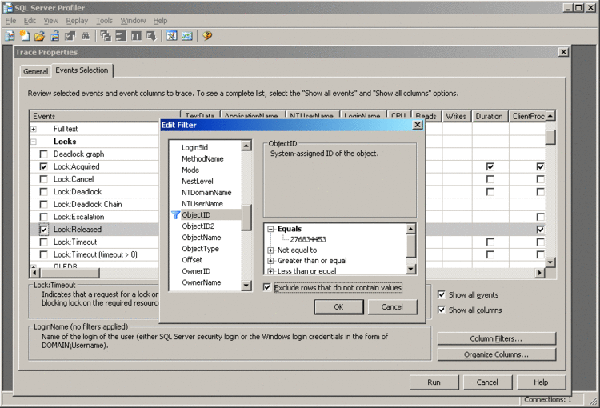
SQL Server 2008 확장 이벤트를 사용한 고급 문제 해결

전 세계의 SQL Server DBA에게 영원히 사라지지 않을 듯이 보이는 한 가지 문제는 바로 문제 해결입니다. 그리고 대부분의 문제 해결은 일종의 성능 문제를 찾기 위해 이루어집니다. 극도의 신중을 기해 설계하고 테스트한 응용 프로그램 시스템이라도 시간이 지남에 따라 변경되는 부분이 생기고, 이는 심각한 성능 문제로 이어질 수 있습니다.   
  
예를 들어 작업 부하가 변경되고(예: 동시 사용자의 수, 쿼리의 수, 새로운 월말 보고서의 수), 처리 데이터의 양이 증가하고, 시스템이 실행되는 하드웨어가 바뀌고(예: 프로세서 코어의 수, 서버 메모리 용량, I/O 하위 시스템 용량), 새로운 동시 작업 부하가 생길 수 있습니다(예: 트랜잭션 복제, 데이터베이스 미러링, 변경 데이터 캡처).  
  
그러나 이런 사안들 외에 다른 문제도 발생합니다. 응용 프로그램 시스템을 디자인하고 테스트하는 경우 디자인과 관련된 예상하지 못한 문제가 빈번하게 발견되며, 그러면 이에 대한 문제 해결 작업이 필요하게 됩니다. 응용 프로그램 수명 주기상 언제 문제가 발견되는지에 관계없이 반드시 문제 해결을 통해 그 원인과 해결책을 찾아야 합니다.   
  
복잡한 응용 프로그램 시스템에는 분석이 필요한 많은 수의 하드웨어 및 소프트웨어 구성 요소가 포함되지만 이번에 살펴보고자 하는 요소는 SQL Server입니다. 다양한 문제 해결 방법론에 대한 이야기는 접어두고(이 기사의 범위를 훨씬 벗어나는 주제임), SQL Server에서 문제 해결을 위해 필요한 도구는 무엇인지 살펴보겠습니다.

SQL Server 2005의 문제 해결

최근 몇 차례의 SQL Server 릴리스를 거치는 동안 성능 문제를 해결하기 위한 도구의 선택 폭이 상당히 넓어졌습니다. 예전부터 SQL Server에는 데이터베이스 엔진의 여러 부분에서 벌어지는 일을 볼 수 있는 다양한 DBCC(Database Console Command) 명령이 제공되었습니다. 또한 SQL Server Profiler도 있고 기본 SQL Trace 메커니즘을 프로그래밍 방식으로 사용하는 방법도 있습니다.

SQL Server는 문제 해결을 위한 방안을 지속적으로 제공해왔지만 이러한 옵션에는 몇 가지 문제가 있습니다. DBCC 출력 전처리의 경우 결과를 임시 테이블로 넘기기 전까지는 아무것도 할 수 없으므로 불편합니다. 또한 SQL Trace/Profiler 실행은 잘못 구성할 경우 성능 하락을 초래할 수 있습니다(예: 작업이 많은 시스템에서 모든 Lock: Acquired 및 Lock: Released 이벤트를 추적하면서 이벤트의 DatabaseId아 ObjectId 열에 대한 필터링을 잊은 경우). 그림 1의 스크린샷은 새 추적에 대한 필터를 구성하는 데 사용되는 대화 상자를 보여 줍니다.

