# 데이터베이스 구성

## Enterprise applications

NetApp
February 10, 2026

# 목차

# 데이터베이스 구성

## CPU 구성

SQL Server 성능은 CPU 및 코어 구성에 대해 여러 가지 종속성이 있습니다.

### 하이퍼 스레딩

하이퍼스레딩은 X86 프로세서에서 수행되는 계산의 병렬화를 개선하는 SMT(동시 다중 스레딩) 구현을 의미합니다. SMT는 Intel 및 AMD 프로세서 모두에서 사용할 수 있습니다.

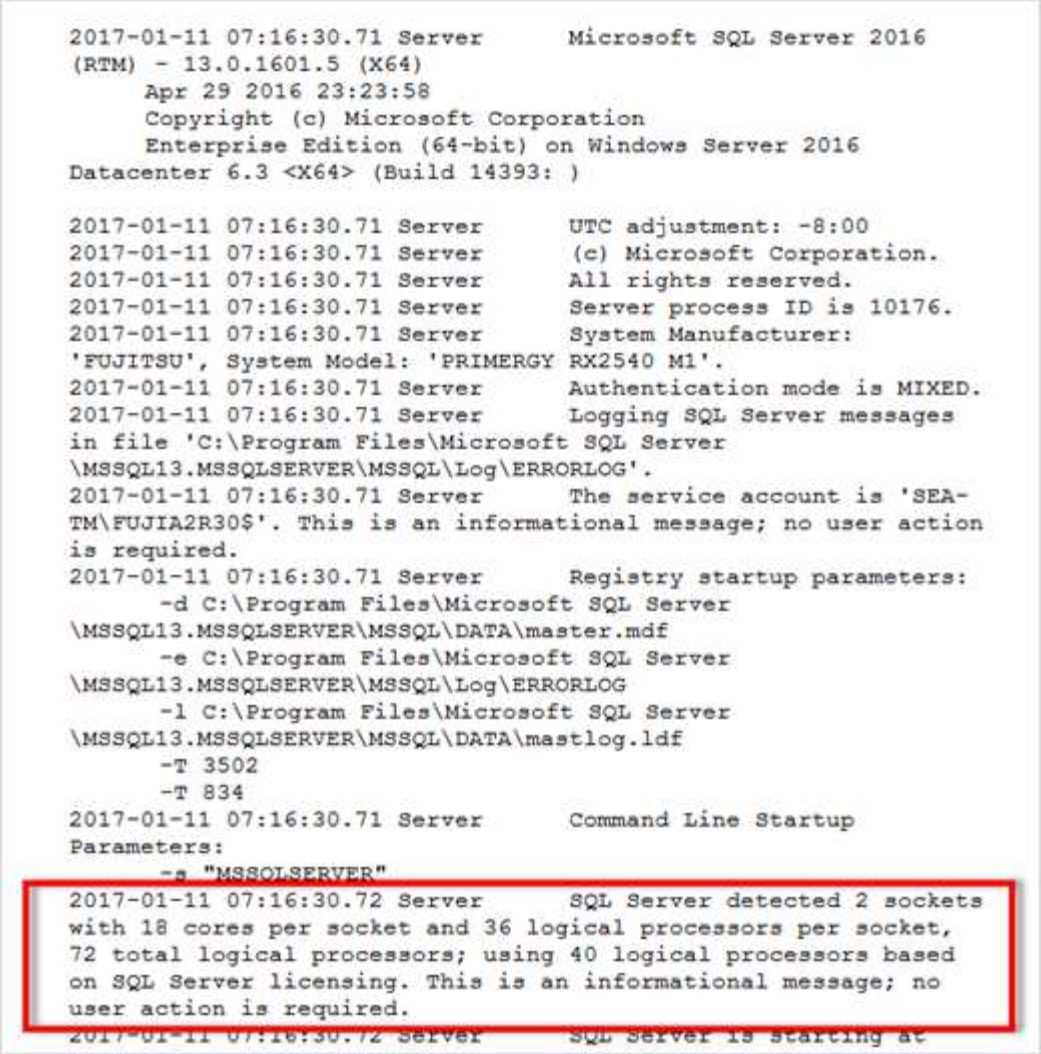
하이퍼 스레딩은 운영 체제에 물리적 CPU로 표시되는 논리 CPU를 생성합니다. 그런 다음 SQL Server는 이러한 추가 CPU를 보고 물리적으로 존재하는 것보다 더 많은 코어가 있는 것처럼 사용합니다. 이를 통해 병렬화를 증가시켜 성능을 크게 향상시킬 수 있습니다.

