Microsoft®
SQL Server™ 2005

SQL Server 2005의 인덱싱된 뷰를 통한 성능 향상

Microsoft®

요약

한층 개선된 SQL Server 2005 Enterprise Edition의 인덱싱된 뷰에 대해 설명하고 성능 향상을 실현할 수 있는 구체적인 시나리오에 대해 논의할 것입니다.

Contents

인덱싱된 뷰의 개요

Microsoft® SQL Server™는 오랫동안 "뷰"라고 불리는 가상 테이블을 만드는 기능을 지원했습니다. 지금까지 이들 뷰는 주로 다음과 같은 목적으로 사용되었습니다.

- 1개 이상의 기본 테이블에서 특정한 데이터 하위 집합의 사용을 제한하는 보안 메커니즘 제공
- 사용자가 기본 테이블에 저장된 데이터를 논리적으로 보는 방법을 개발자가 정의할 수 있는 메커니즘 제공

SQL Server 2000에서는 시스템 성능을 향상시킬 수 있도록 SQL Server 뷰의 기능이 강화되었습니다. 클러스터되지 않은 인덱스는 물론, 뷰에서 클러스터된 고유 인덱스를 만들어 가장 복잡한 쿼리의 데이터 액세스 성능을 향상시킬 수 있습니다. SQL Server 2000 및 2005에서는 클러스터된 고유 인덱스를 포함한 뷰를 인덱스된 뷰라고 부릅니다. 여기에서는 주로 SQL Server 2005에 관련된 내용을 설명하고 있으며 SQL Server 2000에 대해서도 상당 부분 다루고 있습니다.
DBMS(Database Management System)의 관점에서 보면 뷰는 메타 데이타 형식의 데이터에 대한 설명입니다. 일반적인 뷰가 생성되면 가상 테이블로 표현되도록 결과 세트를 정의하는 SELECT 명령문을 캡슐화하는 방식으로 메타데이터를 정의할 수 있습니다. 다른 쿼리의 FROM 구문에서 뷰가 참조되면 메타데이터가 시스템 카탈로그에서 검색되어 뷰의 참조 영역으로 확장됩니다. 뷰 확장 이후, SQL Server 쿼리 최적화 프로그램이 실행 쿼리에 대한 단일 실행 계획을 컴파일합니다. 쿼리 최적화 프로그램은 쿼리에 대해 가능한 실행 계획 세트를 검색하여 각 쿼리 계획의 실행에 소요될 것으로 예상되는 실제 시간을 토대로 가장 저렴한 비용의 계획을 선택합니다.
인덱싱되지 않은 뷰에서는 쿼리 해결에 필요한 뷰 부분이 런타임에 구체화됩니다. 뷰를 참조하는 각 쿼리에 대해 쿼리 실행 동안 조인(join)이나 집계(aggregation) 등과 같은 계산 작업이 수행됩니다 1. 뷰에서 클러스터된 고유 인덱스가 만들어 지고 나면 뷰의 결과 세트가 즉시 구체화되고 데이터베이스의 물리적 스토리지에 지속되기 때문에 비용이 많이 드는 이들 계산 작업을 실행 시간에 수행할 필요가 없습니다.
쿼리 실행을 위해 인덱싱된 뷰를 2가지 방식으로 사용할 수 있습니다. 쿼리는 인덱싱된 뷰를 직접 참조할 수 있으며, 중요한 것은 쿼리 최적화 프로그램이 가장 비용이 낮은 쿼리 계획에서 쿼리의 일부 또는 전부를 뷰로 대체할 수 있는지를 평가하여 해당 뷰를 선택할 수 있다는 것입니다. 후자의 경우 원본으로 사용하는 테이블 및 일반 인덱스 대신 인덱싱된 뷰가 사용됩니다. 쿼리 최적화 프로그램이 쿼리 실행 동안 뷰를 사용할 수 있도록 하기 위해 쿼리에서 뷰를 참조할 필요가 없습니다. 이에 따라 기존 응용 프로그램은 그 어떤 변경 작업을 수행하지 않고도 새로 만들어진 인덱싱된 뷰의 이점을 십분 활용할 수 있습니다.