 **그림 1 SQL Server 2008 Profiler에서 필터 구성**

SQL Server 2005에는 데이터베이스에서 정보를 얻기 위한 방편으로 동적 관리 뷰 및 함수(통칭해서 DMV)가 추가되었습니다. DMV는 일부 DBCC 명령, 시스템 테이블 및 저장 프로시저를 대체하면서 엔진 활동의 여러 새로운 영역을 드러냈습니다. 이러한 DMV는 조합이 가능한 강력한 명령으로, DMV 결과를 필터링 및 전처리하는 복합적인 T-SQL 문에서 사용할 수 있습니다.

예를 들어 그림 2의 코드는 데이터베이스 인덱스에서 리프 수준의 조각화 및 페이지 밀도만 반환하며(두 가지 모두 반올림함), 이때 조각화 수준에 대해 필터를 적용합니다. 이 작업은 기존의 DBCC SHOWCONTIG 명령으로는 수행하기 어렵습니다. DMV에 대한 자세한 내용은 "동적 관리 뷰 및 함수(Transact-SQL)"를 참조하십시오. 또한 SQL Server 2005에는 DDL(데이터 정의 언어) 트리거 및 이벤트 알림을 비롯하여 문제 해결에 사용할 수 있는 다른 여러 기능도 추가되었습니다.

**그림 2 강력한 결과를 얻을 수 있는 DMV 사용**

|  |
| --- |
| SELECT  OBJECT\_NAME (ips.[object\_id]) AS 'Object Name',  si.name AS 'Index Name',  ROUND (ips.avg\_fragmentation\_in\_percent, 2) AS 'Fragmentation',  ips.page\_count AS 'Pages',  ROUND (ips.avg\_page\_space\_used\_in\_percent, 2) AS 'Page Density'  FROM sys.dm\_db\_index\_physical\_stats (  DB\_ID ('SQLskillsDB'), NULL, NULL, NULL, 'DETAILED') ips  CROSS APPLY sys.indexes si  WHERE  si.object\_id = ips.object\_id  AND si.index\_id = ips.index\_id  AND ips.index\_level = 0 -- only the leaf level  AND ips.avg\_fragmentation\_in\_percent > 10; -- filter on fragmentation  GO |

Microsoft 내의 여러 팀에서도 SQLdiag 유틸리티, SQL Server용 RML 유틸리티, SQL Server 2005 Performance Dashboard Reports 및 DMVStats와 같은 유용한 성능 문제 해결 도구를 제공했습니다. SQL Server 2005를 위한 ETW(Windows용 이벤트 추적) 공급자도 있는데, 이를 사용하면 SQL Trace 이벤트를 Windows의 다른 부분에서 발생한 이벤트와 통합할 수 있습니다.

SQL Server 2005에서 DBA의 데이터베이스 엔진 문제 해결 능력은 크게 강화되었지만 여전히 DBA가 효과적으로 해결하기가 거의 불가능한 시나리오가 많았습니다. 그중 자주 언급되는 예는 일부 쿼리가 CPU 리소스를 과도하게 사용하는 상황에서 DMV로는 문제의 임시 쿼리가 무엇인지 찾아내기 위한 충분한 정보를 얻을 수 없는 경우입니다. 그러나 SQL Server 2005와 달리 SQL Server 2008은 SQL Server 확장 이벤트라는 새로운 기능을 통해 이러한 한계를 극복할 수 있습니다.

확장 이벤트

확장 이벤트 시스템의 기능은 SQL Server가 제공한 이전의 이벤트 추적 및 문제 해결 메커니즘보다 훨씬 더 강력합니다. 필자의 의견으로는 확장 이벤트 시스템의 핵심적인 특징은 다음과 같습니다.

* 이벤트는 동기적으로 발생하지만 동기적으로도, 비동기적으로도 처리할 수 있습니다.
* 어떤 대상이든 모든 이벤트를 소비할 수 있고 이벤트와 동작을 자유롭게 짝지을 수 있으므로 심층적인 모니터링 시스템이 가능합니다.
* "똑똑한" 조건자를 통해 부울 논리를 사용하여 복합적인 규칙을 작성할 수 있습니다.
* Transact-SQL을 사용하여 확장 이벤트 세션을 완전하게 제어할 수 있습니다.
* 성능에 민감한 코드를 성능에 대한 영향 없이 모니터링할 수 있습니다.

