
SQL Server 2005 성능 문제 해결

본 문서는 다음 위치에 있는 Microsoft TechNet 문서의 기사로부터 작성되었습니다:

■ Troubleshooting Performance Problems in SQL Server 2005

<http://www.microsoft.com/technet/prodtechnol/sql/2005/tsprfprb.mspx>

요약: SQL Server 데이터베이스에서 일시적인 성능 저하를 경험하기란 좀처럼 드문 일입니다. 작업부하를 고려하지 않은 서투른 데이터베이스 설계나 부적절하게 구성된 시스템은 이러한 성능 문제를 유발하는 여러 가지 잠재적인 원인들입니다. 관리자는 이러한 문제를 예방하거나 최소화해야 하며, 문제가 발생할 때는, 그 원인을 진단한 뒤 문제 해결을 위한 적절한 조치를 취해야 합니다. 이 문서는 SQL Server 프로파일러, 시스템 모니터, 그리고 SQL Server 2005의 새로운 동적 관리 뷰 등과 같은 공개된 도구들을 사용해서 공통적인 성능 문제를 진단 및 해결하기 위한 단계적인 지침을 제공합니다.



역자: 김정선(MVP)

- Microsoft SQL Server MVP

- Microsoft TechNet Honor

- 현) 필라넷 DB 사업부 수석 컨설턴트

- 현) 삼성 SDS 멀티캠퍼스 전임 교수

목차

SQL Server 2005 성능 문제 해결.....	1
소개	2
목적	2
방법론	3
리소스 병목	3
리소스 병목 해결을 위한 도구	4
CPU 병목	4
과도한 컴파일과 재컴파일(Recompile)	5
비효율적인 쿼리 계획	10
쿼리 병렬 처리	11
서투른 커서 사용	15
메모리 병목	16
배경	16
메모리 부족 발견	19
메모리 오류에 대한 일반적인 문제 해결 단계	30
메모리 오류	31
I/O 병목	33

문제 해결	36
Tempdb	38
tempdb 디스크 공간 모니터링	39
디스크 공간 문제 해결	40
과도한 DDL 및 할당 작업	45
느린 쿼리	46
차단	47
인덱스 사용량 모니터링	55
결론	58
부록 A: DBCC MEMORYSTATUS 설명	58
부록 B: 차단 스크립트(Blocking Scripts)	58
인덱스 사용 정보 분석	59
Wait states	71

소개

운영 중인 SQL Server 데이터베이스에 대해 일시적인 성능 문제를 경험할 수 있습니다. 그 이유에는 서투른 데이터베이스 설계에서부터 작업부하를 고려하지 않은 부적절한 시스템이 포함될 수 있습니다. 관리자로서 이러한 문제를 예방하거나 최소화하길 원할 것입니다. 문제가 발생할 때는 그 원인을 진단한 뒤 문제 해결을 위한 적절한 조치를 취해야 합니다. 이 책에서 다루는 문제 범위는 Microsoft® 고객 지원 서비스(CSS 혹은 PSS)에서 잠재적인 문제들에 대한 포괄적인 분석 결과 공통적으로 드러난 문제들로 제한합니다. 이 문서는 SQL Server 프로파일러, 시스템 모니터, 그리고 SQL Server 2005의 새로운 동적 관리 뷰 등과 같은 공개된 도구들을 사용해서 공통적인 성능 문제를 진단 및 해결하기 위한 단계적인 지침을 제공합니다.

목적

이 문서의 주된 목적은 공통적인 고객 시나리오로 공개된 도구들을 사용해서 SQL Server 성능 문제의 진단 및 해결을 위한 일반적인 방법론을 제공하는 것입니다.

SQL Server 2005는 지원성에 있어서 장족의 발전을 했습니다. 커널 계층(SQL-OS)이 재 설계되었으며 내부 구조 및 통계적 데이터를 동적 관리 뷰(DMV)를 통해서 관계형 행집합으로 노출합니다. SQL Server 2000에서는 이러한 정보를 sysprocesses 같은 시스템 테이블을 통해서 노출했지만, 때론 내부 구조로부터 관련 정보를 얻기 위해 SQL Server 프로세스 메모리에 대한 물리적인 덤프를 생성하길 원할 때가 있습니다. 그러나 여기에는 두 가지 주된 쟁점이 있습니다. 첫 번째, 덤프의 크기 및 생성 시간으로 인해서 고객이 항상 물리적 덤프를 제공하기란 어렵다는 것이고, 두 번째, 해당 파일을 분석하기 위해 Microsoft로 보내는 것이 그 문제를 진단하는 것보다 더 오래 걸릴 수 있다는 것입니다.

이것이 이 문서의 두 번째 목적입니다. 바로 DMV를 소개하는 것입니다. DMV는 대부분의 경우에 물리적 덤프를 생성하고 분석하는 절차를 생략함으로써 보다 신속하게 진단할 수 있도록 도와줍니다. 이 문서에서는 SQL Server 2000과 SQL Server 2005에서 동일 문제를 어떻게 해결하는지 단계적으로 비교합니다. DMV는 단순하면서도 보다 친숙한 관계형 인터페이스를 통해서 중요한 시스템 정보를 얻을 수 있도록 지원합니다. 이러한 정보들은 잠재적인 문제 상황 발생 시 관리자에게

경고를 주기 위한 목적의 모니터링에도 사용될 수 있습니다. 또한, 이 정보들은 이후의 전문적인 분석을 위해서 정기적으로 수집될 수 있습니다.

방법론

SQL Server 를 느리게 만드는 원인들은 많이 있습니다만, 다음 3 가지 핵심 증상을 통해서 문제의 진단을 시작합니다.

- **리소스 병목:** 이 문서에서는 CPU, 메모리, I/O 병목을 다루며 네트워크 문제는 고려하지 않습니다. 각 리소스 병목에 대한 문제 식별 방법을 기술하고 그 잠재적인 원인들을 차례로 반복합니다.
- **Tempdb 병목:** SQL Server 인스턴트 별 오직 하나의 tempdb 가 존재하므로, 이것이 성능과 디스크 공간의 병목이 될 수 있습니다. 잘못된 응용 프로그램은 과도한 DDL/DML 작업과 용량 소비라는 두 가지 측면에 있어서 tempdb 의 과부하를 유발합니다. 이것은 동일 서버에서 실행 중인 다른 응용 프로그램의 성능 저하 혹은 작업 실패를 유발할 수 있습니다.
- **악성 쿼리:** 기존 쿼리의 성능이 느려지거나 새로운 쿼리가 예상보다 더 느릴 수 있습니다. 여기엔 많은 원인들이 있습니다. 예를 들어:
 - 통계 정보의 변경으로 인해 기존 쿼리가 잘못된 쿼리 계획을 산출.
 - 인덱스 부재로 인해 테이블 스캔을 유발하고 쿼리가 느려짐.
 - 리소스 사용량이 많지 않음에도 불구하고 차단으로 인해 응용 프로그램이 느려짐

잘못 개발된 응용 프로그램이나 스키마 설계 혹은 트랜잭션에 대해 부적절한 격리 수준의 선택이 과도한 차단의 원인이 될 수 있습니다.

이런 증상들의 원인이 반드시 독립적인 것은 아닙니다. 잘못 선택된 쿼리 계획은 시스템 리소스를 과도하게 소비하고 전반적인 성능 저하의 원인이 될 수 있습니다. 따라서 대용량 테이블에 필요한 인덱스가 없거나 혹은 쿼리 최적화 프로그램(Optimizer)이 그 인덱스를 사용하지 않도록 결정한다면 이것이 악성 쿼리를 유발할 뿐만 아니라, SQL Server 가 접근이 필요한 페이지들을 캐시에 저장하기 위해서 결국 불필요한 데이터 페이지들까지 읽어 들이게 되고 이를 메모리(Buffer Pool)에 뒹군으로써 I/O 하위 시스템의 막대한 부하를 주게 됩니다. 마찬가지로, 자주 실행되는 쿼리에 과도한 재컴파일도 CPU 에 부하를 줍니다.

리소스 병목

다음 절에서는 CPU, 메모리, I/O 하위 시스템 리소스들이 어떻게 병목이 되는지에 대해서 논의할 것입니다. (네트워크 문제는 제외합니다.) 각 리소스 병목에 대해서 그 문제를 식별하는 방법과 발생 가능한 원인들에 대해서 차례로 반복합니다. 예를 들어, 메모리 병목은 과도한 페이지ング을 유발해서 결국 성능에 영향을 미치게 됩니다.

리소스 병목이 있는지를 판단하기 전에, 정상적인 상황에서 리소스가 어떻게 사용되고 있는지 알아야 합니다. 이 문서에서 개략적으로 소개하는 방법들을 사용해서 리소스 사용에 대한 기준 정보를 수집할 수 있습니다.

결국은 용량을 거의 다 소비하고 있는 리소스가 문제이며 현재 구성으로는 그 모든 작업 부하를 지원할 수 없다는 것을 알게 됩니다. 이러한 문제를 처리하기 위해서, 더 강력한 프로세스, 메모리, 혹은 I/O 나

네트워크 채널의 대역폭을 증가시킬 필요가 있을 것입니다. 그러나, 이러한 절차를 밟기 전에 리소스 병목에 대한 몇 가지 공통적인 원인들을 이해하는 것이 도움이 됩니다. 여기엔 서버 구성 정보 변경과 같이 부가적인 리소스를 추가할 필요가 없는 해결 방안들이 있습니다.

리소스 병목 해결을 위한 도구

특정 리소스 병목을 해결하기 위해 다음과 같은 도구들을 사용합니다.

- **시스템 모니터 (PerfMon):** Windows 구성요소로 제공되는 도구입니다. 보다 자세한 정보는 시스템 모니터 문서를 참조하십시오.
- **SQL Server 프로파일러:** SQL Server 2005 프로그램 그룹 내의 성능 도구 그룹의 **SQL Server Profiler** 를 참조하십시오.
- **DBCC 명령:** 자세한 내용은 SQL Server 온라인 설명서와 본 문서의 부록 A 를 참조하십시오.
- **DMV:** 자세한 내용은 본 문서와 함께 SQL Server 온라인 설명서를 참조하십시오.

CPU 병목

CPU 병목은 서버의 다른 부하가 없는 상황에서, 불시에 발생할 수 있습니다. 공통적인 원인들로는 최적화되지 않은 쿼리 계획, 잘못된 서버 구성, 설계 문제, 그리고 부족한 하드웨어 리소스 등을 들 수 있습니다. 보다 빠른 혹은 더 많은 프로세스를 성급하게 구매하기 보다는, 먼저 CPU 대역폭을 가장 많이 소비하는 주범이 누구인지를 식별하고 튜닝이 가능한지를 확인해야 합니다.

서버가 CPU 집중적인지를 판단하는 가장 좋은 방법은 시스템 모니터입니다. **Processor: %Processor Time** 카운터가 높은지를 살펴봐야 합니다. CPU 당 프로세스 시간이 80%를 초과하는 경우 일반적으로 병목이라고 간주합니다. 또는 sys.dm_os_schedulers 뷰를 사용해서 실행 가능한 작업 수(runnable_tasks_count)가 일반적으로 0 이 아닌지를 살펴봄으로써 SQL Server 스케줄러를 모니터 할 수 있습니다. 0 이 아닌 값은 해당 작업이 실행하기 위해서 대기함을 나타냅니다; 이 카운터가 높은 값이면 CPU 병목을 나타내는 증상입니다. 다음 쿼리를 사용해서 모든 스케줄러를 나열하고 실행 가능한 작업 수를 볼 수 있습니다.

```
select
    scheduler_id,
    current_tasks_count,
    runnable_tasks_count
from
    sys.dm_os_schedulers
where
    scheduler_id < 255
```

다음 쿼리는 캐시된 일괄처리(batch)나 프로시저 중에서 CPU 를 가장 많이 소비하는 대상을 알 수 있습니다. 이 쿼리는 동일 plan_handle(동일 일괄 처리나 프로시저의 일부임을 의미)을 기준으로 해당 구문에서 소비한 전체 CPU 사용량을 집계합니다. 만일 plan_handle 을 기준으로 하나 이상의 구문을 포함한다면, CPU 를 가장 많이 소비하는 문제 쿼리를 식별하기 위해서 보다 상세하게 접근하게 될 것입니다.

```
select top 50
    sum(qs.total_worker_time) as total_cpu_time,
    sum(qs.execution_count) as total_execution_count,
    count(*) as number_of_statements,
    qs.plan_handle
from
    sys.dm_exec_query_stats qs
group by qs.plan_handle
order by sum(qs.total_worker_time) desc
```

이 절의 나머지 부분에서는 공통적인 CPU-집약적 작업들의 내용과, 그러한 문제를 찾아서 해결하는 효율적인 방법들에 대해서 논의합니다.

과도한 컴파일과 재컴파일(Recompile)

일괄 처리(Batch)나 원격 프로시저 호출(RPC)이 SQL Server 로 전송되면 서버는 그것을 실행하기 전에 먼저 해당 쿼리 계획에 대한 적합성과 정확성을 검사합니다. 그 중 하나가 실패하면, 일괄 처리는 다른 쿼리 계획을 산출하기 위해서 다시 컴파일 해야만 합니다. 그러한 컴파일을 *재컴파일(recompilation)*이라고 합니다. 재컴파일은 일반적으로 정확성을 보장하기 위해서 필요하며 데이터 변경으로 인해서 서버가 보다 최적화된 쿼리 계획을 산출할 수 있다고 판단하는 경우에 수행됩니다. 컴파일은 본래 CPU 집약적이므로 과도한 재컴파일은 결과적으로 CPU-집중형 성능 문제의 원인이 됩니다.

SQL Server 2000 에서는, 저장 프로시저를 재컴파일 할 때 재컴파일을 유발시킨 해당 구문만이 아니라 저장 프로시저 전체를 재컴파일합니다. SQL Server 2005 는 구문 단위 재컴파일을 도입했습니다. SQL Server 2005 가 저장 프로시저를 재컴파일 하게 되면, 프로시저 전체가 아니라 해당 구문만을 컴파일 합니다. 이러한 동작은 CPU 대역폭을 보다 적게 사용함으로써 COMPILE 잠금과 같은 리소스 상에 충돌을 줄여줍니다. 재컴파일의 원인은 다음과 같습니다:

- 스키마 변경
- 정보 변경
- 지연된 컴파일
- SET 옵션 변경
- 임시 테이블 변경
- OPTION(RECOMPILE) 쿼리 힌트 사용¹

발견

과도한 컴파일과 재컴파일을 찾아내기 위해서 시스템 모니터(PerfMon) 혹은 SQL Trace(SQL Server 프로파일러)를 사용할 수 있습니다.

시스템 모니터 (Perfmon)

¹ 원문에서는 RECOMPILE 쿼리 힌트를 추가로 언급하고 있으나 온라인 설명서, 프로파일러 등을 참조해 보면 OPTION(RECOMPILE)과 같은 것으로 판단됩니다.

SQL Statistics 개체가 SQL Server 인스턴스에 보낸 요청의 유형과 컴파일을 모니터하기 위한 카운터를 제공합니다. 컴파일이 높은 CPU 사용의 원인인지를 알아내기 위해서는 수신된 일괄 처리 수와 함께 관련된 쿼리 컴파일 및 재컴파일 수를 모니터합니다. 이상적으로는, 사용자가 임시(ad hoc) 쿼리를 전송하지 않는 한 **Batch Requests/sec** 대비 **SQL Recompilations/sec** 비율은 아주 낮아야 합니다.

핵심 카운터는 다음과 같습니다.

- SQL Server: **SQL Statistics: Batch Requests/sec**
- SQL Server: **SQL Statistics: SQL Compilations/sec**
- SQL Server: **SQL Statistics: SQL Recompilations/sec**

보다 자세한 정보는, SQL Server 온라인 설명서의 “SQL Statistics 개체“를 참조하십시오.

SQL Trace

PerfMon 카운터가 높은 재컴파일 수치를 나타낸다면, 재컴파일이 SQL Server 로 하여금 많은 CPU 소비하게 만드는 원인이 될 수 있습니다. 이제 프로파일러 추적을 이용해서 재컴파일이 발생하는 해당 저장 프로시저를 찾아야 합니다. SQL Server 프로파일러 추적은 재컴파일 원인을 포함한 추가 정보를 제공합니다. 이러한 정보를 얻기 위해서 다음 이벤트를 사용합니다.

SP: Recompile / SQL: StmtRecompile. SP: Recompile 과 **SQL: StmtRecompile** 이벤트 클래스는 재컴파일된 해당 저장 프로시저 및 문장(Statement)을 나타냅니다. 저장 프로시저를 컴파일 할 때는, 저장 프로시저에 대해 하나의 이벤트가 발생하고 컴파일된 각 문장에 대해 하나가 발생합니다. 그러나, 저장 프로시저를 재컴파일할 때는, 재컴파일을 유발하는 해당 문장만이 재컴파일 됩니다(SQL Server 2000 에서와 같이 저장 프로시저가 전체가 아니라). **SP: Recompile** 이벤트 클래스에 대한 몇 가지 중요한 데이터 열은 아래 목록에 있습니다. 특히 **EventSubClass** 데이터 열은 재컴파일하는 이유를 파악하는데 있어서 중요합니다. **SP: Recompile** 은 재컴파일되는 프로시저나 트리거에 대해 한 번 발생되며 재컴파일이 될 수도 있는 임시 일괄 처리에 대해서는 발생되지 않습니다. SQL Server 2005 에서는 **SQL: StmtRecompile** 를 모니터하는 것이 보다 유용합니다. 이 이벤트 클래스는 모든 유형의 일괄 처리, 임시 쿼리, 저장 프로시저, 트리거가 재컴파일 될 때 발생합니다.

핵심 데이터 열은 다음과 같습니다.

- EventClass
- EventSubClass
- ObjectID (해당 문장을 포함한 저장 프로시저를 의미)
- SPID
- StartTime
- SqlHandle
- TextData

보다 자세한 정보는, SQL Server 온라인 설명서의 “SQL: StmtRecompile 이벤트 클래스“를 참조하십시오.

만일 저장해둔 추적 파일이 있다면, 다음 쿼리를 사용해서 캡처된 재컴파일 이벤트를 모두 볼 수 있습니다.

```
select
    spid,
    StartTime,
    Textdata,
    EventSubclass,
    ObjectID,
    DatabaseID,
    SQLHandle
from
    fn_trace_gettable ( 'e:\recompiletrace.trc' , 1)
where
    EventClass in (37,75,166)
```

참고². EventClass 값과 해당 이벤트

37 = Sp:Recompile, 75 = CursorRecompile, 166=SQL:StmtRecompile

SqlHandle 과 ObjectID 열 혹은 다른 열을 기준으로 쿼리 결과를 그룹화하면, 재컴파일의 주범이 어떤 프로시저인지 알아내거나 혹은 다른 이유(예를 들어 SET 옵션 변경) 때문인지 알 수 있습니다.

Showplan XML For Query Compile. Showplan XML For Query Compile

이벤트 클래스는 Microsoft SQL Server 가 SQL 문장을 컴파일 또는 재컴파일할 때 발생합니다. 이 이벤트에는 컴파일 또는 재컴파일되는 해당 문장에 대한 정보가 있습니다. 이 정보에는 해당 쿼리 계획과 프로시저의 개체 ID 를 포함합니다. 이 이벤트는 각각의 컴파일 혹은 재컴파일에 대해서 캡처가 되므로 상당한 성능 부하를 가집니다. 시스템 모니터에서 **SQL Compilations/sec** 카운터가 높은 값을 보인다면, 이 이벤트를 모니터합니다. 이 정보를 통해서, 어떤 문장이 자주 재컴파일되는지 알 수 있으며 해당 문장의 파라미터를 변경하는데 사용할 수 있으므로 재컴파일 수를 줄일 수 있습니다.

DMV. sys.dm_exec_query_optimizer_info DMV 를 사용하면, SQL Server 가 최적화에 소비하는 시간에 대한 좋은 아이디어를 얻을 수 있습니다. 이 DMV 를 일정 간격을 두고 두 번 실행해서 그 결과를 비교하면, 해당 기간 동안 최적화에 소비한 시간에 대한 좋은 정보를 얻을 수 있습니다.

```
select *
from sys.dm_exec_query_optimizer_info
```

counter	occurrence	value
optimizations	81	1.0
elapsed time	81	6.4547820702944486E-2

특히 elapsed time ³값은 최적화로 인해서 경과된 시간을 나타냅니다.

² 원문에 별다른 언급이 없어서, 역자 임의로 추가합니다.

일반적으로 최적화 동안의 elapsed time 은 최적화를 위해서 사용된 CPU 시간에 가깝습니다(최적화 절차가 매우 CPU 집약적이므로), 따라서 컴파일 시간이 CPU 사용에 미치는 정도에 대한 좋은 측정 방법이 됩니다.

이러한 정보를 캡처하는데 유용한 다른 DMV 로는 **sys.dm_exec_query_stats** 가 있습니다.

관심있게 볼 데이터 열은 다음과 같습니다:

- Sql_handle
- Total worker time
- Plan generation number
- Statement Start Offset

자세한 정보는, SQL Server 온라인 설명서에서 **sys.dm_exec_query_stats** 항목을 참조하십시오.

특히, **plan_generation_num** 은 쿼리 재컴파일 횟수를 나타냅니다. 다음 예제 쿼리는 재컴파일 횟수를 기준으로 상위 25 건의 저장 프로시저를 보여줍니다.

```
select *
from sys.dm_exec_query_optimizer_info

select top 25
    sql_text.text,
    sql_handle,
    plan_generation_num,
    execution_count,
    dbid,
    objectid
from
    sys.dm_exec_query_stats a
    cross apply sys.dm_exec_sql_text(sql_handle) as sql_text
where
    plan_generation_num >1
order by plan_generation_num desc
```

추가 정보를 위해서, Microsoft TechNet 에 [Batch Compilation, Recompilation, and Plan Caching Issues in SQL Server 2005](http://www.microsoft.com/technet/prodtechnol/sql/2005/recomp.msp) (<http://www.microsoft.com/technet/prodtechnol/sql/2005/recomp.msp>) 를 참조하십시오.

해결 방안

과도한 컴파일 및 재컴파일이 문제임을 확인했다면, 다음 옵션들을 고려할 수 있습니다.

- 재컴파일의 원인이 SET 옵션 변경에 있다면, SQL Server 프로파일러를 사용해서 어떤 SET 옵션이 변경되는지 확인하십시오. 저장 프로시저 내에서 SET 옵션의 변경은 피하셔야 합니다. 연결 시의 설정하는 것이

³ sys.dm_exec_query_optimizer_info DMV 에는 원문에 언급된 두 가지 카운터 이외에 상당히 많은 카운터 항목들이 제공됩니다. 오히려 더 유용한 정보들이 많이 있습니다. 반드시 살펴보시길 권장합니다.

더 좋습니다. SET 옵션은 서버에 연결하고 있는 동안에는 변경되지 않도록 하십시오.

- 임시 테이블에 대한 재컴파일 임계값은 일반 테이블보다 더 낮습니다. 임시 테이블에 재컴파일 원인이 통계 정보 변경 때문이라면, 임시 테이블 대신에 테이블 변수를 사용할 수 있습니다. 테이블 변수의 카디널리티⁴ 변경은 재컴파일을 유발하지 않습니다. 이 접근 방법의 문제점은 테이블 변수의 경우 통계정보를 만들거나 보존하지 않기 때문에 최적화 프로그램이 테이블 변수의 카디널리티를 지속적으로 추적하지 않는다는 것입니다. 이것이 결국은 최적화되지 못한 쿼리 계획을 만들게 됩니다. 다른 옵션들을 시험해 보고 최선의 방법을 선택합니다.