여기서 주의해야 할 점은 각 SQL Server 버전에는 사용할 수 있는 컴퓨팅 성능에 대한 제한이 있다는 것입니다. 자세한 내용은 을 "SQL Server 버전별 컴퓨팅 용량 제한"참조하십시오.

### 코어 및 라이센싱

SQL Server 라이선스에는 두 가지 옵션이 있습니다. 첫 번째는 서버 + 클라이언트 액세스 라이센스(CAL) 모델이며 두 번째는 프로세서당 코어 모델입니다. 서버 + CAL 전략을 통해 SQL Server에서 사용할 수 있는 모든 제품 기능에 액세스할 수 있지만 소켓당 20개의 CPU 코어로 하드웨어 제한이 있습니다. 소켓당 20개가 넘는 CPU 코어가 있는 서버에 대해 SQL Server Enterprise Edition + CAL을 사용하는 경우에도 응용 프로그램은 해당 인스턴스에서 이러한 모든 코어를 한 번에 사용할 수 없습니다.

아래 그림에서는 시작 후 코어 제한 적용을 나타내는 SQL Server 로그 메시지를 보여 줍니다.




따라서 모든 CPU를 사용하려면 프로세서당 코어 라이센스를 사용해야 합니다. SQL Server 라이센스에 대한 자세한 내용은 을 참조하십시오 "SQL Server 2022: 최신 데이터 플랫폼".

## CPU 선호도

성능 문제가 발생하지 않는 한 프로세서 선호도 기본값을 변경할 필요는 없습니다. 그러나 성능 문제가 무엇이고 어떻게 작동하는지 이해하는 것이 좋습니다.

SQL Server는 다음 두 가지 옵션을 통해 프로세서 선호도를 지원합니다.

- CPU 선호도 마스크
- 선호도 I/O 마스크

SQL Server는 운영 체제에서 사용할 수 있는 모든 CPU를 사용합니다(프로세서당 코어 라이센스를 선택한 경우). 또한 각 CPU에 스케줄러를 생성하여 모든 워크로드를 위한 리소스를 최대한 활용할 수 있습니다. 멀티태스킹 시 서버의 운영 체제 또는 다른 응용 프로그램은 프로세스 스레드를 하나의 프로세서에서 다른 프로세서로 전환할 수 있습니다. SQL Server는 리소스를 많이 사용하는 응용 프로그램이므로 이 경우 성능에 영향을 줄 수 있습니다. 영향을 최소화하기 위해 모든 SQL Server 로드가 미리 선택된 프로세서 그룹으로 전달되도록 프로세서를 구성할 수 있습니다. 이는 CPU 선호도 마스크를 사용하여 수행할 수 있습니다.

선호도 입출력 마스크 옵션은 SQL Server 디스크 입출력을 CPU의 하위 집합에 바인딩합니다. SQL Server OLTP

환경에서 이 확장은 I/O 작업을 실행하는 SQL Server 스레드의 성능을 크게 향상시킬 수 있습니다.

## 최대 평행 거리(MAXDOP)

기본적으로 SQL Server는 프로세서별 코어 라이센스를 선택한 경우 쿼리 실행 중에 사용 가능한 모든 CPU를 사용합니다.

이 방법은 대규모 쿼리에 유용하지만 성능 문제가 발생하고 동시성이 제한될 수 있습니다. 더 좋은 방법은 병렬 처리를 단일 CPU 소켓의 물리적 코어 수로 제한하는 것입니다. 예를 들어, 하이퍼스레딩에 관계없이 소켓당 12개의 코어가 있는 물리적 CPU 소켓 2개가 있는 서버에서 `MAXDOP` 12로 설정해야 합니다. `MAXDOP` 사용할 CPU를 제한하거나 지정할 수 없습니다. 대신 단일 배치 쿼리에서 사용할 수 있는 CPU 수를 제한합니다.