계속 진행하기 전에 새로운 용어 몇 가지를 정의하겠습니다.   
이벤트 이벤트는 코드에 정의되는 지점입니다. 예를 들어 T-SQL 문이 실행을 멈추는 지점, 또는 잠금 획득이 완료되는 지점입니다. 각 이벤트는 정의된 페이로드(이벤트에서 반환하는 열 집합)를 가지며, ETW 모델(모든 이벤트가 채널 및 키워드를 페이로드의 일부로 반환함)을 사용하여 정의되므로 ETW와의 통합이 가능합니다. SQL Server 2008에는 일단 254개의 이벤트가 정의되어 있고 이후 더 추가될 예정입니다.   
다음 코드를 사용하면 정의된 이벤트 목록을 볼 수 있습니다.

|  |
| --- |
| SELECT xp.[name], xo.\*  FROM sys.dm\_xe\_objects xo, sys.dm\_xe\_packages xp  WHERE xp.[guid] = xo.[package\_guid]  AND xo.[object\_type] = 'event'  ORDER BY xp.[name]; |

다음 코드로는 특정 이벤트에 대한 페이로드를 찾을 수 있습니다.

|  |
| --- |
| SELECT \* FROM sys.dm\_xe\_object\_columns  WHERE [object\_name] = 'sql\_statement\_completed';  GO |

확장 이벤트 시스템에는 모든 이벤트, 대상 등을 기술할 수 있는 포괄적인 정보 제공 DMV 집합이 있습니다. 자세한 내용은 "SQL Server 확장 이벤트 동적 관리 뷰"를 참조하십시오.  
조건자 조건자는 이벤트가 소비되기 전에 논리적 규칙 집합을 사용하여 이벤트를 필터링하는 데 사용되는 메서드입니다. 조건자는 이벤트 페이로드에 반환된 열 중 하나가 특정 값인지 확인하는 것과 같이 단순할 수도 있습니다(예: 개체 ID별로 잠금 획득 이벤트 필터링).   
또한 조건자는 몇 가지 고급 기능도 제공합니다. 예를 들어 세션 중에 특정 이벤트가 발생한 횟수를 계산하거나, 이벤트가 발생한 후에만 이 이벤트를 소비하도록 허용하거나, 조건자 스스로를 동적으로 업데이트하여 비슷한 데이터가 포함된 이벤트의 소비를 금지할 수 있습니다.

조건자는 신속한 판단을 위해 부울 논리를 사용하여 작성할 수 있습니다. 이로써 이벤트가 소비될 것인지 여부를 판단하기 전에 수행되는 동기 처리를 최소화할 수 있습니다.  
동작 동작은 이벤트가 소비되기 전에 동기적으로 수행되는 명령 집합입니다. 모든 동작은 모든 이벤트와 자유롭게 연결이 가능합니다. 동작은 일반적으로 이벤트 페이로드에 추가할 데이터를 수집하거나(예: T-SQL 스택 또는 쿼리 실행 계획) 이벤트 페이로드에 추가되는 몇 가지 계산 작업을 수행합니다.

동작은 리소스를 많이 소모할 수 있으므로 이벤트에 대한 동작은 조건자가 계산된 이후에만 수행됩니다(이후 이벤트가 소비되지 않을 것으로 판단된다면 동기적으로 동작을 수행하는 의미가 없음). 다음 코드를 사용하면 미리 정의된 동작 목록을 볼 수 있습니다.

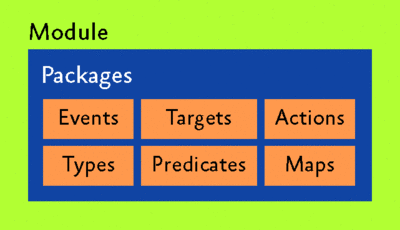
|  |
| --- |
| SELECT xp.[name], xo.\*  FROM sys.dm\_xe\_objects xo, sys.dm\_xe\_packages xp  WHERE xp.[guid] = xo.[package\_guid]  AND xo.[object\_type] = 'action'  ORDER BY xp.[name]; |

대상 대상은 단순히 이벤트를 소비하기 위한 방법을 제공하며, 모든 대상은 모든 이벤트를 소비할 수 있습니다. 또는 대상이 할 일이 없는 경우에는 최소한 이벤트를 폐기할 수 있습니다(예: 감사 대상이 비감사 이벤트를 받은 경우). 대상은 동기적으로(예: 이벤트를 발생시킨 코드가 이벤트가 소비되기를 기다림) 또는 비동기적으로 이벤트를 소비할 수 있습니다.