또 다른 옵션은 **KEEP PLAN** 쿼리 힌트를 사용하는 것입니다. 이 옵션은 임시 테이블에 대한 낮은 재컴파일 임계값을 일반 테이블과 동일하게 설정합니다. **EventSubclass** 열이 “Statistics changed” 를 나타내는 것을 볼 수 있습니다.

- 통계 정보 변경으로 인한 재컴파일을 피하려면(예를 들어, 통계 정보 변경으로 인한 계획이 최적이지 아닌 경우), **KEEPFIXED PLAN** 쿼리 힌트를 지정합니다. 실제로 이 옵션을 사용하면, 재컴파일이 정확성-관련 원인(예를 들어, 테이블의 구조가 변경되어서 더 이상 해당 계획을 적용하지 않는 경우)인 경우에만 발생하고 통계 정보로 인해서는 발생하지 않도록 합니다. 문장에서 참조 중인 테이블의 구조가 변경되거나, 혹은 테이블이 **sp_recompile** 저장 프로시저로 표시되어서 재컴파일이 발생하는 경우가 그 예입니다.
- 테이블 혹은 인덱싱된 뷰에 정의된 인덱스와 통계 정보에 대해 “통계 자동 업데이트” 옵션을 해제하면 해당 개체의 통계 정보 변경으로 인한 재컴파일을 차단하게 됩니다. 주의할 것은, “통계 정보 자동화” 기능을 해제하는 것은 일반적으로 좋은 아이디어가 아닙니다. 이는 쿼리 최적화 프로그램이 해당 개체의 데이터 변경을 더 이상 반영하지 않게 되므로, 최적화 되지 못한 쿼리 계획을 만들게 됩니다. 이 방법은 다른 모든 대체 방법들을 철저히 검증한 후에 최후의 방안으로만 사용하기 바랍니다.
- 일괄 처리 내에서는 반드시 한정된 개체 이름(예, db.Table1)을 사용해야 개체 이름 해석 시의 모호함을 제거하고 재컴파일을 피할 수 있습니다.
- 지연된 컴파일로 인한 재컴파일을 피하기 위해서는, DML 과 DDL 을 교차하지 않습니다. 또한 IF 문장과 같은 조건절에서 DDL 을 구성하지 않습니다.
- 데이터베이스 엔진 튜닝 관리자(DTA)를 수행해서 쿼리 실행 시간 및 컴파일 시간 향상을 위해 어떠한 인덱스 변경이 필요한지 알 수 있습니다.
- 저장 프로시저가 **WITH RECOMPILE** 옵션 혹은 **RECOMPILE** 쿼리 힌트를 사용해서 만들어졌는지 점검합니다. 저장 프로시저가 **WITH RECOMPILE** 옵션으로 만들어진 경우 SQL Server 2005에서는 저장 프로시저 내의 특정 구문만 재컴파일하는 문장 단위 **RECOMPILE** 힌트의 이득을 취할 수 있습니다. 이는 실행될 때마다 전체 프로시저가 매번 재컴파일 되는 것을 피하면서 개별 문장에 대한 컴파일을 허용합니다. **RECOMPILE** 힌트에 대한 자세한 정보는, SQL Server

⁴ 카디널리티(Cardinality)는 수학에서 집합의 원소의 개수를 의미합니다. 여기서는 테이블의 행 수를 의미합니다.

온라인 설명서를 참조하십시오.

비효율적인 쿼리 계획

쿼리에 대한 실행 계획을 산출할 때, 최적화 프로그램은 해당 쿼리에 대해 가장 빠른 응답 속도를 제공하는 계획을 선택하려고 합니다. 가장 빠른 응답 시간이라는 의미가 반드시 I/O 양을 최소화하거나, CPU 를 가장 적게 사용한다는 것은 아닙니다. 여러 가지 리소스에 대한 균형을 맞추는 것입니다.

특정 유형의 연산자는 다른 것보다 더 많은 CPU 를 사용합니다. 근본적으로 **해시(Hash)** 연산자와 **정렬(Sort)** 연산자는 입력 데이터 전체를 스캔합니다. 스캔 작업에 미리 읽기(read ahead)가 적용됩니다. 이는 해당 연산자에 의해서 페이지가 요청되기 전에 대부분의 페이지들이 이미 버퍼 캐시에 적재된 상태로 만들어주는 것입니다. 따라서, 물리적 I/O 에 대한 대기를 최소화하거나 제거할 수 있습니다. 이러한 유형의 연산자가 물리적 I/O 에 더 이상 제약을 받지 않으면 높은 CPU 소비를 보여주게 됩니다. 그에 반해서, 중첩 반복 조인(Nested Loop Join)은 많은 인덱스 검색 연산을 수행하며 해당 페이지들이 버퍼 캐시에 모두 존재하지 않는다면, 테이블의 서로 다른 부분을 돌아다니며 인덱스 검색을 하게 되므로 결국 I/O 집중한 연산이 됩니다.

최적화 프로그램의 가장 중요한 입력은 Showplan(**EstimateRows** 와 **EstimateExecutions** 속성)에서 볼 수 있는 각 연산자별 카디널리티 예측입니다. 이는 여러 가지 대체 쿼리 계획에 대한 비용을 평가하는데 사용됩니다. 정확한 카디널리티 예측이 없다면, 이는 최적화 프로그램에서 사용되는 가장 중요한 입력에 결함이 생기는 것이며, 비효율적인 계획이 최종 계획으로 선택되는 문제를 내포하는 것입니다.

다음 자료, [Statistics Used by the Query Optimizer in Microsoft SQL Server 2005](http://www.microsoft.com/technet/prodtechnol/sql/2005/qrystats.msp)

(<http://www.microsoft.com/technet/prodtechnol/sql/2005/qrystats.msp>)는 SQL Server 최적화 프로그램이 통계 정보를 사용하는 방법을 자세히 설명하는 훌륭한 백서입니다. 이 백서는 최적화 프로그램이 통계 정보를 사용하는 방법, 최신 통계 정보로 유지하기 위한 최적화 사례, 그리고 정확한 카디널리티 예측을 방해함으로써 비효율적인 쿼리 계획의 원인이 되는 몇 가지 공통적인 쿼리 설계 문제를 논의합니다.

발견

보통 비효율적인 쿼리 계획들은 꽤 많이 발견됩니다. 비효율적인 쿼리 계획은 CPU 소비를 증가시킵니다.

sys.dm_exec_query_stats 를 쿼리하면 CPU 를 소비하는 대상 쿼리를 효율적으로 찾을 수 있습니다.

```

select
    highest_cpu_queries.plan_handle,
    highest_cpu_queries.total_worker_time,
    q.dbid,
    q.objectid,
    q.number,
    q.encrypted,
    q.[text]
from
    (select top 50
        qs.plan_handle,
        qs.total_worker_time
    from
        sys.dm_exec_query_stats qs
    order by qs.total_worker_time desc) as highest_cpu_queries
    cross apply sys.dm_exec_sql_text(plan_handle) as q
order by highest_cpu_queries.total_worker_time desc

```

또는, **sys.dm_exec_cached_plans** 을 쿼리해서 CPU 집약적인 여러 가지 연산자들 예를 들어, '%Hash Match%', '%Sort%' 같은 조건으로 필터를 사용함으로써 해당 주범을 찾을 수 있습니다.

해결 방안

비효율적인 쿼리 계획을 발견했다면 다음 옵션들을 고려합니다.

- 데이터베이스 엔진 튜닝 관리자으로 인덱스 권장 구성을 산출합니다.
- 카디널리티 예측 오류가 있는지 검사합니다.

결과 집합의 크기를 최대한 제한하도록 **WHERE** 절을 구성한 쿼리입니까? 그렇지 않은 쿼리는 근본적으로 리소스를 많이 소비합니다.

쿼리에 관련된 테이블에 **UPDATE STATISTICS** 를 실행하고 같은 문제가 계속되는지 확인합니다.

해당 쿼리가 최적화 프로그램으로 하여금 정확한 카디널리티 예측을 할 수 없도록 작성되어 있습니까? 구문 상의 문제가 있다면 쿼리 재작성을 고려하십시오.

- 해당 스키마나 쿼리를 변경할 수 없다면, **SQL Server 2005** 의 새로운 기능인 계획 지침(**Plan Guide**)을 사용해서 특정 텍스트와 일치하는 쿼리에 쿼리 힌트를 지정할 수 있습니다. 이 방법은 저장 프로시저 내부를 비롯한 임시(ad hoc) 쿼리 등에 적용할 수 있습니다. **OPTION (OPTIMIZE FOR)**와 같은 힌트는 최적화 프로그램이 특정 계획을 선택하는데 있어서 카디널리티 예측에 영향을 줍니다. **OPTION (FORCE ORDER)** 나 **OPTION (USE PLAN)**과 같은 힌트는 전반적인 쿼리 계획 조정에 범위를 변경할 수 있습니다.

쿼리 병렬 처리

쿼리 실행 계획 생성 시, **SQL Server** 쿼리 최적화 프로그램은 가장 빠른 응답 시간을 제공하는 계획을 선택하려고 합니다. 그런데 만일 쿼리

비용이 병렬 처리에 대한 비용 임계값(**cost threshold for parallelism**) 옵션에 설정된 값을 초과하게 되면, 최적화 프로그램은 병렬로 실행되는 계획을 생성합니다. 병렬 쿼리 계획은 다중 스레드로 쿼리를 처리합니다, 각 스레드는 가용한 CPU에 분산되고 각 프로세서로부터 CPU 시간을 사용합니다. 병렬 처리용 프로세서의 최대 개수는 최대 병렬 처리 수준(**max degree of parallelism**) 서버 옵션을 사용한 서버 범위에서 혹은 **OPTION(MAXDOP)** 힌트를 사용한 개별 쿼리 수준에서 제한할 수가 있습니다.

주어진 연산을 병렬로 수행할 스레드 수를 평가하는 실제 병렬 처리 수(**degree of parallelism, DOP**) 결정은 실행 시점까지 연기됩니다. **SQL Server 2005**는 쿼리를 실행하기 전에, 몇 개의 스케줄러가 사용 중인지를 판단하고 나머지 스케줄러를 충분히 사용하도록 쿼리의 **DOP**을 선택합니다. **DOP**이 선택되면, 해당 병렬 처리 수준으로 완료 때까지 실행됩니다. 병렬 쿼리는 보통 직렬 실행 계획과 비교해서 약간 더 높은⁵ CPU 시간을 사용하지만, 경과 시간은 더 단축됩니다. 다른 병목 예를 들어, 물리적 I/O를 위한 대기 등이 없다고 가정한다면 일반적으로 병렬 계획은 전체 프로세서에 걸쳐서 100%의 CPU를 사용할 것입니다.

병렬 계획 실행을 이끌어내는 핵심 요소(시스템 유휴 상태)는 쿼리가 실행된 후에 변경될 수가 있습니다. 예를 들어, 유휴 상태에서 쿼리가 요청된다면 서버는 병렬 계획을 선택하고 네 개의 **DOP**을 사용해서 4개의 서로 다른 프로세서에 스레드를 만들 수 있습니다. 해당 스레드가 실행될 때 기존 연결에서 많은 CPU를 요구하는 또 다른 쿼리가 전송될 수 있습니다. 그 시점에서는, 그 모든 스레드들이 현재 가용 CPU 시간을 나누어 사용해야 하므로 아주 짧은 시간 동안만 CPU를 사용할 수 있게 되고, 이것이 결국은 높은 쿼리 실행 시간을 유발하게 됩니다.

병렬 쿼리가 본질적으로 나쁜 것은 아니며 쿼리에 대한 빠른 응답 시간을 제공합니다. 그러나 쿼리에 대한 해당 응답 시간을 위해서 시스템의 전체 작업량과 다른 쿼리의 응답에 부담을 줍니다. 병렬 쿼리는 일반적으로 일괄 처리 작업과 의사 결정 지원용 작업에 적합하며 트랜잭션 처리 환경에서는 바람직하지 않습니다.

발견

쿼리 병렬 처리 문제를 찾아내기 위해서 다음 방법들을 사용합니다.

시스템 모니터 (Perfmon)

SQL Server: SQL Statistics – Batch Requests/sec 카운터를 조사합니다. **SQL Server** 온라인 설명서에서 “**SQL Statistics Object**”에 대한 보다 자세한 정보를 참조하십시오.

병렬 계획으로 결정되기 위해서는 쿼리 예측 비용이 “병렬 처리에 대한 비용 임계값” 구성 설정(기본값 5)을 초과할 정도로 커야하므로, 서버가 초당 처리하는 일괄 처리 수는 병렬 계획으로 실행될 때 훨씬 더 작아집니다. 병렬 쿼리가 많이 실행되는 서버는 일반적으로 작은 수의 초당 일괄 처리 요청 수를 가집니다 (예를 들어, 100 이하의 값).

DMV

다음 쿼리를 사용하면 특정 세션에서 병렬로 실행 중인 활성 요청을 알 수 있습니다.

⁵ 역자가 실제로 튜닝 컨설팅을 하면서 경험한 바로는 **DOP**의 개수에 따라서 2배 이상의 높은 CPU가 사용되는 경우를 자주 경험했습니다.

```

select
    r.session_id,
    r.request_id,
    max(isnull(exec_context_id, 0)) as number_of_workers,
    r.sql_handle,
    r.statement_start_offset,
    r.statement_end_offset,
    r.plan_handle
from
    sys.dm_exec_requests r
    join sys.dm_os_tasks t on r.session_id = t.session_id
    join sys.dm_exec_sessions s on r.session_id = s.session_id
where
    s.is_user_process = 0x1
group by
    r.session_id, r.request_id,
    r.sql_handle, r.plan_handle,
    r.statement_start_offset, r.statement_end_offset
having max(isnull(exec_context_id, 0)) > 0

```

해당 쿼리는 **sys.dm_exec_sql_text** 를 사용해서 쉽게 구할 수 있으며, 쿼리 계획은 **sys.dm_exec_cached_plan** 을 사용해서 구할 수 있습니다.

병렬 실행에 적합한 계획을 찾을 수 있습니다. 이는 **Parallel** 속성 값이 0 이 아닌 관계 연산자를 가진 캐시된 계획을 찾는 것으로 수행 가능합니다. 이러한 계획이 병렬로 실행되지 않을 수도 있지만, 시스템에 여유가 있다면 병렬 처리가 적합함을 의미합니다.

```

--
-- Find query plans that may run in parallel
--
select
    p.*,
    q.*,
    cp.plan_handle
from
    sys.dm_exec_cached_plans cp
    cross apply sys.dm_exec_query_plan(cp.plan_handle) p
    cross apply sys.dm_exec_sql_text(cp.plan_handle) as q
where
    cp.cacheobjtype = 'Compiled Plan' and
    p.query_plan.value('declare namespace
p="http://schemas.microsoft.com/sqlserver/2004/07/showplan";
    max(//p:RelOp/@Parallel)', 'float') > 0

```

일반적으로, 쿼리 실행 시간은 CPU 시간보다 깁니다. 이는 쿼리에서 잠금이나 물리적 I/O 같은 리소스 대기에 추가 시간을 소비하기 때문입니다. CPU 시간이 쿼리 경과 시간보다 더 많이 소비하는 경우는 CPU 를 동시에 사용하는 다중 쓰레드를 가진 병렬 계획으로 쿼리가 실행되는 경우입니다. 그러나 모든 병렬 쿼리가 이에 해당되지는 않습니다.

Note⁶: 다음 표에 기술한 코드 중 일부는 가독성을 위해서 여러 줄에 표시되었습니다. 실제로는 한 줄로 입력합니다.

⁶ 원문 코드의 주석 처리가 제대로 되어 있지 못해, 역자의 판단으로 주석 처리를 합니다.

```

select
    qs.sql_handle,
    qs.statement_start_offset,
    qs.statement_end_offset,
    q.dbid,
    q.objectid,
    q.number,
    q.encrypted,
    q.text
from
    sys.dm_exec_query_stats qs
    cross apply sys.dm_exec_sql_text(qs.plan_handle) as q
where
    qs.total_worker_time > qs.total_elapsed_time
/*
SQL Trace
CPU 시간이 Duration 보다 긴 문장이나 일괄 처리를 병렬 쿼리의 증상으로
보고 검색을 수행
*/
select
    EventClass,
    TextData
from
    ::fn_trace_gettable('c:\temp\high_cpu_trace.trc', default)
where
    EventClass in (10, 12)
        --RPC:Completed,SQL:BatchCompleted
    and CPU > Duration/1000
        --CPU 는 밀리초,Duration 은 밀리초나 마이크로초
/*
병렬 연산자를 가진 showplan (인코딩되지 않은)
*/
select
    EventClass,
    TextData
from
    ::fn_trace_gettable('c:\temp\high_cpu_trace.trc', default)
where
    TextData LIKE '%Parallelism%'

```

해결 방안

병렬 계획으로 실행되는 쿼리는 최적화 프로그램이 병렬 처리에 대한 비용 임계값(대략 5 초의 실행 시간)을 초과할 정도로 높은 비용임을 의미합니다. 위 코드를 통해서 식별된 쿼리들이 추가 튜닝 후보들입니다.

- 데이터베이스 엔진 튜닝 관리자를 사용해서 인덱스 변경, 인덱싱된 뷰 변경, 혹은 테이블 분할 변경 등이 쿼리 비용을 줄일 수 있는지 알아봅니다.
- 카디널리티 예측이 쿼리 비용 예측의 주요 요소이므로 예측과 실제 카디널리티 사이에 큰 차이가 있는지 확인합니다. 그러한 차이가 발견된다면:

“통계 자동 생성” 데이터베이스 옵션이 해제된 경우, Showplan(혹은 그래픽 실행 계획)의 Warnings 칼럼에 NO STATS⁷(열에 통계가 없음)로 표기된 항목이 없는지 확인합니다.

카디널리티 예측이 해제(OFF)되어 있는 테이블에 UPDATE STATISTICS 를 실행해 봅니다.

쿼리의 구조가 최적화 프로그램으로 하여금 정확한 예측을 어렵게 만들지 않는지 확인합니다. 다중 문 테이블-값 함수 혹은 CLR 함수, 테이블 변수, Transact-SQL 변수를 가진 비교(파라미터의 경우에 무관) 연산자의 사용 등이 이에 해당합니다.

- 쿼리를 다른 Transact-SQL 문장 혹은 표현을 사용해서 보다 효율적인 형태로 작성될 수 있는지 검토합니다.

서투른 커서 사용

SQL Server 2005 이전 버전에서는 서버 연결 당 하나의 활성 연결만을 지원했습니다. 활성 연결은 쿼리가 실행 중이거나 클라이언트로 결과 집합 전송이 진행 중인 상태를 말합니다. 때로는, 클라이언트 응용 프로그램이 결과 집합 전체를 읽으면서 각 행 별로 다른 쿼리를 실행하기 원할 때가 있습니다. 이러한 동작은 기본 결과 집합⁸상태에서는 수행할 수가 없습니다. 이 경우의 일반적인 해결 방법은 서버측 커서를 사용하도록 연결 속성을 변경하는 것입니다.

서버측 커서 사용 시, 데이터베이스 클라이언트 소프트웨어(OLE DB 공급자 혹은 ODBC 드라이버)는 클라이언트 요청을 **sp_cursoropen**, **sp_cursorfetch** 등과 같은 내부의 특별한 확장 저장 프로시저로 투명하게 캡슐화되어서 호출됩니다. 이러한 커서를 *API 커서*라고 합니다 (TSQL 커서와 상반되는). 사용자가 쿼리를 실행하면, 쿼리 텍스트는 **sp_cursoropen** 을 통해서 서버에 전송되며, 결과 집합으로부터 읽기 요청은 **sp_cursorfetch** 를 통해서 지정된 행 수 단위로 반환됩니다. 반입(Fetch)될 행 수를 조정함으로써 ODBC 드라이버 혹은 OLE DB 공급자가 하나 이상의 행을 캐시할 수 있습니다. 이는 서버 입장에서 클라이언트가 전체 행을 모두 읽을 때까지 대기하는 상황을 막아줍니다. 즉, 서버는 해당 연결에서 새로운 요청을 받을 수 있는 준비가 되는 것입니다.

커서를 열고 한 번에 한 행씩(혹은 작은 단위 행 수) 반입하는 응용 프로그램은 특히 광대역 네트워크(WAN)와 같은 환경에서 네트워크 지연으로 인한 병목을 아주 쉽게 유발합니다. 대량의 동시 사용자 연결을 가진 상황에서 많은 커서 요청을 처리하는 경우에 그 오버헤드는 더욱 분명해집니다. 결과 집합 내에서 커서를 특정 위치로 이동하는 작업과 관련된 오버헤드, 개별 요청을 처리하는 오버헤드, 그리고 유사한 처리 작업들로 인한 오버헤드 때문에, 한 번에 한 행씩 처리하면서 100 번의 개별 요청을 처리하는 것보다, 단일 요청으로 100 개의 행을 반환하는 것이 보다 효율적입니다.

발견

커서 사용 문제를 해결하기 위해서 다음 방법들을 사용할 수 있습니다.

⁷ 한글 버전을 기준으로 언급합니다.

⁸ SQL Server 가 클라이언트로 결과 집합을 전송하는 기본 방식으로, 이전의 Firehose 에 해당합니다. 전진-전용, 읽기-전용 방식입니다.

시스템 모니터 (Perfmon)

성능 카운터 **SQL Server: Cursor Manager By Type – Cursor Request/Sec**를 조사하면 시스템상의 대략적인 커서 사용량을 알 수 있습니다. 작은 행 단위 반입으로 인해 높은 CPU 사용량을 보이는 시스템은 보통 초당 수백의 커서 요청을 가집니다. 반입되는 버퍼 크기에 관해 알려주는 정확한 카운터는 없습니다.

DMV

다음 예제는 fetch buffer 크기가 한 행인 API 커서를 사용하는 연결을 알 수 있습니다. 한 행 이상 보다 큰 fetch buffer를 사용하는 것이 훨씬 더 효율적입니다.

```
select
    cur.*
from
    sys.dm_exec_connections con
    cross apply sys.dm_exec_cursors(con.session_id) as cur
where
    cur.fetch_buffer_size = 1
    and cur.properties LIKE 'API%'
    -- API cursor (TSQL cursors always have fetch buffer of 1)
```

SQL Trace

SQL Trace에서 **sp_cursorfetch** 문장을 검색하기 위해 **RPC: Completed** 이벤트 클래스를 포함시켜서 사용합니다. 4 번째 매개 변수 값이 반환되는 행 수입니다. 반환될 최대 행 수는 해당 **RPC: Starting** 이벤트 클래스에 입력 파라미터로 지정됩니다.

해결 방안

- 커서가 가장 적합한 방법인지에 대해서 재고해야 합니다. 일반적으로 집합 기반 처리가 보다 효율적입니다.
- SQL Server 2005에 연결한다면 다중 활성 결과 집합(MARS) 사용을 고려합니다.
- 사용하는 API 관련 문서를 참조해서 어느 정도 fetch buffer 크기가 커서에 적합한지 확인합니다.

ODBC - **SQL_ATTR_ROW_ARRAY_SIZE**

OLE DB – **IRowset::GetNextRows** or **IRowsetLocate::GetRowsAt**

메모리 병목

이번 절은 특히 적은 메모리 구성에서 메모리에 대한 진단 분석, 메모리 관련 오류, 해당 원인 및 해결 방안 등을 다룹니다.