* NetApp는 데이터 웨어하우스와 같은 DSS에 대해 50부터 시작하여 필요한 경우 조정 기능을 위 또는 아래로 탐색할 것을 `MAXDOP` 권장합니다. 변경할 때 응용 프로그램에서 중요한 쿼리를 측정해야 합니다.

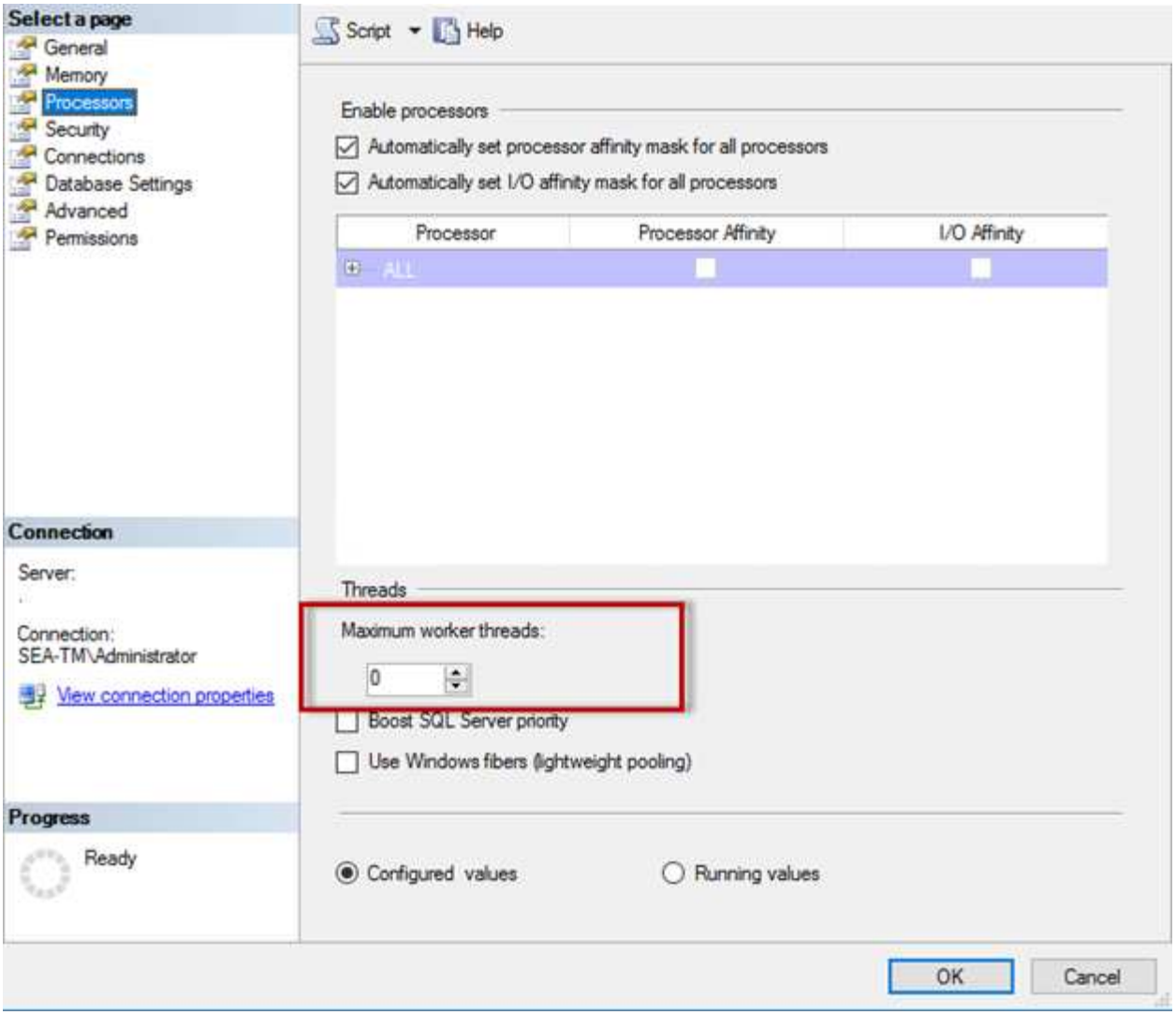
## 최대 작업자 스레드 수

최대 작업자 스레드 옵션을 사용하면 많은 수의 클라이언트가 SQL Server에 연결될 때 성능을 최적화할 수 있습니다.