주 인덱싱된 뷰는 SQL Server 2000 및 2005의 모든 버전에서 지원되는 기능입니다. Developer 및 Enterprise Edition의 경우, 쿼리 프로세서가 이름으로 뷰를 참조하지 않는 경우에도 인덱싱된 뷰를 사용하여 구조적으로 뷰에 일치하는 쿼리를 해결할 수 있습니다. 다른 버전에서는 이름으로 뷰를 참조하고 뷰 참조에서 NOEXPAND 힌트를 사용하여 인덱싱된 뷰의 내용을 쿼리해야 합니다.

인덱싱된 뷰를 통한 성능 향상

쿼리 성능 개선을 위해 인덱스를 사용하는 것은 새로운 개념이 아닙니다. 하지만 인덱싱된 뷰는 표준 인덱스로서는 기대할 수 없는 성능 향상을 실현할 수 있습니다. 인덱싱된 뷰는 다음과 같은 방법으로 쿼리 성능을 향상시킬 수 있습니다.

- 쿼리 실행 동안 비용이 많이 드는 계산 작업을 최소화하기 위해 집계를 사전 계산하여 인덱스에 저장
- 테이블을 사전 조인하고 결과 데이터 집합을 저장
- 조인 또는 집계의 조합을 저장

아래 그래프는 쿼리 최적화 프로그램에 인덱싱된 뷰를 사용했을 때 실현되는 일반적인 성능 향상을 보여 주고 있습니다. 표시된 쿼리는 복잡도(예를 들어, 집계 계산의 수, 사용 테이블 수, 조건자의 수)에 차이가 있으며 실제 운영 환경에서 가져온 수백만 개의 행 테이블을 포함하고 있습니다.

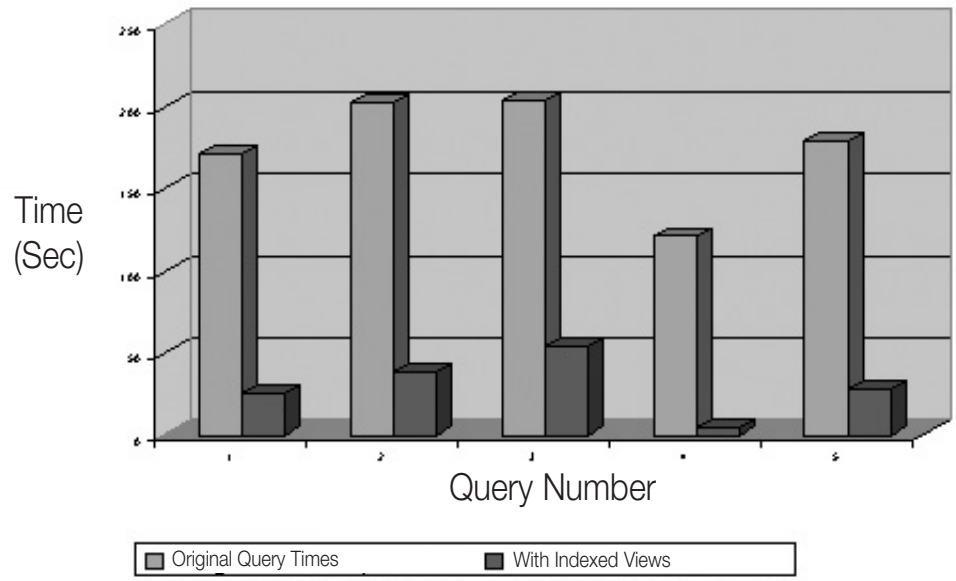

그림 1

뷰에 클러스터되지 않은 인덱스 사용

뷰에 클러스터되지 않은 보조 인덱스를 사용하여 쿼리 성능을 향상시킬 수 있습니다. 테이블에 클러스터되지 않은 인덱스를 사용하는 것과 마찬가지로 뷰에 클러스터되지 않은 인덱스를 사용하면 컴파일 프로세스 동안 쿼리 최적화 프로그램이 선택할 수 있는 보다 많은 옵션이 제공됩니다. 예를 들어 쿼리에 클러스터된 인덱스에 해당되지 않는 열이 포함되어 있을 경우 최적화 프로그램은 쿼리 계획에서 1개 이상의 보조 인덱스를 선택하여 많은 시간이 소요되는 인덱싱된 뷰 또는 기본 테이블에 대한 전체 스캔 작업을 피할 수 있습니다.
인덱스는 지속적인 유지 관리가 필요하기 때문에 스키마에 인덱스를 추가하면 데이터베이스에서 오버헤드가 증가합니다. 인덱스와 유지 관리 오버헤드 간의 적절한 균형을 유지할 수 있도록 세심한 주의가 필요합니다.