배경

서로 다른 메모리 리소스이지만 그냥 메모리라는 용어를 사용하는 것이 일반적입니다. 메모리 리소스의 유형이 여러가지이므로, 실제로 어떤 메모리 리소스를 말하는지 이해하고 구분하는 것이 중요합니다.

가상 주소 공간과 실제 메모리

마이크로소프트 Windows®는, 각 프로세스 단위로 가상 주소 공간(VAS)을 가집니다. 프로세스에서 가용한 모든 가상 주소 집합으로 VAS 크기를 구성합니다. VAS의 크기는 아키텍처(32 혹은 64 비트)와 운영 시스템에 따라 결정됩니다. 메모리 문제 해결 측면에서 가상 메모리는 고갈될 수 있는 메모리 리소스이며 64 비트 플랫폼에서 실제 메모리가 가용한 상황에서도 응용 프로그램이 메모리 부족을 경험할 수 있다는 점을 이해하는 것이 중요합니다.

가상 주소 공간에 대한 추가 정보는 SQL Server 온라인 설명서의 “프로세스 주소 공간”에 대한 항목과 MSDN 기사 [Virtual Address Space](http://msdn2.microsoft.com/en-us/library/aa366912.aspx) (<http://msdn2.microsoft.com/en-us/library/aa366912.aspx>⁹)을 참조하십시오.

Address Windowing Extensions (AWE)과 SQL Server

Address Windowing Extensions (AWE)는 32 비트 응용 프로그램이 32 비트 주소 한계를 초과하는 실제 메모리를 사용하도록 지원하는 API입니다. AWE 메커니즘은 기술적으로 64 비트 플랫폼에서 필요치 않지만 계속 지원하고 있습니다. 64 비트 플랫폼에서 AWE 메커니즘을 통해서 할당된 메모리 페이지를 *잠긴 페이지(locke page)*라고 합니다.

32 비트와 64 비트 양쪽 플랫폼에서 AWE 메커니즘을 통해 할당된 메모리는 페이징되지 않습니다. 이것이 응용 프로그램에 이득이 될 수 있습니다. (이 점이 64 비트 플랫폼에서 AWE 메커니즘을 사용하는 이유 중의 하나입니다.) 그러나 이러한 동작이 시스템과 다른 응용 프로그램이 사용할 RAM 용량에 부정적인 영향을 줍니다. 그런 이유로, AWE를 사용하기 위해서는 SQL Server를 실행하는 Windows 계정에 대해 **메모리 페이지 잠그기(Lock Pages in Memeory)** 사용 권한이 부여되어야 합니다.

문제 해결 측면에서 본다면 SQL Server 버퍼 풀이 AWE 매핑 메모리를 사용한다는 점입니다; 그러나, 데이터베이스(해시) 페이지만이 AWE를 통해 할당된 메모리의 모든 이득을 얻을 수 있습니다. AWE 메커니즘을 통해 할당된 메모리 정보는 작업 관리자 혹은 **Process: Private Bytes** 성능 카운터를 통해서만 제공되지 않습니다. 이러한 정보를 얻기 위해서는 SQL Server 특정 카운터나 동적 관리 뷰(DMV)를 사용해야 합니다.

AWE 매핑 메모리에 대한 자세한 정보는 SQL Server 온라인 설명서의 목차에서 “큰 데이터베이스의 메모리 관리”, “메모리 아키텍처” 항목과 MSDN 기사 [Large Memory Support](http://msdn2.microsoft.com/en-us/library/aa366718.aspx) (<http://msdn2.microsoft.com/en-us/library/aa366718.aspx>¹⁰)를 참조하십시오.

다음 표는 SQL Server 2005의 각 구성별 지원 가능한 최대 메모리 옵션을 요약한 것입니다. (SQL Server 혹은 Windows 각 에디션별로 지원 메모리 크기에 대한 제한이 있음을 참고하십시오.)

표 1

구성	VAS	최대 실제 메모리	AWE/잠긴 메모리 지원
Native 32-bit on 32-bit OS	2 GB	64 GB	Yes

⁹ 원본의 주소가 달라 변경합니다.

¹⁰ 원본의 주소가 달라 변경합니다.

구성	VAS	최대 실제 메모리	AWE/잠긴 메모리 지원
with /3GB boot parameter ¹	3 GB	16 GB	Yes
32-bit on x64 OS (WOW)	4 GB	64 GB	Yes
32-bit on IA64 OS (WOW)	2 GB	2 GB	No
Native 64-bit on x64 OS	8 terabyte	1 terabyte	Yes
Native 64-bit on IA64 OS	7 terabyte	1 terabyte	Yes

¹ boot 매개변수에 대한 상세 정보는 SQL Server 온라인 설명서 목차의 "AWE 사용"을 참조하십시오.

메모리 부족

메모리 부족이란 가용 메모리가 제한된 상태를 말합니다. SQL Server 가 메모리 부족 상태에서 실행되고 있음을 확인하는 것은 메모리 관련 문제를 해결하는데 도움을 줍니다. SQL Server 는 메모리 부족 유형에 따라 다른 응답을 합니다. 다음 표는 일반적인 메모리 부족 유형과 근본 원인을 요약합니다.

표 2

부족	외부	내부
실제	<p>실제 메모리(RAM)가 많지 않음. 이로 인해 현재 실행 중인 프로세스의 작업 메모리가 감소하게 되고 결국 성능 저하를 유발합니다.</p> <p>SQL Server 는 이러한 상태를 감지하고 구성 옵션 설정 상태에 따라 버퍼 풀에 대상 메모리 크기를 줄이고 내부 캐시를 해제할 수 있습니다.</p>	<p>SQL Server 는 내부 구성 요소간의 메모리 재분배를 유발하는 내부의 높은 메모리 소비를 감지합니다.</p> <p>내부 메모리 부족 상태의 원인은:</p> <ul style="list-style-type: none"> 외부 메모리 부족 상태를 반영. 메모리 설정 변경(예, "최대 서버 메모리"). 내부 구성 요소간의 메모리 분배 변경(버퍼 풀에서 높은 비율의 예약 공간과 stolen page 로 인해).
가상	<p>시스템 페이지 파일 공간이 부족한 상태로 실행. 이는 현재 할당된 메모리가 페이징될 수 없도록 만들고, 시스템에서 메모리 할당이 실패하는 원인이 됩니다. 이러한 상태가 결국 시스템 전체 응답을 매우 느려지게 만들거나 정지</p>	<p>조각(많은 VAS 가 가용하지만 작은 블록 단위), 혹은 메모리 소비(직접 할당, SQL Server VAS 로 DLL 로드, 대량의 쓰레드)로 인한 낮은 VAS 상태로 실행.</p> <p>SQL Server 가 이러한 상태를</p>

부족	외부	내부
	상태로 만들 수 있습니다.	감지하고 VAS 에 예약된 영역 감소, 버퍼 풀의 대상 메모리 감소, 그리고 캐시를 축소시킬 수 있습니다.

Windows 는 실제 메모리가 높은 혹은 낮은 상태로 실행하면 알람을 주는 메커니즘을 가지고 있습니다. SQL Server 는 이러한 메커니즘을 사용해서 메모리를 관리합니다.

표 3 은 경우에 따른 일반적인 메모리 문제 해결 단계를 설명합니다.

표 3

부족	외부	내부
실제	<ul style="list-style-type: none"> 시스템 메모리를 소비하는 주범을 찾습니다. 	<ul style="list-style-type: none"> SQL Server 내부의 메모리 소비 주범을 찾습니다.
	<ul style="list-style-type: none"> (가능하다면) 제거합니다. 	<ul style="list-style-type: none"> 서버 구성 옵션 설정을 확인합니다.
	<ul style="list-style-type: none"> 시스템 RAM 이 충분한지 검사하고 RAM 추가를 고려합니다. 	<ul style="list-style-type: none"> 조사 결과에 따른 추가 조치: 작업 부하; 설계 문제; 다른 리소스 병목 등을 검사합니다.
가상	<ul style="list-style-type: none"> 스와핑 파일 크기를 증가시킵니다. 	<ul style="list-style-type: none"> 위에서 소개한 내부 실제 메모리 부족 단계를 따릅니다.
	<ul style="list-style-type: none"> 실제 메모리 소비 주범을 검사하고 위에서 소개한 외부 실제 메모리 부족에 대한 수행 단계를 따릅니다. 	

도구

다음 도구와 정보들을 이용해서 문제를 해결할 수 있습니다.

- 메모리 관련 DMV
- DBCC MEMORYSTATUS 명령
- 성능 카운터: SQL Server 특정 개체에 대한 성능 모니터 혹은 DMV
- Task Manager 작업 관리자
- 이벤트 뷰어: 응용 프로그램 로그, 시스템 로그

메모리 부족 발견

메모리 부족 그 자체가 문제를 나타내지는 않습니다. 메모리 부족이란 서버의 메모리가 필요하지만 충분하지는 않은 상태로 나중에 메모리

오류가 발생할 수 있음을 말합니다. 메모리가 부족한 상태에서 작업하는 것이 서버에 정상적인 운영 상태일 수 있습니다. 그러나, 메모리 부족으로 인한 증상은 서버가 자신의 용량에 근접한 상태로 실행 중이며 잠재적인 메모리 부족 오류가 있음을 나타내는 것이기도 합니다. 정상적으로 운영 중인 서버의 경우엔 이러한 정보들이 차후 메모리 부족에 대한 판단 근거의 기준선으로 제공될 수 있습니다.

외부 실제 메모리 부족

작업 관리자를 열고 성능 탭에서 사용 가능 **실제 메모리** 항목의 **사용 가능** 값을 검사합니다. 사용 가능 메모리 양이 작다면, 외부 메모리가 부족한 것입니다. 정확한 값은 여러가지 요소에 따라 달라지지만 그 값이 50-100MB 로 떨어진 경우에 조사를 시작합니다. 이 값이 10MB 이하로 떨어지면 분명히 외부 메모리가 부족한 상태입니다.

동일한 정보를 시스템 모니터의 **Memory: Available Bytes** 카운터를 통해서 얻을 수 있습니다.

외부 메모리 부족이 존재하고 메모리 관련 오류를 만난다면 시스템의 실제 메모리를 소비하는 주범이 누구인지 찾아내야 합니다. **Process: Working Set** 성능 카운터나 작업 관리자의 **프로세스** 탭에서 **메모리 사용** 열을 살펴보고 그 주범을 찾아냅니다.

다음 카운터들을 요약함으로써 시스템의 실제 메모리 전체 사용량을 대략적으로 구할 수 있습니다.

- 각 프로세스별 **Process** 개체, **Working Set** 카운터
- **Memory** 개체
 - 시스템 작업 집합(working set)에 대한 **Cache Bytes** 카운터
 - 비페이지 풀 크기에 대한 **Pool Nonpaged Bytes** 카운터
 - **Available Bytes** (작업 관리자의 **사용 가능** 값과 동일)

외부 메모리 부족이 없다면, **Process: Private Bytes** 카운터나 작업 관리자의 가상 메모리 크기가 작업 집합(**Process: Working Set** 혹은 작업 관리자 **사용 가능**) 크기에 가깝습니다. 이는 메모리가 페이징되지 않음을 의미합니다.

작업 관리자의 메모리 사용 열 그리고 해당하는 성능 카운터는 AWE 를 통해서 할당된 메모리는 계산하지 않음을 참고하십시오.

다음처럼 sys.dm_os_memory_clerks DMV 를 사용해서 AWE 메커니즘을 통해서 할당된 SQL Server 메모리량을 알아낼 수 있습니다.

```
select
    sum(awe_allocated_kb) / 1024 as [AWE allocated, Mb]
from
    sys.dm_os_memory_clerks
```

SQL Server 는 현재 AWE 옵션이 활성화된 경우에 버퍼 풀 클럭(type = 'MEMORYCLERK_SQLBUFFERPOOL')에 대해서만 이 메커니즘을 사용한다는 점을 참고하십시오.

가능한 실제 메모리를 소비하는 주범을 식별하고 제거함으로써 외부 메모리 부족을 제거합니다, 혹은 더 많은 메모리를 추가함으로써 메모리 관련 문제를 해결할 수 있습니다.

외부 가상 메모리 부족

페이지 파일이 현재 메모리 할당을 수용할만한 충분한 공간이 있는지를 판단해야 합니다. 이를 위해, 작업 관리자를 열고 성능 탭에서 **할당된 메모리** 항목을 검사합니다. 전체가 한도에 가깝다면, 페이지 파일 공간이 부족할 가능성이 있습니다. 한도는 페이지 파일 공간을 확장하지 않고 할당될 수 있는 최대 메모리 크기를 나타냅니다. 작업 관리자의 **할당된 메모리 전체**는 페이지 파일 사용에 대한 잠재성이며, 실제 사용을 나타내지 않습니다. 실제 사용은 실제 메모리 부족 상태에서 증가합니다.

동일한 정보를 **Memory: Commit Limit, Paging File: %Usage, Paging File: %Usage Peak** 카운터에서 얻을 수 있습니다.

Process: Private Bytes 카운터에서 **Process: Working Set** 값을 빼면 프로세스당 페이지징되는 메모리 크기를 대략적으로 예측할 수 있습니다.

Paging File: %Usage Peak (혹은 할당된 메모리 최고)가 높으면, 페이지 파일 증가 혹은 “running low on virtual memory” 알림을 나타내는 이벤트가 있는지 시스템 이벤트 로그를 확인합니다. 페이지 파일 크기를 증가시킵니다. 높은 **Paging File: %Usage** 는 실제 메모리가 과도하게 소비되었으며 외부 실제 메모리의 부족 상태를 함께 고려해야 함을 나타냅니다.

내부 실제 메모리 부족

내부 메모리 부족은 SQL Server 자체 설정에서 비롯되므로, 그 논리적인 단계는 SQL Server 내부의 메모리 분배를 살펴보고 버퍼 분배에 어떤 이상이 없는지를 검사하는 것입니다. 정상적으로는 버퍼 풀이 SQL Server 에 의해서 할당된 메모리의 대부분을 차지합니다. 버퍼 풀에 속한 메모리 크기를 판단하기 위해 DBCC MEMORYSTATUS 출력을 살펴봅니다. Buffer Counts 부분에서 **Target** 값을 찾습니다. 다음은 서버가 정상적인 부하를 가진 상태에서 DBCC MEMORYSTATUS 출력의 일부를 보여줍니다.

Buffer Counts	Buffers
Committed	201120
Target	201120
Hashed	166517
Reserved Potential	143388
Stolen Potential	173556
External Reservation	0
Min Free	256
Visible	201120
Available Paging File	460640

Target 은 페이지징 없이 할당될 수 있는 8KB 페이지 수로 SQL Server 에 의해서 계산됩니다. **Target** 은 Windows 로부터 낮은/높은 메모리 알림에 응답하며 정기적으로 다시 계산됩니다. 정상적으로 부하를 가진 서버에서 target 페이지의 수가 감소되는 것은 외부 실제 메모리 부족을 나타낼 수 있습니다.

SQL Server 가 많은 메모리를 소비한다면(**Process: Private Bytes** 혹은 작업 관리자의 **메모리 사용** 열에 의해서 판단할 수 있는), **Target** 카운트가 메모리의 상당 부분을 차지하는지 확인합니다. AWE 옵션이 활성화 상태라면 AWE 로 할당된 메모리는 **sys.dm_os_memory_clerks** 나 DBCC MEMORYSTATUS 출력에서 계산해야 합니다.

위 예제(AWE 는 비활성화)에서 $\text{Target} * 8\text{KB} = 1.53\text{GB}$ 라고 가정합니다, **Process: Private Bytes** 가 대략 1.62GB 라면 SQL Server 가 소비하는 메모리의 94%가 버퍼 풀 target 에 해당합니다. 서버가 정상적인 부하를 포함한 상태가 아니라면, 정상적인 상황일 때는 **Target** 이 **Process: Private Bytes** 성능 카운터 값을 초과할 수 있습니다.

Target 이 낮은 반면, 서버의 **Process: Private Bytes** 혹은 작업 관리자의 **메모리 사용** 열이 높다면, 이는 버퍼 풀 이외에 다른 메모리 구성 요소로부터 내부 메모리 부족을 나타낼 수 있습니다. SQL Server 프로세스 영역으로 로드되는 COM 개체, 연결된 서버, 확장 저장 프로시저, SQLCLR 및 기타 구성 요소들이 버퍼 풀 외부에서 메모리를 소비하는 메모리 구성 요소들입니다. 특히 이 구성 요소들이 SQL Server 메모리 인터페이스를 사용하지 않을 경우 소비되는 메모리를 추적하기가 어렵습니다.

SQL Server 메모리 관리 메커니즘을 사용하는 구성 요소들은 작은 메모리 할당을 위해 버퍼 풀을 사용합니다. 이 구성 요소들이 8KB 이상을 할당할 경우, 다중 페이지 할당자(multi-page allocator) 인터페이스를 통해서 버퍼 풀 외부의 메모리를 사용합니다.

다음은 다중 페이지 할당자를 통해서 소비된 메모리 양을 확인할 수 있는 빠른 방법입니다.

```
-- 다중 페이지 할당자 인터페이스를 통해서 할당된 메모리 크기
select sum(multi_pages_kb)
from sys.dm_os_memory_clerks
```

다음 코드로 다중 페이지 할당자를 통해서 할당된 메모리에 대한 분배 정보를 상세하게 알 수 있습니다:

```
select
    type, sum(multi_pages_kb)
from
    sys.dm_os_memory_clerks
where
    multi_pages_kb != 0
group by type
```

type	
-----	-----
MEMORYCLERK_SQLSTORENG	56
MEMORYCLERK_SQLOPTIMIZER	48
MEMORYCLERK_SQLGENERAL	2176
MEMORYCLERK_SQLBUFFERPOOL	536
MEMORYCLERK_SOSNODE	16288
CACHESTORE_STACKFRAMES	16
MEMORYCLERK_SQLSERVICEBROKER	192
MEMORYCLERK_SNI	32

다중 페이지 할당자를 통해 할당된 메모리 양이 크다면(100-200MB 이상), 추가 조사가 필요합니다.

다중 페이지 할당자를 통해서 할당된 대량 메모리가 보인다면, 서버 구성을 확인하고 이전 혹은 다음 쿼리를 사용해서 메모리 소비 주범이 어떤 구성 요소인지를 판단합니다.

Target 은 낮지만 백분율로는 SQL Server 메모리의 대부분을 차지한다면, 이전([외부 실제 메모리 부족](#))에 소개한 외부 메모리 부족에 대한 원인을 찾아보거나 서버 메모리 구성 값을 확인해 봅니다.

최대 서버 메모리나 최소 서버 메모리 옵션을 설정했다면 target 값과 비교해 봐야 합니다. 최대 서버 메모리는 버퍼 풀에 의해서 소비되는 최대 메모리를 제한합니다, 그러나 서버 전체적으로는 더 많은 메모리를 소비할 수 있습니다. 최소 서버 메모리는 버퍼 풀 메모리가 그 이하로는 내려가지 않도록 합니다. Target 이 최소 서버 메모리 설정보다 작고 서버가 정상적으로 부하를 가진 상황이라면 이는 서버가 외부 메모리 부족 상태이며 옵션에 지정된 메모리 양만큼 확보할 수 없음을 나타냅니다. 더불어 위에서 언급한 내부 구성 요소의 메모리 부족도 의미하는 것입니다. Target 카운트는 최대 서버 메모리 옵션 설정을 초과할 수 없습니다.

우선, DBCC MEMORYSTATUS 출력에서 stolen 페이지 카운트를 검사합니다.

Buffer Distribution	Buffers
-----	-----
Stolen	32871
Free	17845
Cached	1513
Database (clean)	148864
Database (dirty)	259
I/O	0
Latched	0

stolen 페이지가 target 에 비례해서 높은 백분율(75-80% 이상)을 가진다면 내부 메모리 부족을 나타냅니다.

서버 구성 요소별 메모리 할당에 대한 자세한 정보는 sys.dm_os_memory_clerks DMV 를 참조합니다.

```
-- 버퍼풀 이외의 구성 요소에 의해 소비된 메모리 계산
-- single_pages_kb 는 BPool 에 해당하므로 제외
-- BPool 은 다음 쿼리에서 계산
select
    sum(multi_pages_kb
        + virtual_memory_committed_kb
        + shared_memory_committed_kb) as
[Overall used w/o BPool, Kb]
from
    sys.dm_os_memory_clerks
where
    type <> 'MEMORYCLERK_SQLBUFFERPOOL'

-- BPool 에 의해 소비된 메모리 계산
-- 현재 AWE 는 BPool 에서만 사용
select
    sum(multi_pages_kb
        + virtual_memory_committed_kb
        + shared_memory_committed_kb
        + awe_allocated_kb) as [Used by BPool with AWE, Kb]
from
    sys.dm_os_memory_clerks
where
    type = 'MEMORYCLERK_SQLBUFFERPOOL'
```

각 구성 요소별 상세 정보는 다음 쿼리에서 구할 수 있습니다. (버퍼 풀과 그 이외에 구성 요소들 모두를 포함합니다.)

```

declare @total_alloc bigint
declare @tab table (
    type nvarchar(128) collate database_default
    ,allocated bigint
    ,virtual_res bigint
    ,virtual_com bigint
    ,awe bigint
    ,shared_res bigint
    ,shared_com bigint
    ,topFive nvarchar(128)
    ,grand_total bigint
);
-- 정상적인 상황에서 버퍼 풀의 committed memory 는
-- 대량 메모리 소비자를 나타내므로 계산에서 제외
select
    @total_alloc =
        sum(single_pages_kb
            + multi_pages_kb
            + (CASE WHEN type <> 'MEMORYCLERK_SQLBUFFERPOOL'
                THEN virtual_memory_committed_kb
                ELSE 0 END)
            + shared_memory_committed_kb)
from
    sys.dm_os_memory_clerks
print
    'Total allocated (including from Buffer Pool): '
    + CAST(@total_alloc as varchar(10)) + ' Kb'
insert into @tab
select
    type
    ,sum(single_pages_kb + multi_pages_kb) as allocated
    ,sum(virtual_memory_reserved_kb) as virtual_res
    ,sum(virtual_memory_committed_kb) as virtual_com
    ,sum(awe_allocated_kb) as awe
    ,sum(shared_memory_reserved_kb) as shared_res
    ,sum(shared_memory_committed_kb) as shared_com
    ,case when (
        (sum(single_pages_kb
            + multi_pages_kb
            + (CASE WHEN type <> 'MEMORYCLERK_SQLBUFFERPOOL'
                THEN virtual_memory_committed_kb
                ELSE 0 END)
            + shared_memory_committed_kb))/
        (@total_alloc + 0.0)) >= 0.05
        then type
        else 'Other'
    end as topFive
    ,(sum(single_pages_kb
        + multi_pages_kb
        + (CASE WHEN type <> 'MEMORYCLERK_SQLBUFFERPOOL'
            THEN virtual_memory_committed_kb
            ELSE 0 END)
        + shared_memory_committed_kb)) as grand_total
from
    sys.dm_os_memory_clerks
group by type

```



```
order by (sum(single_pages_kb + multi_pages_kb
+ (CASE WHEN type <>
'MEMORYCLERK_SQLBUFFERPOOL' THEN
virtual_memory_committed_kb ELSE 0 END) +
shared_memory_committed_kb)) desc
select * from @tab
```

버퍼 풀은 단일 페이지 할당자를 통해서 다른 구성 요소에 메모리를 제공하므로 이전 쿼리에서는 다르게 처리합니다. 다음 쿼리를 사용해서 버퍼 풀 페이지(단일 페이지 할당자를 통하는)의 상위 10 위 소비자를 알 수 있습니다.

```
-- Bpool 의 메모리 소비 상위 10 위
select
    top 10 type,
    sum(single_pages_kb) as [SPA Mem, Kb]
from
    sys.dm_os_memory_clerks
group by type
order by sum(single_pages_kb) desc
```

일반적으로는 내부 구성 요소가 소비하는 메모리에 대한 통제권이 없습니다. 그러나, 어떤 구성 요소가 메모리 소비 주범인지를 판단하는 것은 메모리 문제에 대한 조사 범위를 좁히는데 도움을 줄 것입니다.