일반적으로 각 쿼리에 대해 별도의 운영 체제 스레드가 만들어집니다. SQL Server에 수백 개의 동시 연결이 이루어지면 쿼리당 하나의 스레드 구성이 과도한 시스템 리소스를 소모할 수 있습니다. 이 `max worker threads` 옵션을 사용하면 SQL Server에서 더 많은 쿼리 요청을 종합적으로 처리할 수 있는 작업자 스레드 풀을 만들어 성능을 향상시킬 수 있습니다.

기본값은 0입니다. 이 값을 사용하면 SQL Server가 시작 시 작업자 스레드 수를 자동으로 구성할 수 있습니다. 이는 대부분의 시스템에서 작동합니다. Max worker 스레드는 고급 옵션이므로 숙련된 데이터베이스 관리자(DBA)의 도움 없이 변경할 수 없습니다.

작업자 스레드를 더 많이 사용하도록 SQL Server를 구성해야 하는 경우는 언제입니까? 각 스케줄러의 평균 작업 대기열 길이가 1을 초과하면 시스템에 더 많은 스레드를 추가할 수 있습니다. 단, 로드가 CPU에 바인딩되지 않거나 다른 과중한 대기가 발생하는 경우에만 가능합니다. 이러한 상황이 발생하는 경우 스레드를 더 추가해도 다른 시스템 병목 현상이 발생하기 때문에 도움이 되지 않습니다. 최대 작업자 스레드에 대한 자세한 내용은 을 "최대 작업자 스레드 서버 구성 옵션을 구성합니다"참조하십시오.




**SQL Server Management Studio**를 사용하여 최대 작업자 스레드 구성

다음 예제에서는 T-SQL을 사용하여 최대 작업 스레드 옵션을 구성하는 방법을 보여 줍니다.

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

# 메모리 구성

다음 섹션에서는 데이터베이스 성능을 최적화하는 데 필요한 SQL Server 메모리 설정에 대해 설명합니다.

