL Server per core at the server level and not per instance. For this reason, database administrators are tempted to install as many SQL Server instances as the server can handle to save on licensing costs, which can lead to major performance issues later.

NetApp recommends choosing dedicated SQL Server instances whenever possible to obtain optimal performance.

Tempdb files

The Tempdb database can be heavily utilized. In addition to optimal placement of user database files on ONTAP, tempdb datafiles placement is also critical to reduce allocation contention. Tempdb should be placed on separate disk and not shared with user datafiles.

Page contention can occur on the global allocation map (GAM), shared global allocation map (SGAM), or page free space (PFS) pages when SQL Server must write to special system pages to allocate new objects. Latches lock these pages in memory. On a busy SQL Server instance, it can take a long time to get a latch on a system page in tempdb. This results in slower query run times and is known as latch contention. See the following best practices for creating tempdb data files:

- For < or = to 8 cores: tempdb data files = number of cores
- For > 8 cores: 8 tempdb data files
- The tempdb datafile should be created with equal size

The following example script modifies tempdb by creating eight tempdb files of equal size and moving tempdb to the mount point `C:\MSSQL\tempdb` for SQL Server 2012 and later.

```
use master

go

-- Change logical tempdb file name first since SQL Server shipped with
logical file name called tempdev

alter database tempdb modify file (name = 'tempdev', newname =
'tempdev01');



-- Change location of tempdev01 and log file

alter database tempdb modify file (name = 'tempdev01', filename =
'C:\MSSQL\tempdb\tempdev01.mdf');

alter database tempdb modify file (name = 'templog', filename =
'C:\MSSQL\tempdb\templog.ldf');

GO

-- Assign proper size for tempdev01
```

```
ALTER DATABASE [tempdb] MODIFY FILE ( NAME = N'tempdev01', SIZE = 10GB );

ALTER DATABASE [tempdb] MODIFY FILE ( NAME = N'templog', SIZE = 10GB );

GO

-- Add more tempdb files

ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev02', FILENAME =
N'C:\MSSQL\tempdb\tempdev02.ndf' , SIZE = 10GB , FILEGROWTH = 10%);

ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev03', FILENAME =
N'C:\MSSQL\tempdb\tempdev03.ndf' , SIZE = 10GB , FILEGROWTH = 10%);

ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev04', FILENAME =
N'C:\MSSQL\tempdb\tempdev04.ndf' , SIZE = 10GB , FILEGROWTH = 10%);

ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev05', FILENAME =
N'C:\MSSQL\tempdb\tempdev05.ndf' , SIZE = 10GB , FILEGROWTH = 10%);

ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev06', FILENAME =
N'C:\MSSQL\tempdb\tempdev06.ndf' , SIZE = 10GB , FILEGROWTH = 10%);

ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev07', FILENAME =
N'C:\MSSQL\tempdb\tempdev07.ndf' , SIZE = 10GB , FILEGROWTH = 10%);

ALTER DATABASE [tempdb] ADD FILE ( NAME = N'tempdev08', FILENAME =
N'C:\MSSQL\tempdb\tempdev08.ndf' , SIZE = 10GB , FILEGROWTH = 10%);

GO
```

Beginning with SQL Server 2016, the number of CPU cores visible to the operating system is automatically detected during installation and, based on that number, SQL Server calculates and configures the number of tempdb files required for optimum performance.

Copyright information

Trademark information