시스템 모니터(Perfmon)

메모리가 부족한지 판단하기 위해서 다음 성능 카운터를 확인할 수 있습니다. (상세한 설명은 SQL Server 온라인 설명서를 참조하십시오.):

SQL Server: Buffer Manager 개체

- Buffer cache hit ratio 가 작은 경우
- Page life expectancy 가 작은 경우
- Checkpoint pages/sec 가 높은 경우
- Lazy writes/sec 가 높은 경우

메모리 부족과 I/O 오버헤드는 일반적으로 서로 관련된 병목입니다. 본 문서의 [I/O 병목](#)를 참조하십시오.

캐시와 메모리 부족

외부 및 내부 메모리 부족을 조사하는 또 다른 방법은 메모리 캐시 동작을 조사하는 것입니다.

SQL Server 2005 의 내부 구현에 있어서, SQL Server 2000 과 비교되는 차이점의 하나는 바로 캐싱(caching) 프레임워크의 단일화입니다. 프레임워크는 캐시에서 사용 빈도가 적은 항목들을 제거하기 위해 클럭 알고리즘을 구현합니다. 현재는 내부 클럭 포인터와 외부 클럭 포인터로 두 개의 클럭 포인터¹¹(clock hands)를 가집니다.

내부 클럭 포인터는 다른 캐시에 비례하여 캐시 크기를 조절합니다. 해당 캐시가 한계점에 다달았음을 프레임워크가 예상하게 되면 이동을 시작합니다.

¹¹ 온라인 설명서의 번역을 따릅니다.

외부 클럭 포인트는 SQL Server 메모리가 부족한 상태가 되면 이동을 시작합니다. 외부 클럭 포인트의 이동은 외부 및 내부 메모리의 부족이 그 원인이 될 수 있습니다. 내부 및 외부 클럭 포인트의 이동을 내부 및 외부 메모리 부족과 혼동하지 마십시오.

다음 코드와 같이 클럭 포인트 이동에 관한 정보는

sys.dm_os_memory_cache_clock_hands DMV 를 통해서 알 수 있습니다. 각 캐시 항목은 내부 및 외부 클럭 포인트별로 하나의 행을 가집니다. **rounds_count** 와 **removed_all_rounds_count** 값이 증가한다면 서버의 내부/외부 메모리가 부족한 상태입니다.

```
select *
from
    sys.dm_os_memory_cache_clock_hands
where
    rounds_count > 0
    and removed_all_rounds_count > 0
```

다음과 같이 **sys.dm_os_cache_counters** DMV 와 조인해서 캐시 크기와 같은 추가 정보들을 얻을 수 있습니다.

```
select
    distinct cc.cache_address,
    cc.name,
    cc.type,
    cc.single_pages_kb + cc.multi_pages_kb as total_kb,
    cc.single_pages_in_use_kb + cc.multi_pages_in_use_kb
    as total_in_use_kb,
    cc.entries_count,
    cc.entries_in_use_count,
    ch.removed_all_rounds_count,
    ch.removed_last_round_count
from
    sys.dm_os_memory_cache_counters cc
    join sys.dm_os_memory_cache_clock_hands ch on
        (cc.cache_address =ch.cache_address)
/*
-- 포인트가 이동된 캐시만을 볼 경우 이 블록 주석을 제거
where
    ch.rounds_count > 0
    and ch.removed_all_rounds_count > 0
*/
order by total_kb desc
```

USERSTORE 항목에 대해서는 사용 중인 페이지 정보가 출력되지 않으며 대신 NULL 이 됨을 참고하십시오.

링 버퍼

메모리 정보 진단을 위한 의미 있는 크기는 **sys.dm_os_ring_buffers** DMV 에서 얻을 수 있습니다. 각 링 버퍼는 특정 유형에 마지막 알림 정보를 가지는 하나의 record 를 유지합니다.

RING_BUFFER_RESOURCE_MONITOR

메모리 상태 변경을 식별하기 위해 리소스 모니터 알림 정보를 이용할 수 있습니다. 내부적으로 서로 다른 메모리 부족을 모니터링하는 프레임워크를 가지고 있습니다. 메모리 상태가 변경되면, 리소스 모니터가 알림을 생성합니다. 이 알림은 메모리 상태에 따른 메모리 사용 조절을 위해서

해당 구성 요소가 내부적으로 사용하며 다음 코드와 같이 **sys.dm_os_ring_buffers** DMV 를 통해서 사용자에게 보여줍니다.

```
select record
from sys.dm_os_ring_buffers
where ring_buffer_type = 'RING_BUFFER_RESOURCE_MONITOR'
record 는 다음과 같습니다:
```

```
<Record id="1701" type="RING_BUFFER_RESOURCE_MONITOR"
time="149740267">
  <ResourceMonitor>
    <Notification>RESOURCE_MEMPHYSICAL_LOW<
    /Notification>
    <Indicators>2</Indicators>
    <NodeId>0</NodeId>
  </ResourceMonitor>
  <MemoryNode id="0">
    <ReservedMemory>1646380</ReservedMemory>
    <CommittedMemory>432388</CommittedMemory>
    <SharedMemory>0</SharedMemory>
    <AWEMemory>0</AWEMemory>
    <SinglePagesMemory>26592</SinglePagesMemory>
    <MultiplePagesMemory>17128</MultiplePagesMemory>
    <CachedMemory>17624</CachedMemory>
  </MemoryNode>
  <MemoryRecord>
    <MemoryUtilization>50</MemoryUtilization>
    <TotalPhysicalMemory>3833132</TotalPhysicalMemory>
    <AvailablePhysicalMemory>3240228<
    /AvailablePhysicalMemory>
    <TotalPageFile>5732340</TotalPageFile>
    <AvailablePageFile>5057100</AvailablePageFile>
    <TotalVirtualAddressSpace>2097024<
    /TotalVirtualAddressSpace>
    <AvailableVirtualAddressSpace>336760
  </AvailableVirtualAddressSpace>
    <AvailableExtendedVirtualAddressSpace>0
    </AvailableExtendedVirtualAddressSpace>
  </MemoryRecord>
</Record>
```

이 레코드에서 서버가 실제 메모리 부족 알림을 받았음을 알 수 있습니다. 더불어 킬로바이트단위의 메모리 크기도 알 수 있습니다. SQL Server 의 XML 기능을 활용해서 이러한 정보들을 쿼리할 수 있습니다.

```

select
    x.value('(/Notification)[1]', 'varchar(max)') as [Type],
    x.value('(/Record/@time)[1]', 'bigint') as [Time Stamp],
    x.value('(/AvailablePhysicalMemory)[1]', 'int')
    as [Avail Phys Mem, Kb],
    x.value('(/AvailableVirtualAddressSpace)[1]', 'int')
    as [Avail VAS, Kb]
from
    (select cast(record as xml)
     from sys.dm_os_ring_buffers
     where ring_buffer_type = 'RING_BUFFER_RESOURCE_MONITOR')
    as R(x)
order by [Time Stamp] desc

```

메모리 부족 알림을 받았으므로, 버퍼 풀 target 크기를 다시 계산합니다. target 카운트는 **최소 서버 메모리**와 **최대 서버 메모리** 옵션에 의해서 지정된 범위 내에 있음을 참고하십시오. 버퍼 풀을 위해 새로 할당된 target 크기가 현재 크기보다 작은 경우, 버퍼 풀은 외부 실제 메모리 부족이 없어질 때까지 축소합니다. 참고로 SQL Server 2000에서는 AWE가 활성화된 경우 실제 메모리 부족에 반응하지 못했습니다.

RING_BUFFER_OOM

이 링 버퍼는 다음 예제 코드처럼 서버 메모리 부족(out-of-memory)를 나타내는 레코드를 포함합니다.

```

select record
from sys.dm_os_ring_buffers
where ring_buffer_type = 'RING_BUFFER_OOM'

```

record 는 다음과 같습니다:

```

<Record id="7301" type="RING_BUFFER_OOM" time="345640123">
  <OOM>
    <Action>FAIL_VIRTUAL_COMMIT</Action>
    <Resources>4096</Resources>
  </OOM>

```

이 레코드는 실패한 작업의 유형(커밋, 예약, 혹은 페이지 할당)과 요청된 메모리 크기를 알려줍니다.

RING_BUFFER_MEMORY_BROKER 와 내부 메모리 부족

내부 메모리 부족이 감지되면, 메모리 할당 원본이 되는 버퍼 풀을 사용하는 구성 요소의 메모리 부족 알림(low memory notification)이 설정됩니다. 이는 캐시와 또 다른 구성 요소로부터 페이지 반환을 허용하는 것입니다.

내부 메모리 부족은 **최대 서버 메모리** 옵션을 조정하는 경우 혹은 stolen 페이지가 버퍼 풀의 80%를 초과하는 경우에도 발생할 수 있습니다.

내부 메모리 부족 알림('Shrink')은, 다음 예제 코드와 같이 메모리 브로커(memory broker) 링 버퍼를 쿼리함으로써 알 수 있습니다.

```
select
    x.value('(/Record/@time)[1]', 'bigint') as [Time Stamp],
    x.value('(/Notification)[1]', 'varchar(100)')
    as [Last Notification]
from
    (select cast(record as xml)
     from sys.dm_os_ring_buffers
     where ring_buffer_type = 'RING_BUFFER_MEMORY_BROKER')
    as R(x)
order by
    [Time Stamp] desc
```

RING_BUFFER_BUFFER_POOL

이 링 버퍼는 버퍼 풀의 메모리 부족 상태를 포함한 중대한 버퍼 풀 실패를 나타내는 레코드를 포함합니다.

```
select record
from sys.dm_os_ring_buffers
where ring_buffer_type = 'RING_BUFFER_BUFFER_POOL'
```

record 는 다음과 같습니다:

```
<Record id="1234" type="RING_BUFFER_BUFFER_POOL"
time="345640123">
  < BufferPoolFailure id="FAIL_OOM">
    <CommittedCount>84344 </CommittedCount>
    <CommittedTarget>84350 </CommittedTarget >
    <FreeCount>20</FreeCount>
    <HashedCount>20345</HashedCount>
    <StolenCount>64001 </StolenCount>
    <ReservedCount>64001 </ReservedCount>
  </ BufferPoolFailure >
```

이 레코드는 실패 내용(FAIL_OOM, FAIL_MAP, FAIL_RESERVE_ADJUST, FAIL_LAZYWRITER_NO_BUFFERS)과 그 시점을 알려줍니다.

내부 가상 메모리 부족

VAS(가상 주소 공간) 소비는 `sys.dm_os_virtual_address_dump` DMV 를 사용해서 추적할 수 있습니다. 다음과 같은 뷰를 사용해서 VAS 요약 정보를 구할 수 있습니다.

```
-- 가상 주소 공간 요약 뷰
-- SQL Server 영역에 대한 목록을 생성
-- 예약되거나 빈 영역 수를 보여줌
CREATE VIEW VASummary AS
SELECT
    Size = VaDump.Size,
    Reserved = SUM(CASE(CONVERT(INT, VaDump.Base)^0)
    WHEN 0 THEN 0 ELSE 1 END),
    Free = SUM(CASE(CONVERT(INT, VaDump.Base)^0)
    WHEN 0 THEN 1 ELSE 0 END)
FROM
(
    --- 기준 주소 포인트가 0 인 경우를 제외하고,
    --- 할당 기준 주소별로 영역 크기 집계
    SELECT
        CONVERT(VARBINARY, SUM(region_size_in_bytes))
        AS Size,
        region_allocation_base_address AS Base
    FROM sys.dm_os_virtual_address_dump
    WHERE region_allocation_base_address <> 0x0
    GROUP BY region_allocation_base_address
    UNION
    --- 기준 주소 포인트가 0 인 경우, 그룹화없이 개별 반환
    SELECT CONVERT(VARBINARY, region_size_in_bytes),
    region_allocation_base_address
    FROM sys.dm_os_virtual_address_dump
    WHERE region_allocation_base_address = 0x0
)
AS VaDump
GROUP BY Size
```

다음 쿼리로 VAS 상태를 평가할 수 있습니다.

```
-- 모든 free 영역 가용 메모리 구하기
SELECT SUM(Size*Free)/1024 AS [Total avail mem, KB]
FROM VASummary
WHERE Free <> 0

-- 최대 가용 메모리 크기 구하기
SELECT CAST(MAX(Size) AS INT)/1024 AS [Max free size, KB]
FROM VASummary
WHERE Free <> 0
```

최대 가용 메모리 크기가 4MB 보다 작다면 VAS 부족을 경험할 수 있습니다. SQL Server 2005 는 VAS 부족을 모니터하고 이에 반응합니다. SQL Server 2000 은 VAS 부족 상태를 능동적으로 모니터하지 않지만, 가상 메모리 부족 오류가 발생할 때 캐시를 비우는 동작을 합니다.

메모리 오류에 대한 일반적인 문제 해결 단계

다음 목록은 메모리 오류 문제를 해결하는데 도움을 주는 일반적인 단계를 개략적으로 소개합니다.

1. 서버가 외부 메모리 부족 상태인지 확인합니다. 존재한다면 먼저 메모리 부족 문제를 해결한 뒤 해당 문제나 오류가 계속 발생하는지 확인합니다.

2. SQL Server: Buffer Manager, SQL Server: Memory Manger 의 성능 모니터 카운터를 수집합니다.
3. 메모리 구성 매개 변수(sp_configure), 쿼리당 최소 메모리, 최소/최대 서버 메모리, awe enabled, 그리고 메모리 페이지 잠그기 사용 권한을 확인합니다. 잘못된 설정을 찾고 필요에 따라 수정합니다. SQL Server 2005 에서 증가된 메모리 요구 사항을 검토합니다.
4. sp_configure 매개 변수 중 간접적으로 서버에 영향을 미칠 수 있는 비기본값이 있는지 확인합니다.
5. 내부 메모리 부족 여부를 확인합니다.
6. 메모리 오류 메시지를 만나면 DBCC MEMORYSTATUS 출력을 살펴봅니다.
7. 작업 부하(동시 세션 수, 실행 쿼리 수)를 확인합니다.

메모리 오류

701 – 시스템 메모리가 부족하여 이 쿼리를 실행할 수 없습니다.

원인

가장 일반적인 서버 메모리 부족 오류입니다. 메모리 할당 실패를 나타내며 현재 작업 부하 상에서 메모리 한계를 벗어난 경우를 포함해서 다양한 원인들이 존재합니다. SQL Server 2005 메모리 요구 사항 증가와 특정 구성 설정(최대 서버 메모리 옵션 같은)으로 사용자는 SQL Server 2000 보다 이 오류를 더 많이 만날 수 있습니다. 일반적으로 실패한 트랜잭션은 이 오류의 원인이 아닙니다.

문제 해결

오류의 일관성이나 반복성(동일 상태) 혹은 임시 발생(서로 다른 상태로 불특정 시점에 나타남) 여부에 관계 없이, 오류가 발생하는 동안 서버 메모리 분배를 조사합니다. 이 오류가 발생하면 진단 분석 쿼리 또한 실패할 가능성이 있습니다. 외부 요소 평가에서부터 조사를 시작하십시오. [메모리 오류에 대한 일반적인 문제 해결 단계](#)를 따릅니다.

해결 방안: 외부 메모리 부족 문제 제거. 최대 서버 메모리 설정 증가. 다음 명령 중 하나를 사용해서 캐시 비우기: DBCC FREESYSTEMCACHE, DBCC FREESESSIONCACHE, 혹은 DBCC FREEPROCACHE. 문제가 계속되면, 작업 부하를 줄입니다.

802 – 버퍼 풀에 사용할 수 있는 메모리가 부족합니다.

원인

이 오류가 반드시 메모리 부족 상태를 나타내는 것은 아닙니다. 버퍼 풀 메모리가 사용 중임을 나타냅니다. SQL Server 2005 에서는 드물게 발생할 것입니다.

문제 해결

701 오류에서 소개한 일반적인 문제 해결 단계의 권장 사항을 따릅니다.

8628 – 쿼리 최적화를 기다리는 중 시간이 초과되었습니다. 쿼리를 다시 실행하십시오.

원인

이 오류는 쿼리 컴파일 작업을 완료하는데 필요한 메모리 확보에 실패한 경우를 나타냅니다. 쿼리는 구문 해석, 대수 연산 처리, 그리고 최적화를 포함한 컴파일 처리를 거쳐야 합니다. 따라서 다른 쿼리와 메모리 리소스 경합이 일어납니다. 쿼리가 리소스를 기다리는 동안 정의된 시간을 초과하면 오류가 반환됩니다. 이 오류의 주된 원인은 서버에서 대량의 쿼리 컴파일이 발생하는 경우입니다.

문제 해결

1. 서버 메모리 소비로 인한 영향인지 알아보기 위해 일반적인 문제 해결 단계를 따릅니다.
2. 작업 부하를 검사합니다. 각 구성 요소별 소비되는 메모리 양을 확인합니다. (본 문서의 내부 실제 메모리 부족 참조)
3. DBCC MEMORYSTATUS 출력에서 각 게이트웨이 상의 대기자(waiters) 수를 점검합니다. (이 정보는 많은 메모리를 소비하는 다른 쿼리가 실행 중인지를 알려줍니다).

Small Gateway	Value
Configured Units	8
Available Units	8
Acquires	0
Waiters	0
Threshold Factor	250000
Threshold	250000
(6 개 행 적용됨)	
Medium Gateway	Value
Configured Units	2
Available Units	2
Acquires	0
Waiters	0
Threshold Factor	12
(5 개 행 적용됨)	
Big Gateway	Value
Configured Units	1
Available Units	1
Acquires	0
Waiters	0
Threshold Factor	8

4. 할 수 있다면 작업 부하를 줄입니다.

8645¹²– 쿼리를 실행하기 위해 메모리 리소스를 기다리는 중 시간이 초과되었습니다. 쿼리를 다시 실행하십시오.

¹² 제품에 포함된 한글 오류 메시지와는 다르게 번역하였습니다.

원인

이 오류는 메모리 집중한 쿼리가 동시에 많이 실행되고 있음을 나타냅니다. 정렬(**ORDER BY**)과 조인을 사용하는 쿼리는 실행하는 동안 많은 메모리를 소비할 수 있습니다. 최대 병렬 처리 수준 설정 값이 높거나 쿼리가 정렬되지 않은 인덱스를 가진 분할 테이블 상에서 동작하는 경우 요구되는 메모리 양은 상당히 증가합니다. 쿼리에서 필요한 메모리 리소스를 기다리는 동안 정의된 시간을 초과하면(기본값은 **sp_configure 'query wait'** 설정 값이나 쿼리 예측 비용의 25 배) 이 오류를 받게 됩니다. 일반적으로 이 오류를 받은 쿼리는 메모리를 소비하는 쿼리가 아닙니다.

문제 해결

1. 서버 메모리 상태를 평가하는 일반적인 단계를 따릅니다.
2. 의심되는 쿼리를 확인합니다: 분할 테이블을 사용하는 많은 수의 쿼리가 있다면, 정렬되지 않은 인덱스를 사용하는지 점검합니다, 조인 혹은 정렬을 포함한 쿼리가 많은지 점검합니다.
3. **sp_configure** 매개 변수인 최대 병렬 처리 수준과 쿼리 당 최소 메모리를 점검합니다. 최대 병렬 처리 수준을 낮추어 보거나 쿼리 당 최소 메모리가 높게 설정되어 있지 않는지 확인합니다. 높은 값이라면, 작은 쿼리조차도 지정된 양의 메모리를 요구하게 됩니다.
4. 쿼리가 **RESOURCE_SEMAPHORE** 로 대기 중인지 알아봅니다, 이는 이후에 차단에서 소개됩니다.

8651 – 최소 쿼리 메모리를 사용할 수 없어서 요청한 작업을 수행할 수 없습니다. ‘쿼리 당 최소 메모리’ 서버 구성 옵션의 구성 값을 줄이십시오.

원인

8645 오류와 유사한 원인입니다. 또한 일반적인 서버 메모리 부족 상태를 나타낼 수 있습니다. 쿼리 당 최소 메모리(**min memory per query**) 옵션이 너무 높은 경우 이 오류가 발생할 수 있습니다.

문제 해결

1. 일반적인 메모리 오류 문제 해결 단계를 따릅니다.
2. **sp_configure** 의 쿼리 당 최소 메모리 옵션 설정을 확인합니다.

I/O 병목

SQL Server 성능은 I/O 하위 시스템의 사용량에 따라 결정됩니다. 데이터베이스가 실제 메모리 크기에 적합하지 않다면 SQL Server 는 데이터베이스 페이지에서 버퍼 풀로 가져오거나 혹은 가져가는 동작이 빈번하게 일어날 것입니다. 이는 상당한 I/O 트래픽을 발생시킵니다. 더불어 로그 레코드 또한 트랜잭션이 커밋되기 전에 디스크로 기록됩니다. 마침내 SQL Server 는 중간 결과 저장, 정렬, 행 버전 관리 등의 다양한 목적으로 **tempdb** 를 사용합니다. 따라서 충분한 I/O 하위 시스템이 SQL Server 성능에 핵심이 됩니다.

tempdb 를 포함한 데이터 파일이 랜덤하게 액세스되는 반면, 로그 파일 액세스는 트랜잭션이 롤백되는 경우를 제외하고는 순차적입니다. 따라서 일반적인 규정대로, 보다 좋은 성능을 위해 로그 파일은 데이터 파일과 분리된 물리적 디스크 상에 두어야 합니다. 본 문서의 초점은 I/O 장치를

구성하는 방법이 아니라 I/O 병목이 있는지를 식별하기 위한 방법을 설명하는 것입니다. I/O 병목이 확인되면, I/O 하위 시스템의 재구성을 고려할 필요가 있습니다.

느린 I/O 하위 시스템을 가지고 있다면, 사용자들은 느린 응답 시간이나 시간 초과로 인한 작업 취소와 같은 성능 문제를 경험할 것입니다.

I/O 병목을 식별하기 위해서 다음 성능 카운터를 사용할 수 있습니다. 참고로, 이러한 평균값은 수집 간격을 짧게 가져야 합니다. 예를 들어, 60 초 간격으로 수집하는 경우엔 I/O 가 치솟는 특성을 알려주기가 어렵습니다. 또한, 하나의 카운터만으로 병목을 판단해서는 안됩니다. 여러 카운터 값을 참조해서 검증해야 합니다.

- **PhysicalDisk Object: Avg. Disk Queue Length** 물리적 디스크의 수집 기간 동안 큐에 대기한 평균 물리적 읽기와 쓰기 수입니다. I/O 하위 시스템의 부하가 크다면 더 많은 읽기/쓰기 작업이 대기 상태가 될 것입니다. SQL Server 가장 많이 사용되는 기간 동안 카운터가 지속적으로 2를 초과한다면 I/O 병목을 가정할 수 있습니다.
- **Avg. Disk Sec/Read** 디스크에서 데이터를 읽는 평균 시간(초)입니다.