대상은 이벤트 파일, 링 버퍼와 같은 단순한 소비자부터 이벤트 쌍 구성을 수행할 수 있는 더 복잡한 소비자에 이르기까지 다양합니다. 다음 코드를 사용하면 사용 가능한 대상 목록을 볼 수 있습니다.

|  |
| --- |
| SELECT xp.[name], xo.\*  FROM sys.dm\_xe\_objects xo, sys.dm\_xe\_packages xp  WHERE xp.[guid] = xo.[package\_guid]  AND xo.[object\_type] = 'target'  ORDER BY xp.[name]; |

대상에 대한 자세한 내용은 "SQL Server 확장 이벤트 대상"을 참조하십시오.  
패키지 패키지는 확장 이벤트 개체(이벤트, 동작, 대상 등)를 정의하는 컨테이너입니다. 패키지는 패키지가 기술하는 모듈(실행 파일 또는 DLL) 내에 포함됩니다(그림 3 참조).

 **그림 3 모듈, 패키지 및 확장 이벤트 개체 간의 관계**

패키지가 확장 이벤트 엔진에 등록되면 패키지에 의해 정의되는 모든 개체는 사용 가능한 상태가 됩니다. 패키지에 대한 자세한 내용과 확장 이벤트 용어의 전체 목록은 "SQL Server 확장 이벤트 패키지"를 참조하십시오.

세션 세션은 처리를 위해 확장 이벤트 개체를 서로 연결하는 방법입니다(대상에 의해 소비될 동작이 있는 이벤트). 세션은 모든 등록된 패키지의 개체를 연결할 수 있으며 여러 세션(수에 제한 없음)에서 같은 이벤트, 동작 등을 사용할 수 있습니다. 다음 코드를 사용하면 정의된 확장 이벤트 세션을 볼 수 있습니다.

|  |
| --- |
| SELECT \* FROM sys.dm\_xe\_sessions;  Go |

세션은 T-SQL 명령을 사용하여 작성, 삭제, 변경, 중지 및 시작합니다. 따라서 유연성이 높습니다. 세션 자체에 의해 캡처되고 있는 데이터에 대한 프로그래밍 방식 분석을 기반으로 해당 세션을 동적으로 변경할 수도 있습니다. 세션에 대한 자세한 내용은 "SQL Server 확장 이벤트 세션"을 참조하십시오.

성능 고려 사항

CREATE EVENT SESSION을 사용하여 확장 이벤트 세션을 구성할 때는 올바르게 구성하도록 주의를 기울여야 하는 일부 설정이 있습니다. 이러한 설정은 잘못 구성될 경우 성능에 부정적인 영향을 미치기 때문입니다. 우선 이벤트를 동기적으로 소비할지 비동기적으로 소비할지 결정해야 합니다. 짐작할 수 있듯이 동기 대상은 비동기 대상에 비해 모니터링되는 코드의 성능에 더 큰 영향을 미칩니다.

위에서 설명한 바와 같이 이벤트가 동기적으로 소비되는 경우 해당 이벤트를 발생시킨 코드는 이벤트가 소비될 때까지 기다려야 합니다. 이벤트 소비 프로세스가 복잡한 경우 이는 코드의 속도를 저하시킵니다.

예를 들어 초당 수천 개의 작은 트랜잭션을 서비스하는 바쁜 시스템에서 쿼리 계획을 캡처하는 동작이 있는 sql\_statement\_completed 이벤트를 동기적으로 소비할 경우 이는 성능에 부정적인 영향을 미칠 가능성이 매우 높습니다. 또한 조건자는 항상 동기적으로 실행되므로 성능에 민감한 코드에 의해 발생하는 이벤트에 대해서는 지나치게 복잡한 조건자를 만들지 않도록 주의해야 합니다.

한편 이벤트를 동기적으로 소비해야만 하는 경우도 있습니다. 특정 이벤트의 발생 횟수를 계산하기 위한 가장 쉬운 방법은 synchronous\_event\_counter 대상을 사용하는 것입니다.

두 번째로 고려해야 할 부분은 비동기 대상을 사용하기로 결정한 경우 이벤트 버퍼링을 구성하는 방법입니다. 이벤트 버퍼링에서 기본적으로 사용할 수 있는 메모리는 4MB입니다. 이벤트 발생과 그 이후 대상에 의한 소비 사이의 발송 대기 시간은 기본적으로 30초입니다. 예를 들어 10초마다 이벤트 통계를 작성하려면 이 대기 시간을 조정해야 합니다.

이벤트 버퍼렁 설정에 따라서 이벤트를 버퍼링하는 데 사용되는 메모리의 분할 방식이 결정됩니다. 기본적으로는 전체 인스턴스에 대한 버퍼 집합을 만듭니다. SMP(대칭적 다중 프로세서) 및 NUMA(Non-uniform Memory Access) 시스템에서 이 방식을 사용하면 프로세서가 메모리에 대한 액세스를 기다려야 하므로 성능 문제가 발생할 수 있습니다.