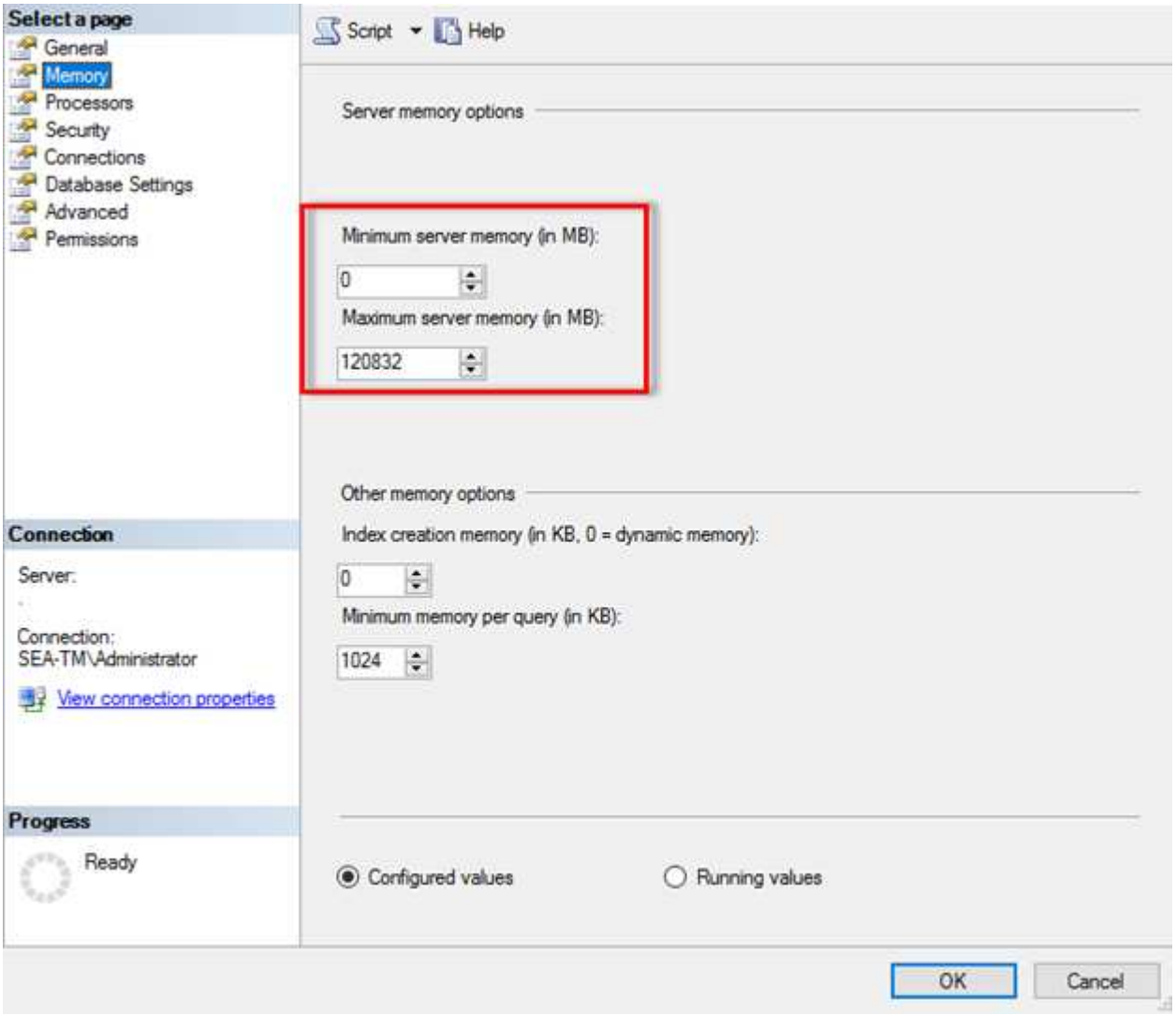
## 최대 서버 메모리

최대 서버 메모리 옵션은 SQL Server 인스턴스가 사용할 수 있는 최대 메모리 양을 설정합니다. 일반적으로 SQL Server가 실행 중인 동일한 서버에서 여러 응용 프로그램이 실행되고 있고 이러한 응용 프로그램이 제대로 작동할 수 있는 충분한 메모리를 확보하려는 경우에 사용됩니다.

일부 응용 프로그램은 시작할 때 사용 가능한 메모리만 사용하고 메모리 압력을 받는 경우에도 추가 메모리를 요청하지 않습니다. 최대 서버 메모리 설정이 재생되는 위치입니다.

SQL Server 인스턴스가 여러 개 있는 SQL Server 클러스터에서 각 인스턴스가 리소스를 놓고 경쟁할 수 있습니다. 각 SQL Server 인스턴스에 대한 메모리 제한을 설정하면 각 인스턴스에 대해 최상의 성능을 보장할 수 있습니다.

*NetApp는 성능 문제를 방지하기 위해 운영 체제에 최소 4GB~6GB의 RAM을 남겨 둘 것을 권장합니다.




**SQL Server Management Studio**를 사용하여 최소 및 최대 서버 메모리 조정

SQL Server Management Studio를 사용하여 최소 또는 최대 서버 메모리를 조정하려면 SQL Server 서비스를 다시 시작해야 합니다. 다음 코드를 사용하여 T-SQL(Transact SQL)을 사용하여 서버 메모리를 조정할 수도 있습니다.

```
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
```

## 비균일 메모리 액세스

NUMA(Non Uniform Memory Access)는 프로세서 버스의 추가 부하를 방지하는 메모리 액세스 최적화 기술입니다.

NUMA가 SQL Server가 설치된 서버에 구성되어 있는 경우 SQL Server가 NUMA를 인식하며 NUMA 하드웨어에서 제대로 작동하기 때문에 추가 구성이 필요하지 않습니다.