10ms 이하	- 매우 좋음
10 - 20ms 사이	- 팬찮음
20 - 50ms 사이	- 느림, 주의 요망
50ms 이상	- 심각한 I/O 병목
- **Avg. Disk Sec/Write** 디스크에 데이터를 쓰는 평균 시간(초)입니다. 이전 가이드를 참조하십시오.
- **Physical Disk: %Disk Time** 디스크 드라이브가 읽기 및 쓰기 요청을 서비스하는데 소비한 경과 시간의 비율입니다. 일반적인 가이드로는 50% 이상일 경우 I/O 병목이라고 말할 수 있습니다.
- **Avg. Disk Reads/Sec** 디스크의 읽기 작업 비율입니다. 디스크 용량의 85%를 초과하지 않아야 합니다. 디스크 액세스 시간은 85%를 초과하면서 기하급수적으로 증가합니다.
- **Avg. Disk Writes/Sec** 디스크의 쓰기 작업 비율입니다. 디스크 용량의 85%를 초과하지 않아야 합니다. 디스크 액세스 시간은 85%를 초과하면서 기하급수적으로 증가합니다.

위 카운터를 사용할 때는, 다음 수식을 사용해서 RAID 구성에 대한 적정 값을 찾아야 합니다.

Raid 0 -- 디스크 당 I/O = (읽기 + 쓰기) / 디스크 수
 Raid 1 -- 디스크 당 I/O = [읽기 + (2 * 쓰기)] / 2
 Raid 5 -- 디스크 당 I/O = [읽기 + (4 * 쓰기)] / 디스크 수
 Raid 10 -- 디스크 당 I/O = [읽기 + (2 * 쓰기)] / 디스크 수
 예를 들어, 두 개의 물리적 디스크로 구성된 RAID-1 시스템에서 다음과 같은 카운터 값을 보인다면,

Disk Reads/sec	80
Disk Writes/sec	70
Avg. Disk Queue Length	5

이 경우, 디스크 당 I/O 는 $110 = (80 + (2 * 70)) / 2$ 이며 디스크 대기는 $2.5 = 5/2$ 이므로, I/O 병목 경계선에 있음을 나타냅니다.

또한 래치(latch) 대기를 검사해서 I/O 병목을 식별할 수도 있습니다. 래치 대기는 버퍼 풀에 없는 페이지를 읽거나 혹은 쓰기 위해 액세스하는 페이지에 대한 물리적인 I/O 대기입니다. 페이지를 버퍼 풀에서 찾지 못하면 비동기 I/O 를 지시하고 I/O 상태를 검사합니다. I/O 가 이미

완료되었으면, 정상적으로 진행합니다. 그렇지 않으면, 요청 작업의 유형에 따라 PAGEIOLATCH_EX 혹은 PAGEIOLATCH_SH 대기가 발생합니다. 다음 DMV 쿼리로 I/O 래치 대기 정보를 알 수 있습니다.

```
Select  wait_type,
        waiting_tasks_count,
        wait_time_ms,
        signal_wait_time_ms13
from    sys.dm_os_wait_stats
where   wait_type like 'PAGEIOLATCH%'
order by wait_type
```

wait_type	waiting_tasks_count	wait_time_ms	signal_wait_time_ms
PAGEIOLATCH_DT	0	0	0
PAGEIOLATCH_EX	1230	791	11
PAGEIOLATCH_KP	0	0	0
PAGEIOLATCH_NL	0	0	0
PAGEIOLATCH_SH	13756	7241	180
PAGEIOLATCH_UP	80	66	0

I/O가 완료되면 작업자가 실행 가능 큐에 놓입니다. I/O 완료 시간과 해당 작업자가 실제로 예약될 때까지의 시간 차이가 **signal_wait_time_ms** 열에 계산됩니다. **waiting_task_counts**와 **wait_time_ms**가 평소와는 상당히 다를 때 I/O 문제를 식별할 수 있습니다. 이를 위해, SQL Server가 원활하게 수행 중일 때 성능 카운터와 핵심 DMV 쿼리 결과에 대한 기준선을 가지고 있는 것이 중요합니다. **wait_types** 정보가 I/O 하위 시스템이 병목을 가지고 있는 나타낼 수 있지만, 물리적 디스크에 발생하는 문제에 대한 어떤 가시성을 제공하지는 않기 때문입니다.

현재 지연되고 있는 I/O 요청을 알아내기 위해 다음 DMV 쿼리를 사용할 수 있습니다. I/O 하위 시스템의 검진을 위해 이 쿼리를 정기적으로 실행합니다.

```
select
    database_id,
    file_id,
    io_stall,
    io_pending_ms_ticks,
    scheduler_address
from    sys.dm_io_virtual_file_stats(NULL, NULL)t1,
        sys.dm_io_pending_io_requests as t2
where   t1.file_handle = t2.io_handle
```

예제 출력은 다음과 같습니다. 특정 데이터베이스에 3가지 I/O 대기를 보여줍니다. **database_id**와 **file_id**를 이용해서 물리적 디스크를 알아낼 수 있습니다. **io_pending_ms_ticks**는 큐에서 대기 중인 개별 I/O의 총 시간을 나타냅니다.

¹³ 원본에는 생략되었던 열입니다. 결과와 맞추기 위해 추가합니다.

Database_id	File_Id	io_stall	io_pending_ms_ticks	scheduler_address
-----	-----	-----	-----	-----
6	1	10804	78	0x0227A040
6	1	10804	78	0x0227A040
6	2	101451	31	0x02720040

문제 해결

I/O 병목을 식별하면, 다음 중 하나 이상의 작업을 수행해서 해당 문제에 대처할 수 있습니다:

• SQL Server 메모리 구성을 검사합니다. 메모리가 부족하면 더 많은 I/O 오버헤드가 발생합니다. 다음 카운터를 조사해서 메모리 부족을 확인합니다.

- Buffer Cache hit ratio
- Page Life Expectancy
- Checkpoint pages/sec
- Lazywrites/sec

메모리 부족에 대한 추가 정보는 본 문서의 메모리 부족을 참조하십시오.

• I/O 대역폭 증가.

- 현재 디스크 어레이에 물리적인 드라이브를 추가하거나 더 빠른 드라이브로 대체합니다. 이것으로 읽기 및 쓰기 성능을 향상시키는데 도움을 줍니다.
- 더 빠른 혹은 새로운 I/O 컨트롤러를 추가합니다. 현재 컨트롤러에 캐시를 추가하는 것도 고려할 수 있습니다.

• 실행 계획을 통해서 어떤 계획이 많은 I/O 를 소비하는지 점검합니다. 더 좋은 계획(예를 들어, 인덱스)이 I/O 를 최소화할 수 있습니다. 인덱스가 없다면, 필요한 인덱스를 알아내기 위해 데이터베이스 엔진 튜닝 관리자를 실행해 볼 수 있습니다.

다음 DMV 쿼리로 I/O 의 대부분을 차지하는 해당 일괄 처리나 요청 작업을 찾을 수 있습니다. 물리적 쓰기는 제공되지 않음을 알 수 있습니다. 데이터베이스 작업 방식을 검토하는 것으로는 충분합니다. DML/DDDL 문장은 데이터 페이지를 바로 디스크에 쓰지 않습니다. 대신에, 디스크로의 물리적인 쓰기는 트랜잭션을 커밋하는 문장에 의해서만 수행됩니다. 일반적으로 물리적 쓰기는 검사점(Checkpoint)이나 SQL Server 지연 기록기(lazy writer)에 의해서 수행됩니다. DMV 쿼리는 I/O 의 대부분을 발생시키는 상위 5 개 요청을 찾아냅니다. 해당 쿼리가 더 적은 논리적 읽기를 수행하도록 튜닝함으로써 버퍼 풀 부족 문제를 해소할 수 있습니다. 다음에 수행되는 다른 요청이 필요한 데이터를 버퍼 풀에서 바로 찾을 수 있도록 도와줍니다. 결국엔 시스템 전체 성능이 향상됩니다.

```
select top 5
    (total_logical_reads/execution_count) as avg_logical_reads,
    (total_logical_writes/execution_count) as avg_logical_writes,
    (total_physical_reads/execution_count) as avg_phys_reads,
    Execution_count,
    statement_start_offset as stmt_start_offset,
    sql_handle,
    plan_handle
from sys.dm_exec_query_stats
order by
    (total_logical_reads + total_logical_writes) Desc
```


사용자 개체	<p>사용 세션에 의해서 명시적으로 생성되며 시스템 카탈로그에서 추적됩니다. 해당 항목들은 다음과 같습니다:</p> <ul style="list-style-type: none"> 테이블과 인덱스. 전역(global) 임시 테이블(##t1)과 인덱스. 로컬(local) 임시 테이블 (#t1)과 인덱스. <ul style="list-style-type: none"> 세션 범위. 현재 범위 내의 저장 프로시저 테이블 변수(@t1). <ul style="list-style-type: none"> 세션 범위. 현재 범위 내의 저장 프로시저.
내부 개체	<p>다음은 쿼리를 처리하기 위해 SQL Server 가 생성 및 제거하는 문장 범위 개체들입니다. 시스템 카탈로그에 의해서 추적되지 않습니다:</p> <ul style="list-style-type: none"> 작업 파일 (해시 조인) 정렬 수행 작업 테이블(커서, 스푼과 임시 대용량 개체 데이터 형식(LOB) 저장소) <p>최적화를 위해서 작업 테이블이 삭제되면, 하나의 IAM 페이지와 익스텐트는 새로운 작업 테이블에 사용되도록 보관됩니다.</p> <p>두 가지 예외가 있습니다. 일괄 처리 범위의 임시 LOB 저장소와 세션 범위의 커서 작업 테이블입니다.</p>
버전 관리 저장소	<p>새로운 행 버전 관리 저장을 위해서 사용됩니다. MARS, 온라인 인덱스, 트리거와 스냅샷 기반 격리 수준등이 행 버전 관리에 기반합니다. 이것은 SQL Server 2005 의 새 기능입니다.</p>
여유 공간	<p>tempdb 에서 사용 가능한 디스크 공간을 나타냅니다.</p>

tempdb 에 의해서 사용되는 전체 공간은 사용 개체 + 내부 개체 + 버전 관리 저장소 + 여유 공간에 해당합니다.

이 여유 공간은 성능 카운터의 free space in tempdb 와 같습니다.

tempdb 디스크 공간 모니터링

문제의 해결보다는 예방이 더 중요합니다. tempdb 사용 공간을 모니터링하기 위해 다음 성능 카운터를 사용할 수 있습니다.

- **Free Space in tempdb (KB).** 이 카운터는 **tempdb** 의 여유 공간을 킬로바이트 단위로 추적합니다. 관리자는 이 카운터를 통해서 **tempdb** 의 저장 공간 부족 여부를 판단할 수 있습니다.

그러나 위에서 정의한 각 범주별로 **tempdb** 에서 사용되는 디스크 공간을 확인하는 것이 보다 흥미롭고 유익한 조사 대상입니다

다음 쿼리는 사용자와 내부 개체에 의해서 사용된 **tempdb** 공간을 반환합니다. 현재는 **tempdb** 에 대해서만 정보를 제공합니다.

```
Select
    SUM (user_object_reserved_page_count)*8 as user_objects_kb,
    SUM (internal_object_reserved_page_count)*8 as
internal_objects_kb,
    SUM (version_store_reserved_page_count)*8 as
version_store_kb,
    SUM (unallocated_extent_page_count)*8 as freespace_kb
From sys.dm_db_file_space_usage
Where database_id = 2
```

다음은 출력 결과입니다 (KB 단위).

user_objets_kb	internal_objects_kb	version_store_kb	free space_kb
-----	-----	-----	-----
8736	128	64	
448			

혼합 익스텐트내 페이지에 대해서는 계산하지 않고 있음을 참고하십시오.
혼합 익스텐트내 페이지는 사용자 개체와 내부 개체에 의해서 할당될 수 있습니다.

디스크 공간 문제 해결

사용자 개체, 내부 개체, 그리고 버전 저장소 모두 **tempdb** 디스크 공간 문제를 유발할 수 있습니다. 이번 절에서는, 각 범주별 문제 해결 방법을 검토합니다.

사용자 개체

사용자 개체는 특정 세션에 소유되지 않으므로, 개체를 생성하는 응용 프로그램의 요구 사항을 이해하고, 필요에 따라 **tempdb** 크기를 조정합니다. 사용자 개체에 대한 사용 공간은 `exec sp_spaceused @objname='<user-object>'` 를 실행해서 알 수 있습니다.


```

DECLARE userobj_cursor CURSOR FOR
select
    sys.schemas.name + '.' + sys.objects.name
from sys.objects, sys.schemas
where object_id > 100 and
    type_desc = 'USER_TABLE'and
    sys.objects.schema_id = sys.schemas.schema_id
go

open userobj_cursor
go

declare @name varchar(256)
fetch userobj_cursor into @name
while (@@FETCH_STATUS = 0)
begin
    exec sp_spaceused @objname = @name
    fetch userobj_cursor into @name
end
close userobj_cursor

```

버전 저장소

SQL Server 2005는 기존 기능과 새로운 기능을 구현하는데 사용되는 행 버전 관리 프레임워크를 제공합니다. 현재, 다음 기능들이 행 버전 관리 프레임워크를 사용합니다. 각 기능에 대한 추가 정보는, SQL Server 온라인 설명서를 참조하십시오.

- 트리거
- MARS
- 온라인 인덱스
- 행 버전 기반 격리 수준: 데이터베이스 수준의 옵션 설정이 필요

행 버전 관리는 세션간의 공유됩니다. 행 버전 생성자는 행 버전이 재사용되는 경우의 제어권을 가지고 있지 않습니다. 행 버전을 모두 정리할 때 이를 방해하는 가장 오랜 시간 실행되는 트랜잭션을 찾아서 가능한 제거하는 것이 필요합니다.

다음 쿼리는 버전 저장소에서 버전에 따라 가장 오랜 시간 실행되는 트랜잭션 상위 2건을 반환합니다.

```

select top 2
    transaction_id,
    transaction_sequence_num,
    elapsed_time_seconds
from sys.dm_tran_active_snapshot_database_transactions
order by elapsed_time_seconds DESC

```

다음 출력은 6523 초 동안 활성화 상태인 Transaction ID 8906 이고 XSN 3 인 트랜잭션 에제를 보여줍니다.

transaction_id	transaction_sequence_num	elapsed_time_seconds
8609	3	6523
20156	25	783

두 번째 트랜잭션은 상대적으로 짧은 기간이므로, 첫 번째 트랜잭션을 제거하는 것으로 버전 저장소의 상당한 양을 정리할 수 있습니다. 그러나, 얼마나 많은 공간이 반환될지는 예측할 수 없습니다. 필요한 공간을 정리하기 위해서는 좀 더 많은 트랜잭션을 제거할 필요가 있습니다.

버전 저장소를 포함해서 **tempdb** 용량을 적절히 계산하거나, 혹은 스냅숏 격리 상태에서 오랫동안 실행되는 트랜잭션이나, **read-committed-snapshot** 상태에서 오랫동안 실행되는 쿼리를 제거함으로써 이러한 문제를 줄일 수 있습니다. 다음 수식을 사용해서 대략적인 버전 저장소 크기를 예측할 수 있습니다. (인수 2는 최악의 경우, 장시간 실행되는 트랜잭션 2 개가 겹치는 시나리오를 포함시킨 것입니다.)

[Size of version store] = 2 * [version store data generated per minute] *

[longest running time (minutes) of the transaction]

모든 데이터베이스에서 격리 수준에 기반한 행 버전 관리가 가능하며, 트랜잭션을 위해 분단위로 생성되는 버전 저장소 데이터는 분단위로 생성되는 로그와 대략 유사합니다. 그러나 몇 가지 예외가 있습니다: 유일한 차이점은 변경에 대한 로그입니다; 그리고 새로게 입력된 데이터 행은 버전 관리되지 않습니다 다만 대량 로그 작업 및 복구 모델이 전체 복구로 설정되지 않은 경우에 따라서 로그가 발생할 수 있습니다.

좀 더 정확한 계산을 위해 **Version Generation Rate** 와 **Version Cleanup Rate** 성능 카운터를 사용할 수도 있습니다. **Version Cleanup Rate** 가 0 이면 이는 버전 저장소 정리를 방해하는 장시간 수행되는 트랜잭션이 존재함을 암시합니다.

덧붙여, **tempdb** 공간 부족 오류가 발생하기 전에, SQL Server 2005 는 버전 저장소 축소를 강제하도록 마지막 시도를 합니다. 축소 작업 중에, 아직 행 버전이 생성되지 않은 가장 긴 트랜잭션은 희생자(victims)로 표시됩니다. 이를 통해 사용되는 버전 저장소 공간을 확보할 수 있습니다. 메시지 3967¹⁵이 그와 같은 각 희생자 트랜잭션을 위해 오류 로그에 생성됩니다. 트랜잭션이 희생자로 표시되면 버전 저장소에서 해당 행 버전을 읽거나 새로운 것을 생성할 수 없게 됩니다. 메시지 3966 은 희생자 트랜잭션이 행 버전을 읽으려고 할 때 그 트랜잭션이 롤백되면서 발생합니다. 버전 저장소의 축소가 성공하면 **tempdb** 에서 추가 공간이 확보됩니다. 그렇지 않으면, **tempdb** 는 저장소가 부족하게 됩니다.

내부 개체

내부 개체는 각 문장에 의해서 생성되고 제거됩니다. **tempdb** 가 지나치게 크다면 어떤 세션이나 작업이 그렇게 공간을 많이 소비하는지 해당 주범을 찾아서 해결할 필요가 있습니다.

SQL Server 2005 는 두 가지 추가 DMV 를 제공합니다:

sys.dm_db_session_space_usage 와 **sys.dm_db_task_space_usage** 는 각각 세션과 작업별로 할당된 **tempdb** 공간을 추적합니다. 작업(task)은 세션

¹⁵ 한글 오류 메시지에는 victim 에 대한 직접적인 해석 대신 “교착 상태가 발생...”으로 표현되어 있습니다. 해당 메시지를 직접 참조하십시오.

범위 내에서 실행되므로, 작업에 의해서 사용된 공간은 작업이 완료된 후에 세션에 포함됩니다.

다음 쿼리를 사용해서 내부 개체를 할당하고 있는 상위 세션을 알 수 있습니다. 이 쿼리는 세션에서 완료된 작업만 포함됨을 참고하십시오.

```
select
    session_id,
    internal_objects_alloc_page_count,
    internal_objects_dealloc_page_count
from sys.dm_db_session_space_usage
order by internal_objects_alloc_page_count DESC
```

현재 활성 작업을 포함해서 내부 개체를 할당하고 있는 상위 사용자 세션을 알아 내기 위해 다음 쿼리를 사용할 수 있습니다.

```
SELECT
    t1.session_id,
    (t1.internal_objects_alloc_page_count + task_alloc) as
allocated,
    (t1.internal_objects_dealloc_page_count + task_dealloc)
as
    deallocated
from sys.dm_db_session_space_usage as t1,
    (select session_id,
        sum(internal_objects_alloc_page_count)
        as task_alloc,
        sum (internal_objects_dealloc_page_count) as
        task_dealloc
        from sys.dm_db_task_space_usage group by session_id) as
t2
where t1.session_id = t2.session_id and t1.session_id >50
order by allocated DESC
```

다음은 출력 결과입니다.

session_id	allocated	deallocated
52	5120	5136
51	16	0

내부 개체 할당이 많이 발생하는 작업이 별도로 분리되어 있다면, 해당 Transact-SQL 문장이 무엇인지 그리고 어떤 쿼리 계획인지를 알아내서 보다 상세한 분석이 가능합니다.

과도한 DDL 및 할당 작업

다음과 같은 상황에서 두 가지 원인의 경합이 **tempdb**에 일어날 수 있습니다.

- 대량의 임시 테이블 및 테이블 변수의 생성과 삭제는 메터 데이터의 경합을 유발할 수 있습니다. SQL Server 2005에서는, 메터 데이터 경합을 최소화하기 위해 로컬 임시 테이블 및 테이블 변수가 캐시에 저장합니다. 그러나 다음 조건을 만족해야 하며, 그렇지 않을 경우 캐시되지 않습니다.
 - 테이블에 명명된 제약 조건이 없음.
 - 테이블 생성 문장 이후에 다른 DDL이 없음 (예, CREATE INDEX, CREATE STATISTICS).
- 일반적으로, 대부분의 임시/작업 테이블은 힙(Heap)에 해당합니다; 그러므로 insert, delete, 혹은 drop 작업은 페이지의 사용 가능한 공간(Page Free Space, PFS) 페이지에 많은 경합을 유발합니다. 이러한 테이블 대부분이 64KB 미만이므로 혼합 익스텐트를 사용하게 되며, 이로 인해 공유 전역 할당 맵(Shared Global Allocation Map, SGAM) 페이지에 많은 경합이 일어납니다. SQL Server 2005는 할당 경합을 최소화하기 위해서 하나의 데이터 페이지와 하나의 IAM 페이지를 캐시에 저장합니다. 이러한 캐시 동작은 SQL Server 2000에서 작업 테이블을 대상으로 이미 수행되었습니다.

SGAM과 PFS 페이지는 데이터 파일에서 고정된 간격으로 발생하므로, 해당 리소스 내용을 찾기가 어렵지 않습니다. 예를 들어, 2:1:1은 **tempdb**의 첫 번째 PFS 페이지(데이터베이스-id = 2, 파일-id = 1, 페이지-id = 1)를 나타내며, 2:1:3은 첫 번째 SGAM 페이지를 나타냅니다. SGAM 페이지는 511232 페이지 간격으로 발생하며 PFS 페이지는 8088 페이지 단위로 발생합니다. 이러한 정보를 통해서 **tempdb**의 모든 파일에 모든 PFS 및 SGAM 페이지를 찾을 수 있습니다. 이러한 페이지에 래치를 얻기 위해 작업이 대기하는 시간을, **sys.dm_os_waiting_tasks**를 통해서 볼 수 있습니다. 래치 대기는 일시적이므로 차후에 분석을 위해서는 이 테이블을 자주 쿼리하고(대략 10초 간격) 해당 데이터를 수집해야 합니다. 예를 들어, 다음 쿼리는 **analysis** 데이터베이스의 **waiting_tasks** 테이블로 **tempdb** 페이지에 대기 중인 모든 작업 정보를 입력합니다.

```
-- 현재 시간 구하기
declare @now datetime
select @now = getdate()

-- 차후 분석을 위해 테이블에 입력
insert into analysis..waiting_tasks
select
    session_id,
    wait_duration_ms,
    resource_description,
    @now
from sys.dm_os_waiting_tasks
where wait_type like 'PAGE%LATCH_%' and
    resource_description like '2:%'
```

tempdb 페이지에 래치를 얻기 위해 대기 중인 작업을 볼 때, 이것이 PFS나 SGAM 페이지 때문인지 분석할 수 있습니다. 그렇다면, 이는 **tempdb**에 할당 경합을 암시하는 것입니다. **tempdb**의 다른 페이지에

경합이 보이며 그 페이지가 시스템 테이블에 속한다면 이것은 과도한 DDL 작업으로 인한 경합을 암시합니다.

다음 성능 카운터를 통해서 이례적인 임시 개체 할당이나 위치 처리 활동의 증가를 모니터할 수 있습니다.

- SQL Server:Access Methods\Workfiles Created /Sec
- SQL Server:Access Methods\Worktables Created /Sec
- SQL Server:Access Methods\Mixed Page Allocations /Sec
- SQL Server:General Statistics\Temp Tables Created /Sec
- SQL Server:General Statistics\Temp Tables for destruction

문제 해결

tempdb 경합이 과도한 DDL 작업 때문이라면, 사용 중인 응용 프로그램을 살펴보고 DDL 작업의 최소화가 가능한지 알아봅니다. 다음 제안 사항들을 시도해 볼 수 있습니다.