세 번째 고려 사항은 이벤트 손실을 처리하는 방법입니다. 확장 이벤트 세션을 정의할 때 이벤트의 "손실" 가능 여부를 지정할 수 있습니다. 이벤트가 손실된다는 것은 이벤트를 버퍼링할 메모리가 부족할 경우 해당 이벤트가 삭제됨을 의미합니다. 기본적으로는 하나의 이벤트 삭제가 허용되지만 전체 이벤트 버퍼가 손실되도록 허용하거나(이벤트 버퍼가 차는 속도가 매우 빠른 세션의 경우) 이벤트가 손실되지 않도록 지정할 수도 있습니다.

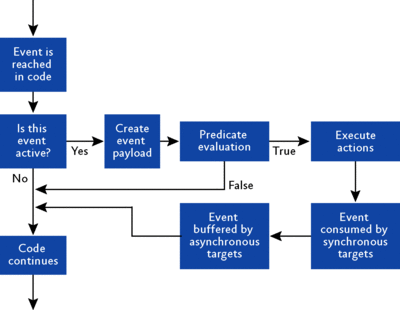
마지막에 언급한 방법은 이벤트를 발생시킨 코드가 이벤트를 저장할 버퍼 메모리가 생길 때까지 기다려야 하기 때문에 상당히 주의하여 사용해야 합니다. 이 옵션은 거의 항상 성능에 부정적으로 작용합니다. 의도하지 않게 이 옵션을 활성화하더라도 서버의 응답성은 한동안 유지되므로 설정을 다시 비활성화할 수 있는 시간은 충분합니다.

일반적으로 이러한 설정은 함께 고려해야 합니다. 여기에는 신중하게 생각하지 않으면 성능 문제에 직면할 수 있다는 점을 제외하고는 일반화된 최상의 방법은 없습니다. 이러한 설정에 대한 자세한 내용은 "CREATE EVENT SESSION(Transact-SQL)"을 참조하십시오.

이벤트의 일생

확장 이벤트 세션이 정의되어 시작되면 모니터링되는 코드에 이벤트가 발생할 때까지 처리가 정상적으로 지속됩니다. 확장 이벤트 시스템이 따르는 단계는 그림 4에 대략적으로 나와 있습니다. 단계는 다음과 같습니다.

1. 이벤트를 모니터링하는 확장 이벤트 세션이 있는지 여부에 대한 확인이 수행됩니다. 세션이 없으면 이벤트가 포함된 코드로 제어가 돌아가고 처리가 계속됩니다.
2. 이벤트 페이로드가 결정되고 필요한 모든 정보가 메모리에 수집됩니다. 즉, 이벤트 페이로드가 생성됩니다.
3. 이벤트에 대해 정의된 조건자가 있는 경우 실행됩니다. 이 시점에서 조건자의 결과가 이벤트를 소비하면 안 되는 것으로 나올 수 있습니다. 이 경우 이벤트가 포함된 코드로 제어가 돌아가고 처리가 계속됩니다.
4. 이제 시스템은 이벤트가 소비될 것임을 알고 이 이벤트와 연결된 동작을 실행합니다. 이벤트는 완전한 페이로드를 가지며 소비될 준비가 됩니다.
5. 동기 대상(있는 경우)에 이벤트가 서비스됩니다.
6. 비동기 대상이 있는 경우 이벤트는 나중에 처리되도록 버퍼링됩니다.
7. 이벤트가 포함된 코드로 제어가 돌아가고 처리가 계속됩니다.

  
**그림 4 확장 이벤트의 일생**

앞서 언급했듯이 이벤트 세션을 만들 때는 동기 동작 또는 비동기 대상을 위한 버퍼링이 모니터링되는 코드의 성능에 영향을 미치지 않도록 주의를 기울여야 합니다.

확장 이벤트 사용

SQL Server 2008 온라인 설명서에는 확장 이벤트 사용에 대한 두 가지 예가 포함되어 있습니다. 하나는 "방법: 잠금을 보유한 쿼리 파악"이고 다른 하나는 "방법: 가장 많은 잠금이 발생한 개체 찾기"입니다.

지금부터는 확장 이벤트 세션을 설정하고 결과를 분석하는 예를 살펴보겠습니다. 필자가 2007년 후반에 확장 이벤트를 사용하기 시작하면서 알게 된 사실은 간단한 세션을 구성하기는 무척 쉽지만(단순한 T-SQL DDL 문 사용) 그 결과를 분석하는 일은 결코 간단하지 않다는 것입니다.