## 인덱스가 메모리를 만듭니다

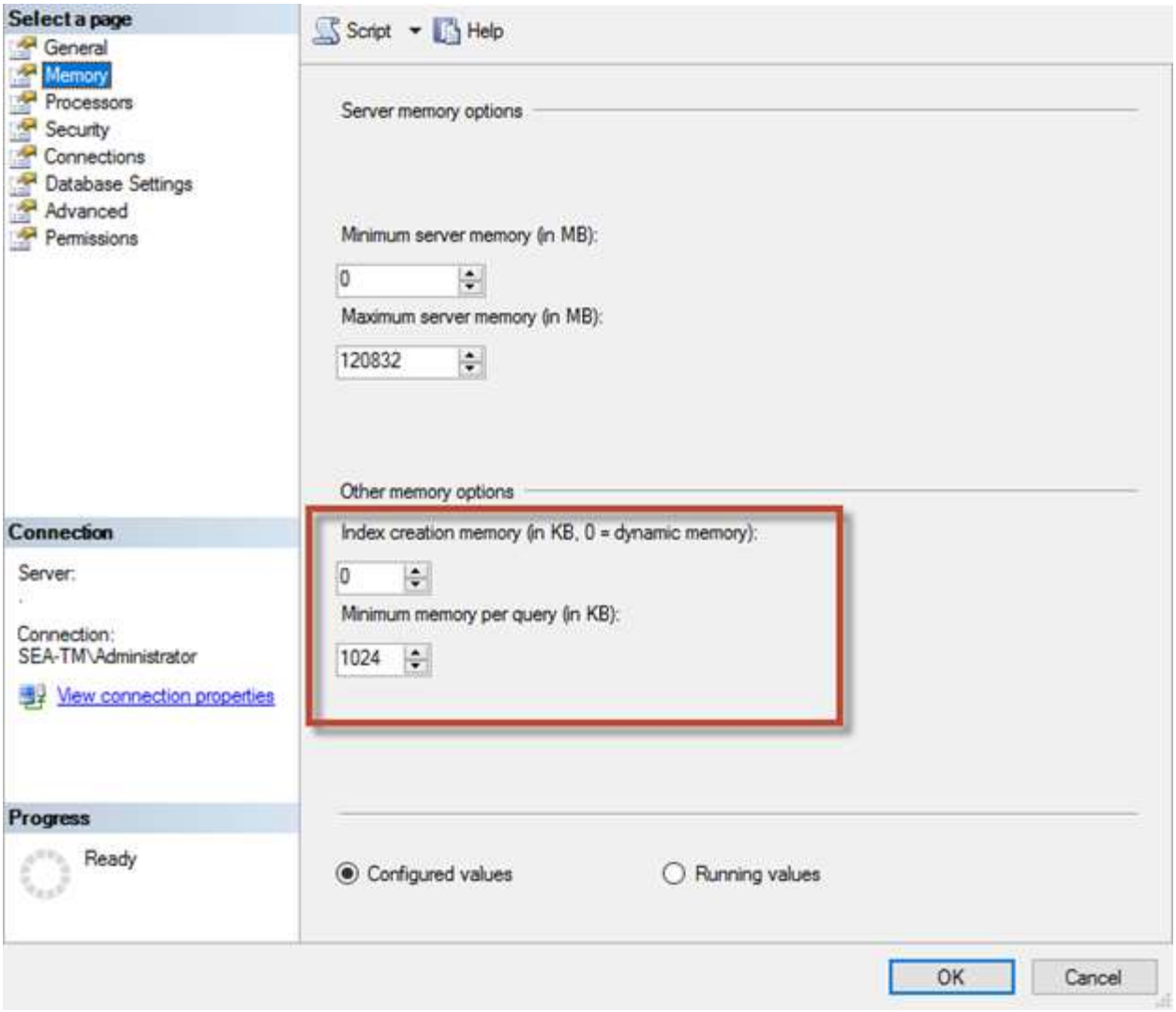
색인 메모리 만들기 옵션은 일반적으로 기본값에서 변경할 필요가 없는 또 다른 고급 옵션입니다.

인덱스 생성을 위해 처음에 할당된 최대 RAM 양을 제어합니다. 이 옵션의 기본값은 0입니다. 즉, 이 옵션은 SQL Server에서 자동으로 관리됩니다. 그러나 인덱스를 만드는 데 문제가 발생하면 이 옵션의 값을 늘리는 것이 좋습니다.

## 쿼리당 최소 메모리

쿼리를 실행하면 SQL Server는 효율적으로 실행할 최적의 메모리 양을 할당하려고 합니다.