- 저장 프로시저 범위의 임시 테이블을 사용한다면, 저장 프로시저 외부로 이동시킬 수 있는지를 고려합니다. 그렇지 않으면, 저장 프로시저가 실행할 때마다 임시 테이블의 생성 및 삭제가 일어날 것입니다.
- 쿼리 계획이 많은 임시 개체, 스푼, 정렬, 혹은 작업 테이블을 생성하는지 점검합니다. 임시 개체를 제거할 필요가 있습니다. 예를 들어, ORDER BY 에 사용되는 열에 인덱스를 생성하면 정렬을 제거할 수 있습니다.

SGAM 과 PFS 페이지에 경합이 원인이라면, 다음 사항을 시도함으로써 경감시킬 수 있습니다:

- 모든 디스크와 파일에 걸쳐 작업 부하를 분산시키도록 **tempdb** 데이터 파일 크기를 동일한 크기로 증가시킵니다. 이상적으로는 CPU 개수(프로세스 선호도를 고려해서)와 동일한 수의 데이터 파일을 만듭니다.
- 혼합 익스텐트 할당을 하지 않도록 TF-1118¹⁶를 사용합니다.

느린 쿼리

느리거나 오래 실행되는 쿼리는 리소스를 과도하게 소비할 수 있으며 쿼리 차단으로 이어집니다.

과도한 리소스 소비는 CPU 리소스에만 국한되지 않고, I/O 저장소 대역폭과 메모리 대역폭 또한 포함될 수 있습니다. SQL Server 쿼리가 WHERE 절을 통해서 결과 집합을 제한함으로써 테이블 전체 스캔을 피하도록 설계되었더라도, 해당 쿼리를 지원하는 적절한 인덱스가 없는 경우에 예상한대로 수행되지 않게 됩니다. 또한, 사용자 입력에 따라 응용 프로그램에서 동적으로 WHERE 절이 구성되는 경우, 기존 인덱스가 모든 경우의 조건을 다룰 수가 없게 됩니다. Transact-SQL 문장에 의한 과도한 CPU, I/O, 그리고 메모리 소비는 본 문서 전반부에서 다루어졌습니다.

¹⁶ 추적 플래그(Trace Flag)를 말합니다.

인덱스 부재와 함께 사용되지 않는 인덱스도 존재할 수 있습니다. 모든 인덱스는 유지 관리 대상이므로, 쿼리 성능에는 영향을 미치지 않더라도, DML 쿼리에는 영향을 줍니다.

논리적 읽기를 위한 대기 상태 혹은 쿼리를 차단하고 있는 시스템 리소스에 대한 대기 상태로 인해 쿼리가 느리게 실행될 수 있습니다. 차단의 원인은 서투른 응용 프로그램 설계, 잘못된 쿼리 계획, 유용한 인덱스 부재, 그리고 작업 부하를 고려해서 적절히 구성되지 못한 SQL Server 인스턴스와 같은 것들이 될 수 있습니다.

이번 절은 느린 쿼리의 두 가지 원인, 차단과 인덱스 문제에 초점을 맞춥니다.

차단

차단은 주로 논리적 읽기에 대한 차단입니다. 이는 특정 리소스에 X 잠금 획득을 위한 대기나 래치 같은 저 수준 동기화 개체로 인한 대기와 같습니다.

논리적 읽기 대기는 이미 잠긴 리소스에 호환되지 않은 다른 잠금을 요청할 때 발생합니다. 이는 Transact-SQL 문장을 실행할 때 트랜잭션 격리 수준에 따라 데이터 일관성을 제공하기 위해서 필요하지만, 최종 사용자에게는 SQL Server 가 느리게 실행된다는 인식을 줍니다. 쿼리가 차단된다는 것이 시스템에 어떤 리소스를 소비하는 것은 아니므로, 시간은 오래 걸리지만 리소스 소비는 작은 쿼리를 찾을 수 있습니다. 동시성 제어 및 차단에 대한 상세 정보는 SQL Server 온라인 설명서를 참조하십시오.

저 수준의 동기화 개체에 대한 대기는 시스템이 작업 부하를 처리할 수 있도록 구성되지 못한 결과입니다.

차단/대기에 대한 공통적인 시나리오는:

- 차단 주범 식별
- 오래 걸리는 차단 식별
- 개체 별 차단
- 페이지 래치 문제
- 차단으로 인한 전반적인 성능 평가를 위해 SQL Server 대기 사용

요청을 처리하는데 필요한 시스템 리소스(혹은 잠금)가 현재 이용할 수 없는 상태이면 SQL Server 세션은 대기 상태에 놓입니다. 다른 말로 하자면, 리소스가 미해결 요청에 대한 큐를 가지는 것입니다. DMV 는 리소스 대기 중인 세션에 대한 정보를 제공합니다.

SQL Server 2005 는 보다 상세하고 일관된 대기 정보를 제공합니다, SQL Server 2000 은 76 개의 대기 유형을 지원한 반면 SQL Server 2005 는 대략 125 개 이상의 대기 유형을 보고해 줍니다. DMV 는 이러한 정보들을 두 가지 범위로 나누어 제공합니다. SQL Server 전체에 누적된 대기를 위한 **sys.dm_os_wait_statistics**¹⁷와, 세션 단위로 분류된 대기를 위한 세션 범위의 **sys.dm_os_waiting_tasks** 입니다. 다음의 DMV 는 특정 리소스에 대기 중인 작업의 대기 큐에 대한 상세 정보를 제공합니다. 예를 들어,

¹⁷ 온라인 설명서에 찾을 수 없으며, 쿼리로도 실행되지 않습니다. 역자의 판단으로 내용 상 sys.dm_os_wait_stats 해당될 것으로 보입니다.


```

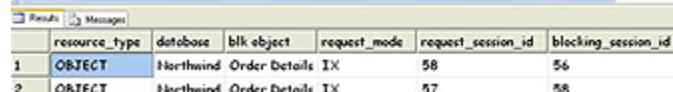
select
    request_session_id as spid,
    resource_type as rt,
    resource_database_id as rdb,
    (case resource_type
        WHEN 'OBJECT' then
object_name(resource_associated_entity_id)
        WHEN 'DATABASE' then ' '
        ELSE (select object_name(object_id)
            from sys.partitions
            where hobt_id=resource_associated_entity_id)
    END) as objname,
    resource_description as rd,
    request_mode as rm,
    request_status as rs
from sys.dm_tran_locks

```

다음은 출력 결과입니다.

spid	rt	rdb	objname	rd	rm	rs
56	DATABASE	9				
53	DATABASE	9				
56	PAGE	9	t_lock	1:143	IS	G
53	PAGE	9	t_lock	1:143	IX	G
53	PAGE	9	t_lock	1:153	IX	G
56	OBJECT	9	t_lock		IS	G
53	OBJECT	9	t_lock		IX	G
53	KEY	9	t_lock	(a400c34cb		
53	RID	9	t_lock	1:143:3	X	G
56	RID	9	t_lock	1:143:3	S	

사실 저장 프로시저 **sp_block** 에서 보여지는대로 위 두 DMV 를 조인할 수 있습니다. 그림 1 은 차단된 세션과 그 세션을 차단하는 세션 목록을 보여줍니다. 스크린샷은 부록 B 에서 찾을 수 있습니다. 필요하다면 저장 프로시저 **sp_block** 를 실행하거나 제거할 수 있습니다. 세션과 잠금



	resource_type	database	blk object	request_mode	request_session_id	blocking_session_id
1	OBJECT	Northwind	Order Details IX	IX	56	56
2	OBJECT	Northwind	Order Details IX	IX	57	56

그림 1: sp_block 결과[원본 크기 이미지로 보기](#)

SQL Server 2000에서는, 다음 문장을 사용해서 차단된 spid에 대한 정보를 알 수 있습니다.

```
select * from master..sysprocesses where blocked <> 0
```

저장 프로시저 sp_lock으로 관련된 잠금 정보를 볼 수 있습니다.

오래 걸리는 차단 식별

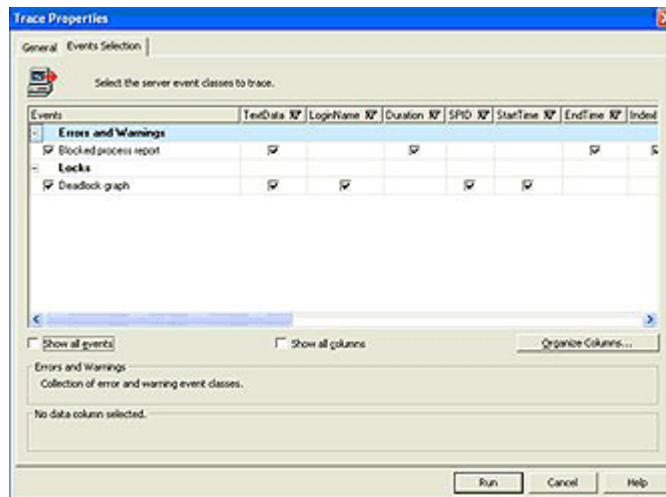
이전에 언급한대로, SQL Server에서 차단은 일반적인 것이며 트랜잭션의 일관성을 유지하기 위해 필요한 논리적 잠금의 결과입니다. 그러나, 잠금에 대한 대기가 한계를 초과하면 응답 시간에 영향을 미칩니다. 오래 걸리는 차단을 식별하기 위해 BlockedProcessThreshold 구성 매개 변수를 사용하면 서버 범위의 차단 임계값을 구성할 수 있습니다. 임계값은 초 단위 시간을 정의합니다. 차단이 임계값을 초과하면 SQL Trace에서 추적할 수 있도록 이벤트가 발생합니다.

예를 들어, 다음과 같이 SQL Server Management Studio에서 차단 프로세스 임계값을 200초로 구성할 수 있습니다:

1. Execute Sp_configure 'blocked process threshold', 200
2. Reconfigure with override

차단 프로세스 임계값을 구성하고 나서, SQL Trace나 프로파일러를 가지고 추적 이벤트를 캡처합니다.

3. SQL Trace를 사용한다면, event_id = 137로 sp_trace_setevent를 설정합니다.
4. SQL Server 프로파일러를 사용한다면, (Errors and Warnings 개체 밑에) Blocked Process Report 이벤트 클래스를 선택합니다. 그림 2를 참조하십시오.

**그림 2: 오래 걸리는 차단과 교착 상태 추적**[원본 크기 이미지로 보기](#)

참고 이것은 가벼운 추적 작업에 해당합니다. (1) 차단이 임계값을 초과하거나 (2) 교착상태가 발생할 때만 이벤트가 캡처됩니다. 차단 잠금에 대해 매 200초 간격으로 추적 이벤트가 발생합니다. 이는

600 초 동안 차단된 단일 잠금의 경우 3 번의 추적 이벤트가 발생하는 것입니다. 그림 3 을 참조하십시오.

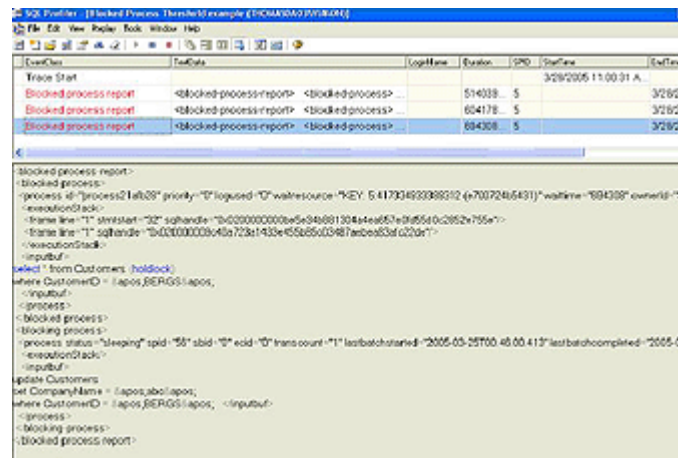


그림 3: 차단 보고 > 차단 임계값

[원본 크기 이미지로 보기](#)

추적 이벤트에는 차단하고 있는 프로세스와 차단 당하는 프로세스 양쪽의 전체 SQL 문장을 포함합니다. 위 경우 “Update Customers” 문장이 “Select from Customers” 문장을 차단하고 있습니다.

상대적으로, SQL Server 2000에서는 차단 시나리오를 검사하기 위해 **Sysprocesses** 시스템 테이블을 조회하고 결과를 처리하는 별도의 코드를 필요로 합니다.

sys.dm_db_index_operational_stats 를 사용한 개체 별 차단

SQL Server 2005 의 새로운 DMV 인 **sys.dm_db_index_operational_stats** 는 차단을 포함해서 총체적인 인덱스 사용 통계 정보를 제공합니다. 차단으로 표현하자면, 테이블, 인덱스, 분할 별 상세한 잠금 통계 정보를 제공합니다. 예를 들어 주어진 인덱스나 테이블에 대한 액세스, 잠금(row_lock_count), 차단(row_lock_wait_count), 그리고 대기(row_lock_wait_in_ms) 이력 정보를 포함합니다.

이 DMV 를 통해서 얻을 수 있는 정보들은 다음과 같습니다:

- 잠금 보유 누적 개수, 예) 행 혹은 페이지.
- 차단 혹은 대기 누적 개수, 예) 행, 페이지.
- 차단 혹은 대기 누적 시간, 예) 행, 페이지.
- 페이지 래치 대기 누적 개수.
- **page_latch_wait** 대기 누적 시간: 순차 키 입력과 같은 경우의 특정 페이지에 대한 경합과 관련됩니다. 그런 경우, 마지막 페이지에 동시에 여러 프로세스가 쓰기 작업을 수행하기 위해 배타적 페이지 래치를 요구하게 되므로 마지막 페이지에 핫 스팟(hot spot)이 일어나게 됩니다. 이것이 **Pagelatch** 대기로 나타나게 됩니다.
- **page_io_latch_wait** 대기 누적 시간: I/O 래치는 사용자가 요청한 페이지가 버퍼 풀에 존재하지 않을 때 발생합니다. 느린 I/O 하위 시스템, 혹은 용량을 초과한 I/O 하위 시스템은 높은 **PageIOlatch** 를 경험할 수 있으며 이것이 실제 I/O 문제를 나타냅니다. 이러한 문제는

캐시 플러시나 인덱스 부재 등에 의해서도 일어날 수 있습니다.

- 페이지 래치 대기 시간¹⁸.

차단 관련 정보 이외에도, 인덱스 액세스에 대한 추가 정보들이 제공됩니다.

- 액세스 유형, 예) 범위, 단일 조회.
- 리프 수준에서의 insert, update, delete.
- 리프 수준 상위 수준에서의 insert, update, delete. 리프 상위의 동작은 인덱스 유지 관리 작업입니다. 상위 수준에는 각 리프 페이지의 첫 행이 입력됩니다. 리프에 새로운 페이지가 할당되면, 해당 페이지의 첫 행이 상위 페이지에 입력됩니다.
- 리프 수준에서의 페이지 병합은 행 삭제로 인해 할당이 해제된 빈 페이지를 나타냅니다.
- 인덱스 유지 관리. 리프 수준 상위에서의 페이지 병합은 리프에서 행 삭제로 인해 중간 수준 페이지가 빈 상태로 남겨지는, 할당 해제된 빈 페이지입니다. 리프 페이지의 첫 행은 상위 수준에 입력되어 있습니다. 리프 수준에서 충분한 수의 행이 삭제되면, 리프 페이지에 대한 첫 행을 입력으로 포함하고 있던 중간 수준의 인덱스 페이지들은 빈 페이지가 됩니다. 이것은 리프 상위 수준에서의 병합을 유발합니다.

이 정보는 SQL Server 인스턴스가 시작되면서부터 누적됩니다. 이 정보는 인스턴스가 재시작하면 사라지며, 또한 재설정하는 방법도 없습니다. 이 DMV에 의해서 반환되는 데이터는 해당 힙이나 인덱스를 나타내는 메타 데이터 캐시가 사용 가능한 경우에만 존재합니다. 힙이나 인덱스를 메타 데이터 캐시로 가져갈 때마다 각 열 값은 0으로 설정됩니다. 통계 정보는 메타 데이터 캐시에서 캐시 개체가 제거될 때까지 누적됩니다. 따라서, 이후 분석을 위해서는 주기적으로 데이터를 수집해서 저장해 두어야 합니다.

부록 B 인덱스 작업 데이터를 수집하는데 사용할 수 있는 저장 프로시저들을 제공합니다. 원하는 주기로 해당 데이터를 분석할 수 있습니다. 다음은 부록 B에 소개한 저장 프로시저를 사용하는 단계입니다.

1. **init_index_operational_stats** 를 사용해서 indexstats 를 초기화합니다.
2. **insert_indexstats** 를 사용해서 기준 상태를 캡처합니다.
3. 작업 부하를 발생시킵니다.
4. **insert_indexstats** 를 사용해서 인덱스 통계 정보에 대한 최종 정보를 캡처합니다.
5. 수집된 인덱스 통계 정보를 분석하기 위해서, 저장 프로시저 **get_indexstats** 를 실행합니다. 평균 잠금 개수(인덱스와 분할에 대한 **Row_lock_count**), 차단, 그리고 인덱스 별 대기 정보를 생성합니다. 높은 값의 **blocking %** 혹은 높은 평균 대기는 인덱스 전략 문제나 쿼리 문제를 나타냅니다.

다음은 저장 프로시저를 사용해서 얻을 수 있는 정보들의 몇 가지 예제¹⁹를 보여줍니다.

¹⁸ 원문에서 실수로 반복된 항목이 아닐까 생각됩니다.

- 인덱스 사용량을 정렬 기준으로 모든 데이터베이스의 상위 5 개 인덱스 출력.

```
exec get_indexstats
    @dbid=-1,
    @top='top 5',
    @columns='index, usage',
    @order='index usage'
```

- 잠금 승격을 시도한 적이 있는 상위 5 개 인덱스 잠금 승격 출력.

```
exec get_indexstats
    @dbid=-1,
    @top='top 5',
    @order='index lock promotions',
    @threshold='[index lock promotion attempts] > 0'
```

- 평균 잠금 대기 시간이 2ms 를 초과하는 상위 5 개 단일 조회 작업에 대해 wait, scan, singleton 열을 포함해서 출력.

```
exec get_indexstats
    @dbid=5,
    @top='top 5',
    @columns='wait,scan,singleton',
    @order='singleton lookups',
    @threshold='[avg row lock wait ms] > 2'
```

- 모든 데이터베이스에서 행 잠금 대기가 1 을 초과하는 상위 10 개에 대해 avg, wait 열을 포함해서 대기 시간을 기준으로 출력.

```
exec get_indexstats
    @dbid=-1,
    @top='top 10 ',
    @columns='wait,row',
    @order='row lock wait ms',
    @threshold='[row lock waits] > 1'
```

- 평균 행 잠금 대기 시간을 정렬 기준으로 상위 5 개 행 출력.

```
exec get_indexstats
    @dbid=-1,
    @top='top 5',
    @order='avg row lock wait ms'
```

- 평균 페이지 래치 대기 시간을 정렬 기준으로 상위 5 개 행 출력.

```
exec get_indexstats
    @dbid=-1,
    @top='top 5',
    @order='avg page latch wait ms'
```

- 평균 페이지 I/O 래치 시간을 정렬 기준으로 상위 3%의 인덱스 통계 출력.

```
exec get_indexstats
    @dbid=-1,
    @top='top 5 percent',
    @order='avg pageio latch wait ms',
    @threshold='[pageio latch waits] > 0'
```

¹⁹ 원본 예제는 각 예제별 설명과 실제 코드 간의 차이가 있어서, 역자 임의로 조정했습니다.

- db=5 에서 block%를 기준으로 0.1 을 초과하는 상위 10 위 정보 출력.

```

--- get indexstats for all dbid, include columns containing usage,wait
exec get_indexstats @dbid=-1,@columns='rlock,usage,wait',@order='rlock,wait',
@threshold='[rlock,wait]>0'

```

start time	end time	duration (hh:mm:ss...)	Report
2005-04-27 12:57...	2005-04-27 13:16...	00:18:43:467	all databases, include only columns containing rlock,wait

db_id	db_name	object	indexid	pro	index usage	rlocks	rlock waits	block %	rlock wait ms	avg rlock
6	Northwind	Orders	1	1	530745	530854	8	0.00	330	36.7
6	Northwind	Orders	3	1	538	73074	8	0.01	3164	351.6
6	Northwind	Order Details	1	1	531437	1580909	1	0.00	10	5.0
6	Northwind	Orders	2	1	130	528706	3	0.00	1192	298.0

그림 4: 차단 분석 보고

[원본 크기 이미지로 보기](#)

SQL Server 2000에서는 개체나 인덱스 사용량에 대한 통계 정보를 제공하지 않습니다.

차단으로 인한 전반적인 성능 평가를 위해 SQL 대기 사용

SQL Server 2000은 76개의 대기 유형을 보고합니다. SQL Server 2005는 응용 프로그램 성능 추적을 위해서 100개 이상의 추가 대기 유형을 제공합니다. 사용자 연결이 대기 중인 동안, SQL Server는 대기 시간을 누적합니다. 예를 들어, 응용 프로그램이 I/O, 잠금, 혹은 메모리 같은 리소스를 요청하면 해당 리소스를 사용 가능할 때까지 기다릴 수 있습니다. 작업 부하가 발생했을 때 해당하는 성능 프로필을 얻을 수 있도록 이러한 대기 정보들이 모든 연결에 걸쳐 요약되고 분류됩니다. 따라서, SQL 대기 유형은 응용 프로그램 작업 부하로부터 발생하는 사용자(혹은 쓰레드) 대기 정보를 식별하고 분류합니다.

다음 쿼리는 SQL Server에 상위 10위 대기 정보를 보여줍니다. 이 대기 정보는 누적되는 정보이지만 DBCC SQLPERF ([sys.dm_os_wait_stats], clear)를 사용해서 재설정할 수가 있습니다.

```

select top 10 *
from sys.dm_os_wait_stats
order by wait_time_ms desc

```

다음은 출력 결과입니다. 주목할 만한 몇 가지 핵심 요점은:

- 지연 기록기(lazy writer)와 같이 백그라운드 쓰레드로 인한 몇 가지 대기는 정상적입니다.
- 몇 가지 세션은 SH 잠금을 얻기 위해 장 시간 대기했습니다.
- *signal wait*은 작업자(worker)가 리소스에 액세스를 허가 받았을 때부터 CPU 예약을 획득한 시점까지의 대기 시간입니다. 오랜 signal wait은 높은 CPU 경합을 암시합니다.

wait_type	waiting_tasks_count	wait_time_ms	max_wait_time_ms
signal_wait_time_ms			

LAZYWRITER_SLEEP	415088	415048437	1812
156			
SQLTRACE_BUFFER_FLUSH	103762	415044000	4000
0			
LCK_M_S	6	25016812	23240921
0			
WRITELOG	7413	86843	187
406			

LOGMGR_RESERVE_APPEND	82	82000	1000
0			
SLEEP_BPOOL_FLUSH	4948	28687	31
15			
LCK_M_X	1	20000	20000
0			
PAGEIOLATCH_SH	871	11718	140
15			
PAGEIOLATCH_UP	755	9484	187
0			
IO_COMPLETION	636	7031	203
0			

대기 정보를 분석하기 위해서, 정기적으로 데이터를 수집해야 합니다. 부록 B에서 두 가지 저장 프로시저를 제공합니다.