처음에 필자는 결과가 XML로 제공된다는 점에 놀랐는데, 하나의 세션에서 수집 가능한 이벤트와 동작의 가능한 조합이 까마득하게 많기 때문에 이러한 광범위한 스키마를 저장하기 위해서는 XML 외의 다른 적절한 방법은 없다는 사실을 곧 깨달았습니다.

필자는 오랜 시간 동안 SQL Server 저장소 엔진 팀의 개발자를 지냈고 지금은 스스로를 C와 C++, 어셈블리에 매우 능숙한 프로그래머라고 생각하지만, XML 데이터에서 프로그래밍 방식으로 이벤트 페이로드 필드를 추출하는 데 필요한 코드를 고안하는 데 몇 시간이나 걸렸습니다. 이 말은 확장 이벤트를 사용하지 말라는 의미가 아닙니다. 단지 XML 데이터를 사용한 작업에 익숙하지 않다면 결과 분석에 어려움이 있을 것이란 사실을 말하고 싶을 뿐입니다.

필자의 시나리오는 다음과 같습니다. 필자는 SQL Server 2008의 리소스 관리자 기능을 사용하여 회사의 다양한 그룹을 프로덕션 서버 중의 하나에 샌드박스하는 DBA입니다. 두 개의 리소스 관리자 리소스 풀(Development와 Marketing)을 만들어 이 서버를 사용하는 각 팀을 나타내도록 했습니다. 리소스 관리자를 사용하면 각 풀의 CPU 및 쿼리 실행 메모리 사용량은 제한할 수 있지만 이들이 사용하는 I/O 리소스의 양은 제한할 수 없습니다. 그래서 서버의 I/O 사용에 대해 각 팀에게 비용을 청구함으로써 새로운 SAN(저장 영역 네트워크)로의 업그레이드 비용 상환에 도움이 되는 비용 부과 메커니즘을 마련하고자 합니다.

필자는 I/O 정보 캡처를 트리거하는 가장 쉬운 시점은 T-SQL 문이 완료될 때라고 생각하며, package0 패키지에 sql\_statement\_completed라는 이벤트가 있음을 알고 있습니다. 그렇다면 이벤트 페이로드에는 어떤 데이터가 수집될까요?  
다음 코드를 실행하면 읽기와 쓰기를 포함하는 모든 데이터의 목록을 얻을 수 있습니다.

|  |
| --- |
| SELECT [name] FROM sys.dm\_xe\_object\_columns  WHERE [object\_name] = 'sql\_statement\_completed';  GO |

이러한 데이터가 물리적인 읽기와 쓰기(데이터 읽기와 쓰기가 버퍼 풀의 메모리에서만이 아니라 디스크에서 수행되는 경우)라고는 생각하지 않지만 이를 통해 각 팀에서 사용하는 I/O 리소스의 비율을 파악할 수 있을 것입니다.

이제 어느 팀이 특정 T-SQL 문을 실행했는지 파악해야 하므로 이를 알려 줄 동작이 필요합니다. 이 코드를 실행하면 이벤트가 발생할 때 수행할 수 있는 모든 동작의 목록을 얻을 수 있습니다. 여기에는 sqlserver 패키지의 session\_resource\_pool\_id를 수집하는 동작이 포함됩니다.

|  |
| --- |
| SELECT xp.[name], xo.\*  FROM sys.dm\_xe\_objects xo, sys.dm\_xe\_packages xp  WHERE xp.[guid] = xo.[package\_guid]  AND xo.[object\_type] = 'action'  ORDER BY xp.[name]; |

리소스 관리자에 대해 정의한 리소스 풀의 목록을 얻고, 이를 필자의 확장 이벤트 세션에 의해 수집된 ID와 연관지을 수 있습니다. 이제 세션을 정의할 준비가 되었습니다. 이 코드를 실행하면 코드는 먼저 동일한 이름을 가진 이벤트 세션이 있는지 여부부터 확인합니다. 동일한 이름을 가진 세션을 발견하면 해당 세션을 삭제합니다. 코드는 다음과 같습니다.

|  |
| --- |
| IF EXISTS (  SELECT \* FROM sys.server\_event\_sessions  WHERE name = 'MonitorIO')  DROP EVENT SESSION MonitorIO ON SERVER;  GO  CREATE EVENT SESSION MonitorIO ON SERVER  ADD EVENT sqlserver.sql\_statement\_completed  (ACTION (sqlserver.session\_resource\_pool\_id))  ADD TARGET package0.ring\_buffer;  GO |