기본적으로 쿼리당 최소 메모리 설정은 실행할 각 쿼리에 대해 >=~1024KB를 할당합니다. SQL Server에서 인덱스 생성 작업에 할당된 메모리 양을 동적으로 관리할 수 있도록 이 설정을 기본값으로 두는 것이 좋습니다. 그러나 SQL Server의 RAM이 효율적으로 실행하는 데 필요한 것보다 많은 경우 이 설정을 높이면 일부 쿼리의 성능이 향상될 수 있습니다. 따라서 SQL Server, 다른 응용 프로그램 또는 운영 체제에서 사용하지 않는 서버에서 메모리를 사용할 수 있는 한 이 설정을 높이면 SQL Server의 전반적인 성능이 향상될 수 있습니다. 사용 가능한 메모리가 없는 경우 이 설정을 늘리면 전체 성능이 저하될 수 있습니다.




# 공유 인스턴스와 전용 인스턴스 비교

SQL Server는 서버당 단일 인스턴스 또는 여러 인스턴스로 구성할 수 있습니다. 적절한 결정은 일반적으로 서버가 운영 또는 개발에 사용되는지 여부, 인스턴스가 비즈니스 운영 및 성능 목표에 중요한지를 여부 등의 요인에 따라 달라집니다.

처음에는 공유 인스턴스 구성이 더 쉬울 수 있지만 리소스가 나누어지거나 잠기는 문제가 발생할 수 있으며, 이로 인해 공유 SQL Server 인스턴스에서 데이터베이스가 호스팅되는 다른 응용 프로그램의 성능 문제가 발생할 수 있습니다.

어떤 인스턴스가 근본 원인인지 파악해야 하기 때문에 성능 문제의 해결은 복잡할 수 있습니다. 이 질문은 운영 체제 라이선스와 SQL Server 라이선스 비용을 기준으로 합니다. 애플리케이션 성능이 가장 중요한 경우에는 전용 인스턴스를 사용하는 것이 좋습니다.

Microsoft는 코어당 SQL Server의 라이센스를 인스턴스가 아닌 서버 레벨에서 부여합니다. 이러한 이유로 데이터베이스 관리자는 서버에서 처리할 수 있는 SQL Server 인스턴스를 많이 설치하여 라이센스 비용을 절감하려는 경향이 있으며, 이로 인해 나중에 주요 성능 문제가 발생할 수 있습니다.

*NetApp는 최적의 성능을 얻기 위해 가능한 한 전용 SQL Server 인스턴스를 선택할 것을 권장합니다.

# tempdb 파일

Tempdb 데이터베이스를 많이 활용할 수 있습니다. ONTAP에 사용자 데이터베이스 파일을 최적으로 배치할 수 있을 뿐 아니라 할당 경합을 줄이기 위해 tempdb 데이터 파일을 배치하는 것도 중요합니다. tempdb는 별도의 디스크에 배치해야 하며 사용자 데이터 파일과 공유하지 않아야 합니다.

SQL Server가 새 개체를 할당하기 위해 특수 시스템 페이지에 써야 하는 경우 GAM(전역 할당 맵), SGAM(공유 전역 할당 맵) 또는 PFS(페이지 사용 가능 공간) 페이지에서 페이지 경합이 발생할 수 있습니다. 래치가 이러한 페이지를 메모리에 잠급니다. 사용 중인 SQL Server 인스턴스의 경우 tempdb의 시스템 페이지에 래치가 표시되는 데 시간이 오래 걸릴 수 있습니다. 이로 인해 쿼리 실행 시간이 느려지며 래치 경합이라고 합니다. tempdb 데이터 파일을 생성하는 방법은 다음 Best Practice를 참조하십시오.

- 또는 = 8코어: tempdb 데이터 파일 = 코어 수입니다
- 8코어 이상의 경우: 8tempdb 데이터 파일
- tempdb 데이터 파일은 같은 크기로 만들어야 합니다

다음 예제 스크립트는 크기가 같은 tempdb 파일 8개를 생성하고 tempdb를 SQL Server 2012 이상의 마운트 지점으로 이동하여 tempdb를 `C:\MSSQL\tempdb` 수정합니다.

```
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
```

SQL Server 2016부터 운영 체제에 표시되는 CPU 코어 수가 설치 중에 자동으로 감지되며, 이 수에 따라 SQL Server는 최적의 성능을 위해 필요한 tempdb 파일 수를 계산 및 구성합니다.

## 저작권 정보



## 상표 정보