- **Track_waitstats.** 원하는 샘플 숫자와 샘플 간격으로 데이터를 수집할 수 있습니다. 다음은 호출 예제입니다.

```
exec dbo.track_waitstats_2005 @num_samples=6
    ,@delay_interval=30
    ,@delay_type='s'
    ,@truncate_history='y'
    ,@clear_waitstats='y'
```

- **Get_waitstats.** 이전 단계에서 수집된 데이터를 분석합니다. 다음은 호출 예제입니다.

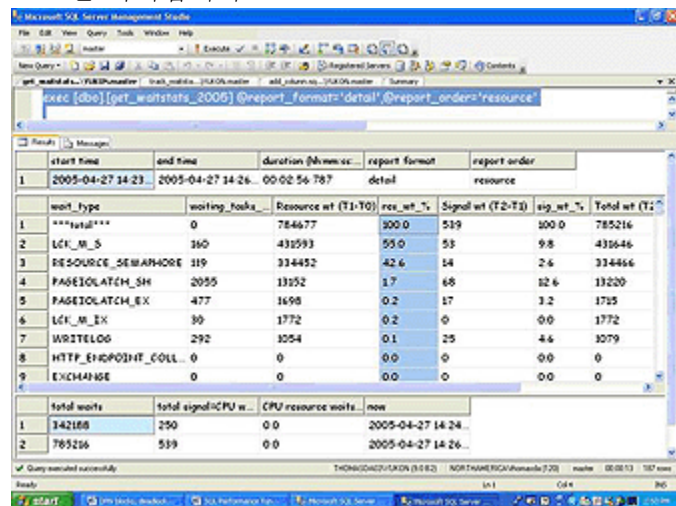


그림 5: 대기에 대한 통계 정보 분석 리포트

[원본 크기 이미지로 보기](#)

그림 5의 대기 분석 리포트 예제는 성능 문제가 차단(LCK_M_S)과 메모리 할당(RESOURCE_SEMAPHORE)에 있음을 알려줍니다. 특히 전체 대기 중 메모리 요구로 인한 문제가 43%인 반면, 55%가 공유 잠금입니다. 개체 별 차단 분석은 경합의 핵심 요점을 식별합니다.

인덱스 사용량 모니터링

쿼리 성능의 또 다른 관점은 DML 쿼리, 데이터를 삭제, 추가, 변경하는 쿼리와 관련됩니다. 테이블에 더 많은 인덱스를 정의하면 데이터를 수정하는데 더 많은 리소스를 필요로 합니다. 트랜잭션에서 점유하는 잠금과 결합해서, 오랜 시간 동안의 변경 작업은 동시성을 방해하게

됩니다. 따라서, 시간이 지난 뒤에 응용 프로그램에 의해서 사용되는 인덱스가 무엇인지 알아내는 것이 매우 중요합니다. 그리고 나서, 사용되지 않는 인덱스로 인해 데이터베이스 스키마가 불필요하게 크지 않은지를 알아내야 합니다.

SQL Server 2005 는 **sys.dm_db_index_usage_stats** 동적 관리 뷰를 새로 제공합니다. 이 뷰는 어떤 인덱스가 사용되고 있는지, 해당 인덱스가 사용자 쿼리에 의해서 사용되는지 시스템 작업에 의해서만 사용되는지를 알려줍니다. 쿼리를 실행할 때마다, 쿼리에서 사용한 쿼리 계획에 따라서 이 뷰내의 열 값이 증가됩니다. 데이터는 SQL Server 실행 동안 수집됩니다. 이 DMV 의 데이터는 메모리에서만 유지되며 영구히 보존되지는 않습니다. 따라서 SQL Server 인스턴스가 중지되면, 데이터는 손실됩니다. 정기적으로 데이터를 수집하고 저장해서 이후 분석을 위해 사용합니다.

인덱스에 대한 작업은 사용자 유형과 시스템 유형으로 분류됩니다. 사용자 유형은 **SELECT** 와 **INSERT/UPDATE/DELETE** 작업입니다. 시스템 유형은 **DBCC** 나 **DDL** 명령 혹은 **update statistics** 와 같은 명령입니다. 각 문장의 범주별로 열이 구별됩니다:

- 인덱스에 대한 검색 (**user_seeks** 나 **system_seeks**)
- 인덱스에 대한 조회 작업 (**user_lookups** 나 **system_lookups**)
- 인덱스에 대한 스캔 작업 (**user_scans** 나 **system_scans**)
- 인덱스에 대한 변경 작업 (**user_updates** 나 **system_updates**)

인덱스 액세스 유형별로, 마지막 액세스 시간 또한 알 수 있습니다.

인덱스 자체는 세 가지 칼럼, **database_id**, **object_id**, **index_id** 를 통해서 알 수 있습니다. **index_id = 0** 은 힙 테이블을 나타내며, **index_id = 1** 은 클러스터형 인덱스를, **index_id > 1** 인 경우는 비클러스터형 인덱스를 나타냅니다.

SQL Server 2005 에서 검색, 스캔, 그리고 조회 동작에 대한 규칙과 정의는 다음과 같습니다.

- 검색: 데이터 액세스를 위해서 B-트리 구조가 사용된 카운트를 나타냅니다. 한 행을 찾기 위해서 인덱스 각 수준에서 적은 페이지를 읽거나 혹은 기가바이트 데이터나 수 백만 행을 읽기 위해 인덱스 페이지 반을 읽었더라도 문제가 되지 않습니다. 따라서 인덱스를 사용한 것으로 해당 범주에 누적됩니다.
- 스캔: B-트리 인덱스 중 하나를 사용하지 않고 테이블의 데이터 계층을 사용해서 스캔을 한 카운트를 나타냅니다. 테이블에 인덱스가 없는 경우가 이에 해당합니다. 인덱스가 존재하지만 쿼리에 의해서 사용되지 않은 경우에도 해당합니다.
- 조회: 비클러스터형 인덱스가 사용된 경우를 ‘검색’으로 정의한다면, 추가로 클러스터형 인덱스(혹은 힙)를 사용해서 데이터를 조회한 경우를 나타냅니다. 이는 SQL Server 2000 에서 *책갈피 조회(bookmark lookup)* 시나리오를 나타냅니다. 이는 테이블 액세스에 사용된 비클러스터형 인덱스가 쿼리 **select** 목록에 열과 **where** 절에 정의된 열을 모두 포함하지 못하는 시나리오를 나타냅니다. SQL Server 는 비클러스터형 인덱스에 대해 **user_seeks** 열 값을 증가시키고 클러스터형 인덱스 항목의 **user_lookups** 열에 더합니다. 이 카운트는 테이블에 비클러스터형 인덱스가 많은 경우 매우 높은 값을 가질 수 있습니다. 클러스터형 인덱스에 **user_seeks** 값이 매우 높고, 더불어 **user_lookups** 값도 높으며 특정 비클러스터형 인덱스의

`user_seeks` 값 또한 매우 높다면, 높은 값을 가진 비클러스터형 인덱스를 클러스터형 인덱스로 바꾸어 만들면 쿼리 성능을 더 향상시킬 수도 있습니다.

다음 DMV 쿼리로 모든 데이터베이스의 모든 개체에 대해 인덱스 사용량에 대한 유용한 정보를 알 수 있습니다.

```
select object_id, index_id, user_seeks, user_scans,
       user_lookups
from sys.dm_db_index_usage_stats
order by object_id, index_id
```

다음 결과는 특정 테이블 정보를 보여줍니다.

object_id	index_id	user_seeks	user_scans	user_l ookups
-----	-----	-----	-----	-----

521690298	1	0		251
123				
521690298	2	123		0
0				

이 경우, 인덱스를 사용하지 않고 테이블의 데이터 계층을 직접 액세스한 쿼리가 251 번 실행되었습니다. 첫 번째 비클러스터형 인덱스를 사용해서 테이블을 액세스한 쿼리가 123 번 실행되었습니다. 이는 쿼리의 `select` 열이나 `WHERE` 절에 지정된 열이 비클러스터형 인덱스에 모두 존재하지 않으므로 클러스터형 인덱스를 123 번 조회한 것입니다.

가장 관심있게 살펴볼 범주는 ‘사용자 유형 문장’입니다. ‘시스템 범주’내에 나타나는 사용량은 기존 인덱스에 결과로 보여질 수 있습니다. 인덱스가 존재하지 않으면, 통계 정보를 변경하거나 일관성을 검사할 필요가 없기 때문입니다. 따라서, 임시(ad hoc) 문장이나 사용자 응용 프로그램에 의한 사용량을 나타내는 4 개의 칼럼에 집중해서 분석합니다.

SQL Server 가 시작된 이후에 특정 테이블에서 사용되지 않은 인덱스에 대한 정보를 얻기 위해서, 해당 테이블이 존재하는 데이터베이스에서 다음 쿼리를 실행합니다.

```
select i.name
from sys.indexes i
where i.object_id=object_id('<table_name>') and
      i.index_id NOT IN (select s.index_id
                        from sys.dm_db_index_usage_stats s
                        where s.object_id=i.object_id and
                              i.index_id=s.index_id and
                              database_id = <dbid> 20)
```

사용되지 않은 모든 인덱스 정보는 다음 쿼리로 알 수 있습니다:

²⁰ 이 부분을 실제 Database ID 로 변경하거나, `DB_ID()` 함수를 사용합니다.

```
select object_name(object_id), i.name
from sys.indexes i
where i.index_id NOT IN (select s.index_id
                        from sys.dm_db_index_usage_stats s
                        where s.object_id=i.object_id and
                              i.index_id=s.index_id and
                              database_id = <dbid> )
order by object_name(object_id) asc
```

이 경우, 테이블 이름과 인덱스 이름은 테이블 이름에 따라서 정렬됩니다.

실제로 이 동적 관리 뷰의 목적은 오랫동안 인덱스의 사용량을 관찰하는 것입니다. 뷰나 쿼리를 수행한 결과를 매일 혹은 특정 주기로 저장하고 시간이 지나면서 각각 비교를 하는 것입니다. 분기별 보고나 회계연도 보고와 같이 한 달 혹은 특정 기간 동안 사용되지 않는 인덱스를 알아낸다면, 데이터베이스에서 해당 인덱스를 삭제할 수 있습니다.

결론

추가 정보는 다음을 참조하십시오:

<http://www.microsoft.com/technet/prodtechnol/sql/default.mspx>

부록 A: DBCC MEMORYSTATUS 설명

DBCC MEMORYSTATUS 명령을 사용해서 얻을 수 있는 몇 가지 정보가 있습니다. 그러나, 일부는 동적 관리 뷰(DMV)를 통해서도 얻을 수 있습니다.

SQL Server 2000 DBCC MEMORYSTATUS 에 대한 기술 정보

<http://support.microsoft.com/?id=271624>

SQL Server 2005 DBCC MEMORYSTATUS 에 대한 기술 정보

<http://support.microsoft.com/?id=907877>

부록 B: 차단 스크립트(Blocking Scripts)

부록은 본 문서에서 참조한 저장 프로시저의 소스 목록입니다. 필요에 따라 수정해서 사용하십시오.

sp_block

```
create proc dbo.sp_block (@spid bigint=NULL)
as
-- This stored procedure is provided "AS IS" with no warranties, and
-- confers no rights.
-- Use of included script samples are subject to the terms specified at
-- http://www.microsoft.com/info/copyright.htm
--
-- T. Davidson
-- This proc reports blocks
-- 1. optional parameter @spid
--

select
    t1.resource_type,
    'database'=db_name(resource_database_id),
    'blk object' = t1.resource_associated_entity_id,
    t1.request_mode,
    t1.request_session_id,
    t2.blocking_session_id
from
    sys.dm_tran_locks as t1,
    sys.dm_os_waiting_tasks as t2
where
    t1.lock_owner_address = t2.resource_address and
    t1.request_session_id = isnull(@spid,t1.request_session_id)
```

인덱스 사용 정보 분석

아래 저장 프로시저들은 인덱스 사용량 분석에 사용할 수 있습니다.

get_indexstats

```
create proc dbo.get_indexstats
    (@dbid smallint=-1
    ,@top varchar(100)=NULL
    ,@columns varchar(500)=NULL
    ,@order varchar(100)='lock waits'
    ,@threshold varchar(500)=NULL)
as
--
-- This stored procedure is provided "AS IS" with no warranties, and
-- confers no rights.
-- Use of included script samples are subject to the terms specified at
-- http://www.microsoft.com/info/copyright.htm
--
-- T. Davidson
-- This proc analyzes index statistics including accesses, overhead,
-- locks, blocks, and waits
--
-- Instructions: Order of execution is as follows:
-- (1) truncate indexstats with init_indexstats
-- (2) take initial index snapshot using insert_indexstats
-- (3) Run workload
```

```

--      (4) take final index snapshot using insert_indexstats
--      (5) analyze with get_indexstats

-- @dbid limits analysis to a database
-- @top allows you to specify TOP n
-- @columns is used to specify what columns from
--      sys.dm_db_index_operational_stats will be included in
-- the report
--      For example, @columns='scans,lookups,waits' will include
-- columns
--      containing these keywords
-- @order used to order results
-- @threshold used to add a threshold,
--      example: @threshold='[block %] > 5' only include if
-- blocking is over 5%
--
----- definition of some computed columns returned
-- [blk %] = percentage of locks that cause blocks e.g. blk% = 100 * lock
-- waits / locks
-- [index usage] = range_scan_count + singleton_lookup_count +
-- leaf_insert_count
-- [nonleaf index overhead]=nonleaf_insert_count + nonleaf_delete_count +
-- nonleaf_update_count
-- [avg row lock wait ms]=row_lock_wait_in_ms/row_lock_wait_count
-- [avg page lock wait ms]=page_lock_wait_in_ms/page_lock_wait_count
-- [avg page latch wait ms]=page_latch_wait_in_ms/page_latch_wait_count
-- [avg pageio latch wait
-- ms]=page_io_latch_wait_in_ms/page_io_latch_wait_count
-----
-----
--- Case 1 - only one snapshot of sys.dm_db_operational_index_stats was
-- stored in
---      indexstats. This is an error - return errormsg to user
--- Case 2 - beginning snapshot taken, however some objects were not
-- referenced
---      at the time of the beginning snapshot. Thus, they will
-- not be in the initial
---      snapshot of sys.dm_db_operational_index_stats, use 0 for
-- starting values.
---      Print INFO msg for informational purposes.
--- Case 3 - beginning and ending snapshots, beginning values for all
-- objects and indexes
---      this should be the normal case, especially if SQL Server
-- is up a long time
-----
-----
set nocount on
declare @orderby varchar(100), @where_dbid_is varchar(100), @temp
varchar(500), @threshold_temptab varchar(500)
declare @cmd varchar(max), @col_stmt varchar(500), @addcol varchar(500)
declare @begintime datetime, @endtime datetime, @duration datetime,
@mincount int, @maxcount int

select @begintime = min(now), @endtime = max(now) from indexstats

if @begintime = @endtime

```

```

begin
    print 'Error: indexstats contains only 1 snapshot of
sys.dm_db_index_operational_stats'
    print 'Order of execution is as follows: '
    print '    (1) truncate indexstats with init_indexstats'
    print '    (2) take initial index snapshot using insert_indexstats'
    print '    (3) Run workload'
    print '    (4) take final index snapshot using insert_indexstats'
    print '    (5) analyze with get_indexstats'
    return -99
end

select @mincount = count(*) from indexstats where now = @ begintime
select @maxcount = count(*) from indexstats where now = @ endtime

if @mincount < @maxcount
begin
    print 'InfoMsg1: sys.dm_db_index_operational_stats only contains
entries for objects referenced since last SQL re-cycle'
    print 'InfoMsg2: Any newly referenced objects and indexes captured
in the ending snapshot will use 0 as a beginning value'
end

select @top = case
    when @top is NULL then ''
    else lower(@top)
end,
    @where_dbid_is = case (@dbid)
    when -1 then ''
    else ' and i1.database_id = ' + cast(@dbid as varchar(10))
end,
--- thresholding requires a temp table
    @threshold_temptab = case
    when @threshold is NULL then ''
    else ' select * from #t where ' + @threshold
end
--- thresholding requires temp table, add 'into #t' to select statement
select @temp = case (@threshold_temptab)
    when '' then ''
    else ' into #t '
end
select @orderby=case(@order)
when 'leaf inserts' then 'order by [' + @order + ']'
when 'leaf deletes' then 'order by [' + @order + ']'
when 'leaf updates' then 'order by [' + @order + ']'
when 'nonleaf inserts' then 'order by [' + @order + ']'
when 'nonleaf deletes' then 'order by [' + @order + ']'
when 'nonleaf updates' then 'order by [' + @order + ']'
when 'nonleaf index overhead' then 'order by [' + @order + ']'
when 'leaf allocations' then 'order by [' + @order + ']'
when 'nonleaf allocations' then 'order by [' + @order + ']'
when 'allocations' then 'order by [' + @order + ']'
when 'leaf page merges' then 'order by [' + @order + ']'
when 'nonleaf page merges' then 'order by [' + @order + ']'
when 'range scans' then 'order by [' + @order + ']'
when 'singleton lookups' then 'order by [' + @order + ']'

```

```

when 'index usage' then 'order by [' + @order + ']'
when 'row locks' then 'order by [' + @order + ']'
when 'row lock waits' then 'order by [' + @order + ']'
when 'block %' then 'order by [' + @order + ']'
when 'row lock wait ms' then 'order by [' + @order + ']'
when 'avg row lock wait ms' then 'order by [' + @order + ']'
when 'page locks' then 'order by [' + @order + ']'
when 'page lock waits' then 'order by [' + @order + ']'
when 'page lock wait ms' then 'order by [' + @order + ']'
when 'avg page lock wait ms' then 'order by [' + @order + ']'
when 'index lock promotion attempts' then 'order by [' + @order + ']'
when 'index lock promotions' then 'order by [' + @order + ']'
when 'page latch waits' then 'order by [' + @order + ']'
when 'page latch wait ms' then 'order by [' + @order + ']'
when 'pageio latch waits' then 'order by [' + @order + ']'
when 'pageio latch wait ms' then 'order by [' + @order + ']'
else ''
end

if @orderby <> '' select @orderby = @orderby + ' desc'
select
    'start time'=@begintime,
    'end time'=@endtime,
    'duration (hh:mm:ss:ms)'=convert(varchar(50),
    @endtime-@begintime,14),
    'Report'=case (@dbid)
        when -1 then 'all databases'
        else db_name(@dbid)
    end +
    case
        when @top = '' then ''
        when @top is NULL then ''
        when @top = 'none' then ''
        else ', ' + @top
    end +
    case
        when @columns = '' then ''
        when @columns is NULL then ''
        when @columns = 'none' then ''
        else ', include only columns containing ' + @columns
    end +
    case(@orderby)
        when '' then ''
        when NULL then ''
        when 'none' then ''
        else ', ' + @orderby
    end +
    case
        when @threshold = '' then ''
        when @threshold is NULL then ''
        when @threshold = 'none' then ''
        else ', threshold on ' + @threshold
    end

select @cmd = ' select i2.database_id, i2.object_id, i2.index_id,
i2.partition_number '

```

```

select @cmd = @cmd + ' , begintime=case min(i1.now) when max(i2.now) then
NULL else min(i1.now) end '
select @cmd = @cmd + '      , endtime=max(i2.now) '
select @cmd = @cmd + ' into #i '
select @cmd = @cmd + ' from indexstats i2 '
select @cmd = @cmd + ' full outer join '
select @cmd = @cmd + '      indexstats i1 '
select @cmd = @cmd + ' on i1.database_id = i2.database_id '
select @cmd = @cmd + ' and i1.object_id = i2.object_id '
select @cmd = @cmd + ' and i1.index_id = i2.index_id '
select @cmd = @cmd + ' and i1.partition_number = i2.partition_number '
select @cmd = @cmd + ' where i1.now >= '' ' +
convert(varchar(100),@begintime, 109) + ''''
select @cmd = @cmd + ' and i2.now = '' ' + convert(varchar(100),@endtime,
109) + ''''
select @cmd = @cmd + ' ' + @where_dbid_is + ' '
select @cmd = @cmd + ' group by i2.database_id, i2.object_id, i2.index_id,
i2.partition_number '
select @cmd = @cmd + ' select ' + @top + ' i.database_id,
db_name=db_name(i.database_id),
object=isnull(object_name(i.object_id),i.object_id), indid=i.index_id,
part_no=i.partition_number '

exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' , [leaf inserts]=i2.leaf_insert_count -
        isnull(i1.leaf_insert_count,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns, @col_stmt=' ,
        [leaf deletes]=i2.leaf_delete_count -
        isnull(i1.leaf_delete_count,0)'

select @cmd = @cmd + @addcol

exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' , [leaf updates]=i2.leaf_update_count -
isnull(i1.leaf_update_count,0)'

select @cmd = @cmd + @addcol

exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' , [nonleaf inserts]=i2.nonleaf_insert_count -
isnull(i1.nonleaf_insert_count,0)'

select @cmd = @cmd + @addcol

exec dbo.add_column
    @add_stmt=@addcol out,

```

```

        @cols_containing=@columns,
        @col_stmt=' ,[nonleaf deletes]=i2.nonleaf_delete_count -
isnull(i1.nonleaf_delete_count,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[nonleaf updates]=i2.nonleaf_update_count -
isnull(i1.nonleaf_update_count,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[nonleaf index overhead]=(i2.nonleaf_insert_count -
isnull(i1.nonleaf_insert_count,0)) + (i2.nonleaf_delete_count -
isnull(i1.nonleaf_delete_count,0)) + (i2.nonleaf_update_count -
isnull(i1.nonleaf_update_count,0))'

select @cmd = @cmd + @addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[leaf allocations]=i2.leaf_allocation_count -
isnull(i1.leaf_allocation_count,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[nonleaf allocations]=i2.nonleaf_allocation_count -
isnull(i1.nonleaf_allocation_count,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[allocations]=(i2.leaf_allocation_count -
isnull(i1.leaf_allocation_count,0)) + (i2.nonleaf_allocation_count -
isnull(i1.nonleaf_allocation_count,0))'

select @cmd = @cmd + @addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[leaf page merges]=i2.leaf_page_merge_count -
isnull(i1.leaf_page_merge_count,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[nonleaf page merges]=i2.nonleaf_page_merge_count -
isnull(i1.nonleaf_page_merge_count,0)'

```



```

select @cmd = @cmd + @addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[range scans]=i2.range_scan_count -
isnull(i1.range_scan_count,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing= @columns,
    @col_stmt=' ,[singleton lookups]=i2.singleton_lookup_count -
isnull(i1.singleton_lookup_count,0)'

select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[index usage]=(i2.range_scan_count -
isnull(i1.range_scan_count,0)) + (i2.singleton_lookup_count -
isnull(i1.singleton_lookup_count,0)) + (i2.leaf_insert_count -
isnull(i1.leaf_insert_count,0))'
select @cmd = @cmd + @addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[row locks]=i2.row_lock_count -
isnull(i1.row_lock_count,0)'
select @cmd = @cmd + @addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[row lock waits]=i2.row_lock_wait_count -
isnull(i1.row_lock_wait_count,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[block %]=cast (100.0 * (i2.row_lock_wait_count -
isnull(i1.row_lock_wait_count,0)) / (1 + i2.row_lock_count -
isnull(i1.row_lock_count,0)) as numeric(5,2))'

select @cmd = @cmd + @addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[row lock wait ms]=i2.row_lock_wait_in_ms -
isnull(i1.row_lock_wait_in_ms,0)'

select @cmd = @cmd + @addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[avg row lock wait ms]=cast ((1.0*(i2.row_lock_wait_in_ms
- isnull(i1.row_lock_wait_in_ms,0)))/(1 + i2.row_lock_wait_count -