그런 다음 하나의 이벤트(sql\_statement\_completed)를 가진 새 세션을 만듭니다. 이 세션은 session\_resource\_pool\_id 동작도 수행하면서 필자가 아직 프로토타입 단계에 있는 동안 모든 항목을 링 버퍼에 로깅합니다. (프로덕션에서는 아마 비동기 파일 대상을 사용하는 방법을 선택하겠지요.)   
세션을 시작하려면 다음 코드를 실행해야 합니다.

|  |
| --- |
| ALTER EVENT SESSION MonitorIO ON SERVER  STATE = START;  GO |

이제 세션이 실행됩니다.   
마케팅 및 개발 팀의 활동을 몇 가지 가져오고 나면 세션 결과를 분석할 준비가 됩니다. 다음 코드는 링 버퍼에서 데이터를 추출합니다.

|  |
| --- |
| SELECT CAST(xest.target\_data AS XML) StatementData  FROM sys.dm\_xe\_session\_targets xest  JOIN sys.dm\_xe\_sessions xes ON  xes.address = xest.event\_session\_address  WHERE xest.target\_name = 'ring\_buffer'  AND xes.name = 'MonitorIO';  GO |

그런데 이 코드는 데이터를 하나의 큰 XML 값으로 추출합니다. 더 세분화하려면 그림 5의 코드를 사용하면 됩니다.

**그림 5 XML 데이터 세분화**

|  |
| --- |
| SELECT  Data2.Results.value ('(data/.)[6]', 'bigint') AS Reads,  Data2.Results.value ('(data/.)[7]', 'bigint') AS Writes,  Data2.Results.value ('(action/.)[1]', 'int') AS ResourcePoolID  FROM  (SELECT CAST(xest.target\_data AS XML) StatementData  FROM sys.dm\_xe\_session\_targets xest  JOIN sys.dm\_xe\_sessions xes ON  xes.address = xest.event\_session\_address  WHERE xest.target\_name = 'ring\_buffer'  AND xes.name = 'MonitorIO') Statements  CROSS APPLY StatementData.nodes ('//RingBufferTarget/event') AS Data2 (Results);  GO |

이 코드는 제대로 동작은 하지만 캡처된 각 이벤트에 대해 한 줄의 출력을 생성합니다. 이 형식은 별로 보기 좋지도 않고, 필자는 집계된 결과를 얻고자 하므로 그림 6에서와 같이 파생 테이블을 사용하기로 했습니다.

**그림 6 집계된 출력 얻기**

|  |
| --- |
| SELECT DT.ResourcePoolID,  SUM (DT.Reads) as TotalReads,  SUM (DT.Writes) AS TotalWrites  FROM  (SELECT  Data2.Results.value ('(data/.)[6]', 'bigint') AS Reads,  Data2.Results.value ('(data/.)[7]', 'bigint') AS Writes,  Data2.Results.value ('(action/.)[1]', 'int') AS ResourcePoolID  FROM  (SELECT CAST(xest.target\_data AS XML) StatementData  FROM sys.dm\_xe\_session\_targets xest  JOIN sys.dm\_xe\_sessions xes ON  xes.address = xest.event\_session\_address  WHERE xest.target\_name = 'ring\_buffer'  AND xes.name = 'MonitorIO') Statements  CROSS APPLY StatementData.nodes ('//RingBufferTarget/event') AS Data2 (Results)) AS DT  WHERE DT.ResourcePoolID > 255 –- only show user-defined resource pools  GROUP BY DT.ResourcePoolID;  GO |

휴, 이제 끝입니다. 물론 약간 복잡한 코드가 있긴 하지만 잘 동작합니다. 이제 원했던 결과를 얻었습니다. 그림 7에서 필자의 테스트 데이터에 대한 이 쿼리의 출력을 볼 수 있습니다.

**그림 7 쿼리 출력**

|  |  |  |
| --- | --- | --- |
| **ResourcePoolID z** | **TotalReads** | **TotalWrites** |
| 256 | 3831 | 244 |
| 257 | 5708155 | 1818 |

리소스 풀 256은 마케팅 팀, 257은 개발 팀에 해당합니다. 따라서 이 수치를 통해 회사에서 이 두 팀의 데이터베이스 활동 규모가 얼마나 되는지 파악할 수 있습니다. 확장 이벤트를 사용하지 않았다면 이러한 결과를 쉽게 생성하지 못했을 것입니다.