```

```

isnull(i1.row_lock_wait_count,0)) as numeric(20,1))'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[page locks]=i2.page_lock_count -
isnull(i1.page_lock_count,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[page lock waits]=i2.page_lock_wait_count -
isnull(i1.page_lock_wait_count,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[page lock wait ms]=i2.page_lock_wait_in_ms -
isnull(i1.page_lock_wait_in_ms,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[avg page lock wait ms]=cast
((1.0*(i2.page_lock_wait_in_ms - isnull(i1.page_lock_wait_in_ms,0)))/(1 +
i2.page_lock_wait_count - isnull(i1.page_lock_wait_count,0)) as
numeric(20,1))'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[index lock promotion
attempts]=i2.index_lock_promotion_attempt_count -
isnull(i1.index_lock_promotion_attempt_count,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[index lock promotions]=i2.index_lock_promotion_count -
isnull(i1.index_lock_promotion_count,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[page latch waits]=i2.page_latch_wait_count -
isnull(i1.page_latch_wait_count,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,
    @col_stmt=' ,[page latch wait ms]=i2.page_latch_wait_in_ms -
isnull(i1.page_latch_wait_in_ms,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
    @add_stmt=@addcol out,
    @cols_containing=@columns,

```

```

        @col_stmt=' ,[avg page latch wait ms]=cast
((1.0*(i2.page_latch_wait_in_ms - isnull(i1.page_latch_wait_in_ms,0)))/(1
+ i2.page_latch_wait_count - isnull(i1.page_latch_wait_count,0)) as
numeric(20,1))'
select @cmd = @cmd +@addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[pageio latch waits]=i2.page_io_latch_wait_count -
isnull(i1.page_latch_wait_count,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[pageio latch wait ms]=i2.page_io_latch_wait_in_ms -
isnull(i1.page_latch_wait_in_ms,0)'
select @cmd = @cmd +@addcol
exec dbo.add_column
        @add_stmt=@addcol out,
        @cols_containing=@columns,
        @col_stmt=' ,[avg pageio latch wait ms]=cast
((1.0*(i2.page_io_latch_wait_in_ms -
isnull(i1.page_io_latch_wait_in_ms,0)))/(1 + i2.page_io_latch_wait_count -
isnull(i1.page_io_latch_wait_count,0)) as numeric(20,1))'

select @cmd = @cmd +@addcol
select @cmd = @cmd + @temp
select @cmd = @cmd + ' from #i i '
select @cmd = @cmd + ' left join indexstats i1 on i.begintime = i1.now and
i.database_id = i1.database_id and i.object_id = i1.object_id and
i.index_id = i1.index_id and i.partition_number = i1.partition_number '

select @cmd = @cmd + ' left join indexstats i2 on i.endtime = i2.now and
i.database_id = i2.database_id and i.object_id = i2.object_id and
i.index_id = i2.index_id and i.partition_number = i2.partition_number '
select @cmd = @cmd + ' ' + @orderby + ' '
select @cmd = @cmd + @threshold_temptab
exec ( @cmd )
go

```

insert_indexstats

```

create proc insert_indexstats (@dbid smallint=NULL,
                              @objid int=NULL,
                              @indid int=NULL,
                              @partitionid int=NULL)

as
--
-- This stored procedure is provided "AS IS" with no warranties, and
confers no rights.
-- Use of included script samples are subject to the terms specified at
http://www.microsoft.com/info/copyright.htm
-- This stored procedure stores a snapshot of
sys.dm_db_index_operational_stats into the table indexstats
-- for later analysis by the stored procedure get_indexstats. Please note
that the indexstats table has an additional
-- column to store the timestamp when the snapshot is taken

```

```
--
-- T. Davidson
-- snapshot sys.dm_db_index_operational_stats
--
declare @now datetime
select @now = getdate()
insert into indexstats
    (database_id
    ,object_id
    ,index_id
    ,partition_number
    ,leaf_insert_count
    ,leaf_delete_count
    ,leaf_update_count
    ,leaf_ghost_count
    ,nonleaf_insert_count
    ,nonleaf_delete_count
    ,nonleaf_update_count
    ,leaf_allocation_count
    ,nonleaf_allocation_count
    ,leaf_page_merge_count
    ,nonleaf_page_merge_count
    ,range_scan_count
    ,singleton_lookup_count
    ,forwarded_fetch_count
    ,lob_fetch_in_pages
    ,lob_fetch_in_bytes
    ,lob_orphan_create_count
    ,lob_orphan_insert_count
    ,row_overflow_fetch_in_pages
    ,row_overflow_fetch_in_bytes
    ,column_value_push_off_row_count
    ,column_value_pull_in_row_count
    ,row_lock_count
    ,row_lock_wait_count
    ,row_lock_wait_in_ms
    ,page_lock_count
    ,page_lock_wait_count
    ,page_lock_wait_in_ms
    ,index_lock_promotion_attempt_count
    ,index_lock_promotion_count
    ,page_latch_wait_count
    ,page_latch_wait_in_ms
    ,page_io_latch_wait_count
    ,page_io_latch_wait_in_ms,
    now)
select database_id
    ,object_id
    ,index_id
    ,partition_number
    ,leaf_insert_count
    ,leaf_delete_count
    ,leaf_update_count
    ,leaf_ghost_count
    ,nonleaf_insert_count
    ,nonleaf_delete_count
```

```

,nonleaf_update_count
,leaf_allocation_count
,nonleaf_allocation_count
,leaf_page_merge_count
,nonleaf_page_merge_count
,range_scan_count
,singleton_lookup_count
,forwarded_fetch_count
,lob_fetch_in_pages
,lob_fetch_in_bytes
,lob_orphan_create_count
,lob_orphan_insert_count
,row_overflow_fetch_in_pages
,row_overflow_fetch_in_bytes
,column_value_push_off_row_count
,column_value_pull_in_row_count
,row_lock_count
,row_lock_wait_count
,row_lock_wait_in_ms
,page_lock_count
,page_lock_wait_count
,page_lock_wait_in_ms
,index_lock_promotion_attempt_count
,index_lock_promotion_count
,page_latch_wait_count
,page_latch_wait_in_ms
,page_io_latch_wait_count
,page_io_latch_wait_in_ms
,@now
from sys.dm_db_index_operational_stats(@dbid,@objid,@indid,@partitionid)
go

```

init_index_operational_stats

```

CREATE proc dbo.init_index_operational_stats
as
--
-- This stored procedure is provided "AS IS" with no warranties, and
-- confers no rights.
-- Use of included script samples are subject to the terms specified at
-- http://www.microsoft.com/info/copyright.htm
--
-- T. Davidson
--
-- create indexstats table if it doesn't exist, otherwise truncate
--
set nocount on
if not exists (select 1 from dbo.sysobjects where
id=object_id(N'[dbo].[indexstats]') and OBJECTPROPERTY(id, N'IsUserTable')
= 1)
    create table dbo.indexstats (
        database_id smallint NOT NULL
        ,object_id int NOT NULL
        ,index_id int NOT NULL
        ,partition_number int NOT NULL
        ,leaf_insert_count bigint NOT NULL
        ,leaf_delete_count bigint NOT NULL

```

```

,leaf_update_count bigint NOT NULL
,leaf_ghost_count bigint NOT NULL
,nonleaf_insert_count bigint NOT NULL
,nonleaf_delete_count bigint NOT NULL
,nonleaf_update_count bigint NOT NULL
,leaf_allocation_count bigint NOT NULL
,nonleaf_allocation_count bigint NOT NULL
,leaf_page_merge_count bigint NOT NULL
,nonleaf_page_merge_count bigint NOT NULL
,range_scan_count bigint NOT NULL
,singleton_lookup_count bigint NOT NULL
,forwarded_fetch_count bigint NOT NULL
,lob_fetch_in_pages bigint NOT NULL
,lob_fetch_in_bytes bigint NOT NULL
,lob_orphan_create_count bigint NOT NULL
,lob_orphan_insert_count bigint NOT NULL
,row_overflow_fetch_in_pages bigint NOT NULL
,row_overflow_fetch_in_bytes bigint NOT NULL
,column_value_push_off_row_count bigint NOT NULL
,column_value_pull_in_row_count bigint NOT NULL
,row_lock_count bigint NOT NULL
,row_lock_wait_count bigint NOT NULL
,row_lock_wait_in_ms bigint NOT NULL
,page_lock_count bigint NOT NULL
,page_lock_wait_count bigint NOT NULL
,page_lock_wait_in_ms bigint NOT NULL
,index_lock_promotion_attempt_count bigint NOT NULL
,index_lock_promotion_count bigint NOT NULL
,page_latch_wait_count bigint NOT NULL
,page_latch_wait_in_ms bigint NOT NULL
,page_io_latch_wait_count bigint NOT NULL
,page_io_latch_wait_in_ms bigint NOT NULL
,now datetime default getdate())
else truncate table dbo.indexstats
go

```

add_column

```

create proc dbo.add_column (
    @add_stmt varchar(500) output,
    @find varchar(100)=NULL,
    @cols_containing varchar(500)=NULL,
    @col_stmt varchar(max))
as
--
-- This stored procedure is provided "AS IS" with no warranties, and
-- confers no rights.
-- Use of included script samples are subject to the terms specified at
-- http://www.microsoft.com/info/copyright.htm
--
-- T. Davidson
-- @add_stmt is the result passed back to the caller
-- @find is a keyword from @cols_containing
-- @cols_containing is the list of keywords to include in the report
-- @col_stmt is the statement that will be compared with @find.
--     If @col_stmt contains @find, include this statement.
--     set @add_stmt = @col_stmt

```

```
--
declare @length int, @strindex int, @EOS bit
if @cols_containing is NULL
begin
    select @add_stmt=@col_stmt
    return
end
select @add_stmt = '', @EOS = 0

while @add_stmt is not null and @EOS = 0
    @dbid=-1,
    select @strindex = charindex(',',@cols_containing)
    if @strindex = 0
        select @find = @cols_containing, @EOS = 1
    else
begin
        select @find = substring(@cols_containing,1,@strindex-1)
        select @cols_containing =
            substring(@cols_containing,
                @strindex+1,
                datalength(@cols_containing) - @strindex)
    end
    select @add_stmt=case
--when @cols_containing is NULL then NULL
        when charindex(@find,@col_stmt) > 0 then NULL
        else ''
    end
end
--- NULL indicates this statement is to be passed back through out parm
@add_stmt
if @add_stmt is NULL select @add_stmt=@col_stmt
go
```

Wait states

이 저장 프로시저들은 차단 분석에 사용할 수 있습니다.

track_waitstats_2005

```

CREATE proc [dbo].[track_waitstats_2005] (
    @num_samples int=10,
    @delay_interval int=1,
    @delay_type nvarchar(10)='minutes',
    @truncate_history nvarchar(1)='N',
    @clear_waitstats nvarchar(1)='Y')
as
--
-- This stored procedure is provided "AS IS" with no warranties, and
-- confers no rights.
-- Use of included script samples are subject to the terms specified at
-- http://www.microsoft.com/info/copyright.htm
--
-- T. Davidson
-- @num_samples is the number of times to capture waitstats, default is 10
-- times
-- default delay interval is 1 minute
-- delaynum is the delay interval - can be minutes or seconds
-- delaytype specifies whether the delay interval is minutes or seconds
-- create waitstats table if it doesn't exist, otherwise truncate
-- Revision: 4/19/05
--- (1) added object owner qualifier
--- (2) optional parameters to truncate history and clear waitstats
set nocount on
if not exists (select 1
    from sys.objects
    where object_id = object_id ( N'[dbo].[waitstats]') and
    OBJECTPROPERTY(object_id, N'IsUserTable') = 1)
    create table [dbo].[waitstats]
        ([wait_type] nvarchar(60) not null,
        [waiting_tasks_count] bigint not null,
        [wait_time_ms] bigint not null,
        [max_wait_time_ms] bigint not null,
        [signal_wait_time_ms] bigint not null,
        now datetime not null default getdate())

If lower(@truncate_history) not in (N'y',N'n')
    begin
        raiserror ('valid @truncate_history values are 'y' or
'N',16,1) with nowait
    end
If lower(@clear_waitstats) not in (N'y',N'n')
    begin
        raiserror ('valid @clear_waitstats values are 'y' or
'N',16,1) with nowait
    end
If lower(@truncate_history) = N'y'
    truncate table dbo.waitstats

If lower (@clear_waitstats) = N'y'
    -- clear out waitstats
    dbcc sqlperf ([sys.dm_os_wait_stats],clear) with no_infomsgs

declare @i int,

```



```

        @delay varchar(8),
        @dt varchar(3),
        @now datetime,
        @totalwait numeric(20,1),
        @endtime datetime,
        @begintime datetime,
        @hr int,
        @min int,
        @sec int

select @i = 1
select @dt = case lower(@delay_type)
    when N'minutes' then 'm'
    when N'minute' then 'm'
    when N'min' then 'm'
    when N'mi' then 'm'
    when N'n' then 'm'
    when N'm' then 'm'
    when N'seconds' then 's'
    when N'second' then 's'
    when N'sec' then 's'
    when N'ss' then 's'
    when N's' then 's'
    else @delay_type
end

if @dt not in ('s','m')
begin
    raiserror ('delay type must be either ''seconds'' or
''minutes'',16,1) with nowait
    return
end
if @dt = 's'
begin
    select @sec = @delay_interval % 60, @min = cast((@delay_interval / 60)
as int), @hr = cast((@min / 60) as int)
end
if @dt = 'm'
begin
    select @sec = 0, @min = @delay_interval % 60, @hr =
cast((@delay_interval / 60) as int)
end
select @delay= right('0'+ convert(varchar(2),@hr),2) + ':' +
    + right('0'+convert(varchar(2),@min),2) + ':' +
    + right('0'+convert(varchar(2),@sec),2)

if @hr > 23 or @min > 59 or @sec > 59
begin
    select 'delay interval and type: ' + convert
(varchar(10),@delay_interval) + ',' + @delay_type + ' converts to ' +
@delay
    raiserror ('hh:mm:ss delay time cannot > 23:59:59',16,1) with nowait
    return
end
while (@i <= @num_samples)
begin

```

```

select @now = getdate()
insert into [dbo].[waitstats] (
    [wait_type],
    [waiting_tasks_count],
    [wait_time_ms],
    [max_wait_time_ms],
    [signal_wait_time_ms],
    now)

select
    [wait_type],
    [waiting_tasks_count],
    [wait_time_ms],
    [max_wait_time_ms],
    [signal_wait_time_ms],
    @now
from sys.dm_os_wait_stats

insert into [dbo].[waitstats] (
    [wait_type],
    [waiting_tasks_count],
    [wait_time_ms],
    [max_wait_time_ms],
    [signal_wait_time_ms],
    now)

select
    'Total',
    sum([waiting_tasks_count]),
    sum([wait_time_ms]),
    0,
    sum([signal_wait_time_ms]),
    @now
from [dbo].[waitstats]
where now = @now

select @i = @i + 1
waitfor delay @delay
end
--- create waitstats report
execute dbo.get_waitstats_2005
go
exec dbo.track_waitstats @num_samples=6
    ,@delay_interval=30
    ,@delay_type='s'
    ,@truncate_history='y'
    ,@clear_waitstats='y'

```

get_waitstats_2005

```

CREATE proc [dbo].[get_waitstats_2005] (
    @report_format varchar(20)='all',
    @report_order varchar(20)='resource')
as
-- This stored procedure is provided "AS IS" with no warranties, and
-- confers no rights.
-- Use of included script samples are subject to the terms specified at
-- http://www.microsoft.com/info/copyright.htm
--

```

```
-- this proc will create waitstats report listing wait types by
-- percentage.
-- (1) total wait time is the sum of resource & signal waits,
--     @report_format='all' reports resource & signal
-- (2) Basics of execution model (simplified)
--     a. spid is running then needs unavailable resource, moves to
--        resource wait list at time T0
--     b. a signal indicates resource available, spid moves to
--        runnable queue at time T1
--     c. spid awaits running status until T2 as cpu works its way
--        through runnable queue in order of arrival
-- (3) resource wait time is the actual time waiting for the
--     resource to be available, T1-T0
-- (4) signal wait time is the time it takes from the point the
--     resource is available (T1)
--     to the point in which the process is running again at T2.
--     Thus, signal waits are T2-T1
-- (5) Key questions: Are Resource and Signal time significant?
--     a. Highest waits indicate the bottleneck you need to solve
--        for scalability
--     b. Generally if you have LOW% SIGNAL WAITS, the CPU is
--        handling the workload e.g. spids spend move through
--        runnable queue quickly
--     c. HIGH % SIGNAL WAITS indicates CPU can't keep up,
--        significant time for spids to move up the runnable queue
--        to reach running status
-- (6) This proc can be run when track_waitstats is executing
--
-- Revision 4/19/2005
-- (1) add computation for CPU Resource Waits = Sum(signal waits /
--     total waits)
-- (2) add @report_order parm to allow sorting by resource, signal
--     or total waits
--
set nocount on

declare @now datetime,
        @totalwait numeric(20,1),
        @totalsignalwait numeric(20,1),
        @totalresourcewait numeric(20,1),
        @endtime datetime,@begintime datetime,
        @hr int,
        @min int,
        @sec int

if not exists (select 1
               from sysobjects
               where id = object_id ( N'[dbo].[waitstats]') and
                     OBJECTPROPERTY(id, N'IsUserTable') = 1)
begin
    raiserror('Error [dbo].[waitstats] table does not exist',
              16, 1) with nowait
    return
end

if lower(@report_format) not in ('all','detail','simple')
```

```

begin
    raiserror ('@report_format must be either 'all',
              'detail', or 'simple'',16,1) with nowait
    return
end
if lower(@report_order) not in ('resource','signal','total')
begin
    raiserror ('@report_order must be either 'resource',
              'signal', or 'total'',16,1) with nowait
    return
end
if lower(@report_format) = 'simple' and lower(@report_order) <> 'total'
begin
    raiserror ('@report_format is simple so order defaults to
    'total'',
              16,1) with nowait
    select @report_order = 'total'
end

select
    @now=max(now),
    @begintime=min(now),
    @endtime=max(now)
from [dbo].[waitstats]
where [wait_type] = 'Total'

--- subtract waitfor, sleep, and resource_queue from Total
select @totalwait = sum([wait_time_ms]) + 1, @totalsignalwait =
sum([signal_wait_time_ms]) + 1
from waitstats
where [wait_type] not in (
    'CLR_SEMAPHORE',
    'LAZYWRITER_SLEEP',
    'RESOURCE_QUEUE',
    'SLEEP_TASK',
    'SLEEP_SYSTEMTASK',
    'Total' , 'WAITFOR',
    '***total***') and
    now = @now

select @totalresourcewait = 1 + @totalwait - @totalsignalwait

-- insert adjusted totals, rank by percentage descending
delete waitstats
where [wait_type] = '***total***' and
now = @now

insert into waitstats
select
    '***total***',
    0,@totalwait,
    0,
    @totalsignalwait,
    @now

```

```

select 'start time'=@begintime,'end time'=@endtime,
       'duration (hh:mm:ss:ms)'=convert(varchar(50),@endtime-
@begintime,14),
       'report format'=@report_format, 'report order'=@report_order

if lower(@report_format) in ('all','detail')
begin
----- format=detail, column order is resource, signal, total. order by
resource desc
    if lower(@report_order) = 'resource'
        select [wait_type],[waiting_tasks_count],
               'Resource wt (T1-T0)'=[wait_time_ms]-[signal_wait_time_ms],
               'res_wt_%'=cast (100*([wait_time_ms] -
                                   [signal_wait_time_ms]) /@totalresourcewait as
numeric(20,1)),
               'Signal wt (T2-T1)'=[signal_wait_time_ms],
               'sig_wt_%'=cast (100*[signal_wait_time_ms]/@totalsignalwait as
numeric(20,1)),
               'Total wt (T2-T0)'=[wait_time_ms],
               'wt_%'=cast (100*[wait_time_ms]/@totalwait as numeric(20,1))
        from waitstats
        where [wait_type] not in (
            'CLR_SEMAPHORE',
            'LAZYWRITER_SLEEP',
            'RESOURCE_QUEUE',
            'SLEEP_TASK',
            'SLEEP_SYSTEMTASK',
            'Total',
            'WAITFOR') and
            now = @now
        order by 'res_wt_%' desc

----- format=detail, column order signal, resource, total. order by signal
desc
    if lower(@report_order) = 'signal'
        select [wait_type],
               [waiting_tasks_count],
               'Signal wt (T2-T1)'=[signal_wait_time_ms],
               'sig_wt_%'=cast (100*[signal_wait_time_ms]/@totalsignalwait
as numeric(20,1)),
               'Resource wt (T1-T0)'=[wait_time_ms]-[signal_wait_time_ms],
               'res_wt_%'=cast (100*([wait_time_ms] -
                                   [signal_wait_time_ms]) /@totalresourcewait as
numeric(20,1)),
               'Total wt (T2-T0)'=[wait_time_ms],
               'wt_%'=cast (100*[wait_time_ms]/@totalwait as
numeric(20,1))
        from waitstats
        where [wait_type] not in (
            'CLR_SEMAPHORE',
            'LAZYWRITER_SLEEP',
            'RESOURCE_QUEUE',
            'SLEEP_TASK',
            'SLEEP_SYSTEMTASK',
            'Total',
            'WAITFOR') and

```

```

        now = @now
        order by 'sig_wt_%' desc

----- format=detail, column order total, resource, signal. order by total
desc
    if lower(@report_order) = 'total'
        select
            [wait_type],
            [waiting_tasks_count],
            'Total wt (T2-T0)'=[wait_time_ms],
            'wt_%'=cast (100*[wait_time_ms]/@totalwait as numeric(20,1)),
            'Resource wt (T1-T0)'=[wait_time_ms]-[signal_wait_time_ms],
            'res_wt_%'=cast (100*([wait_time_ms] -
                [signal_wait_time_ms]) /@totalresourcewait as numeric(20,1)),
            'Signal wt (T2-T1)'=[signal_wait_time_ms],
            'sig_wt_%'=cast (100*[signal_wait_time_ms]/@totalsignalwait as
numeric(20,1))
        from waitstats
        where [wait_type] not in (
            'CLR_SEMAPHORE',
            'LAZYWRITER_SLEEP',
            'RESOURCE_QUEUE',
            'SLEEP_TASK',
            'SLEEP_SYSTEMTASK',
            'Total',
            'WAITFOR') and
            now = @now
        order by 'wt_%' desc
    end
else
---- simple format, total waits only
    select
        [wait_type],
        [wait_time_ms],
        percentage=cast (100*[wait_time_ms]/@totalwait as numeric(20,1))
    from waitstats
    where [wait_type] not in (
        'CLR_SEMAPHORE',
        'LAZYWRITER_SLEEP',
        'RESOURCE_QUEUE',
        'SLEEP_TASK',
        'SLEEP_SYSTEMTASK',
        'Total',
        'WAITFOR') and
        now = @now
    order by percentage desc

---- compute cpu resource waits
select
    'total waits'=[wait_time_ms],
    'total signal=CPU waits'=[signal_wait_time_ms],
    'CPU resource waits % = signal waits / total waits'=
        cast (100*[signal_wait_time_ms]/[wait_time_ms] as
numeric(20,1)),
    now
from [dbo].[waitstats]

```

```
where [wait_type] = '***total***'  
order by now  
go
```

```
declare @now datetime  
select @now = getdate()  
select getdate()
```