마지막으로, 다음 코드를 사용하여 세션을 중지해야 합니다.

|  |
| --- |
| ALTER EVENT SESSION MonitorIO ON SERVER  STATE = STOP;  GO |

이 예에 나오는 각 단계의 출력과 관련하여 필자가 이야기하는 내용에 대해 더 자세히 알아보려면 이 기사와 함께 제공되는 스크린 캐스트를 참조하십시오. 스크린 캐스트는 technetmagazine.com/video에서 볼 수 있습니다.

system\_health 확장 이벤트 세션

SQL Server 2008에는 기본적으로 실행되도록 설정된 미리 정의된 세션이 포함되어 있는데, 바로 system\_health 세션입니다. 제품 지원 팀의 아이디어인 이 세션은 이 팀이 고객 시스템을 디버깅하는 데 일반적으로 사용하는 정보를 추적합니다(예: 교착 상태 또는 심각한 오류가 발생하는 시점). 이 세션은 SQL Server 2008 인스턴스 설치 프로세스의 일부로 만들어져 시작되며 링 버퍼의 이벤트를 추적하므로 메모리를 그다지 많이 소모하지는 않습니다.

다음 코드를 사용하면 링 버퍼에 포함된 내용을 볼 수 있습니다.

|  |
| --- |
| SELECT CAST (xest.target\_data AS XML)  FROM sys.dm\_xe\_session\_targets xest  JOIN sys.dm\_xe\_sessions xes ON  xes.address = xest.event\_session\_address  WHERE xes.name = 'system\_health';  GO |

Microsoft PSS SQL 지원 블로그에는 이 세션이 추적하는 항목에 대한 더 자세한 정보가 나와 있습니다.

결론

필자가 들은 바로는 SQL Server 팀은 앞으로 sqlserver.exe에 더 많은 이벤트를 추가할 계획입니다. 사실 이러한 이벤트의 수는 2007년 2월 CTP(Community Technology Preview) 릴리스에서 165개였지만 RTM(Release To Manufacturing) 릴리스에서는 254개로 늘어났습니다.

살펴보면 흥미로운 이벤트가 몇 가지 있는데, 변경 데이터 캡처(TechNet Magazine 2008년 11월호에 실린 필자의 "엔터프라이즈 데이터베이스의 변경 내용 추적" 기사에서 다룸), 데이터 압축 및 인덱스 페이지 분할을 위한 이벤트가 여기에 속합니다. 인덱스 페이지 분할은 모든 인덱스에 걸쳐 sys.dm\_db\_index\_physical\_stats DMV를 주기적으로 실행하지 않고도 성능을 갉아먹는 조각화를 유발하는 인덱스를 찾아낼 수 있는 유망한 방법으로 보입니다.

전체적으로 새로운 확장 이벤트 시스템은 이전에는 불가능했던 몇 가지 매우 정교한 모니터링을 가능하게 합니다. 데이터를 얻기 위해 필요한 XML 구문 분석을 이해하기 위해서는 약간의 공부가 필요하지만 이 새로운 시스템의 이점은 새 코딩 구문을 익혀야 한다는 어려움을 기꺼이 감수할 가치가 있습니다.

이 기사에서 제공한 확장 이벤트의 개요가 여러분의 흥미를 유도하고 확장 이벤트로 어떤 일을 할 수 있는지에 대한 설명이 되었기를 바랍니다. 문제 해결은 즐거운 일입니다! 언제나 그렇듯이 의견이나 질문은 Paul@SQLskills.com으로 보내 주십시오.

필자소개

Paul S. Randal은 SQLskills.com의 관리 이사 겸 SQL Server MVP입니다. 1999년부터 2007년까지 Microsoft의 SQL Server 저장소 엔진 팀에서 근무한 Paul은 DBCC CHECKDB/repair for SQL Server 2005를 저술했으며 SQL Server 2008 개발 과정에서 핵심 저장소 엔진 부분을 담당했습니다. Paul은 재해 복구, 고가용성 및 데이터베이스 유지 관리 분야의 전문가이며 전 세계 각종 컨퍼런스에서도 꾸준히 의견을 발표하고 있습니다. 블로그 주소는 SQLskills.com/blogs/paul입니다.

[**출처 : 한국마이크로소프트**](http://technet.microsoft.com/)

[**제공 : DB포탈사이트 DBguide.net**](http://www.dbguide.net/)