인덱싱된 뷰 사용 시 이점

인덱싱된 뷰를 구현하기 앞서 먼저 데이터베이스 워크로드를 분석하고 다양한 도구(SQL Profiler 등과 같은)와 쿼리에 대한 지식을 토대로 인덱싱된 뷰가 효과적인 쿼리를 구별해야 합니다. 인덱싱된 뷰는 빈번하게 실행되는 집계 및 조인 작업에 가장 적합합니다. 질문되는 빈도에 관계없이 답변에 많은 시간이 소요되고 신속한 답변이 무엇보다 중요한 쿼리라면 인덱싱된 뷰를 사용하는 것이 좋습니다. 예를 들어 일부 개발자들은 중역들이 매월 말 실행하는 보고서에서 쿼리에 대한 답변을 사전 계산하여 저장한 인덱싱된 뷰를 만드는 것이 유용하다는 사실을 발견했습니다.

모든 쿼리에서 인덱싱된 뷰가 유용한 것은 아닙니다. 일반 인덱스와 마찬가지로 인덱싱된 뷰를 사용하지 않는다면 개선 효과를 기대할 수 없습니다. 이 경우 성능 향상을 실현할 수 없을 뿐만 아니라 디스크 공간, 유지 관리 및 최적화로 인한 추가 비용이 발생할 수 있습니다. 그러나 인덱싱된 뷰를 사용하면 데이터 액세스가 훨씬 개선(수 배 이상)됩니다. 그 이유는 쿼리 최적화 프로그램이 인덱싱된 뷰에 저장되어 있는 사전 계산된 결과값을 사용하여 쿼리 실행 비용을 크게 줄일 수 있기 때문입니다.

쿼리 최적화 프로그램은 일정 수준 이상의 비용을 발생시키는 쿼리에 대해서만 인덱싱된 뷰를 고려하기 합니다. 따라서 인덱싱된 뷰를 사용하여 실현하는 비용 절감 효과보다 쿼리 최적화 동안 다양한 인덱싱된 뷰를 일치시키는데 더 많은 비용이 발생하는 상황을 피할 수 있습니다. 인덱싱된 뷰는 비용이 1 이하인 쿼리에서는 거의 사용되지 않습니다.

인덱싱된 뷰 구현으로 개선 효과를 거둘 수 있는 응용 프로그램은 다음과 같습니다.

- 의사결정 지원 워크로드
- 데이터 마트
- 데이터 웨어하우스
- OLAP 스토어 및 소스
- 데이터 마이닝 워크로드

쿼리 유형과 패턴 측면에서 다음과 같은 응용 프로그램들이 개선 효과를 거둘 수 있을 것입니다.

대형 테이블의 조인 및 집계

- 반복 패턴의 쿼리
- 동일 또는 중첩된 열 세트의 반복 집계
- 동일 키상에서 동일 테이블의 반복 조인
- 위 항목 모두

반대로 쓰기 작업이 많은 OLTP 시스템이나 업데이트가 잦은 데이터베이스 응용 프로그램은 뷰와 기본 테이블을 업데이트하는 것과 관련한 유지 관리 비용이 증가하기 때문에 인덱싱된 뷰를 사용하는 데 따른 개선 효과를 거둘 수 없습니다.

쿼리 최적화 프로그램의 인덱싱된 뷰 사용 방법

SQL Server 쿼리 최적화 프로그램은 주어진 쿼리 실행을 위해 언제 인덱싱된 뷰를 사용할 수 있는지를 자동으로 결정합니다. 최적화 프로그램에 대한 쿼리에서 직접 뷰를 참조할 필요 없이 쿼리 실행 계획에서 사용할 수 있습니다. 따라서 기존 응용 프로그램을 전혀 변경하지 않고도 인덱싱된 뷰를 활용할 수 있으며 인덱싱된 뷰만 만들면 됩니다.

최적화 프로그램 고려 사항

쿼리 최적화 프로그램은 몇 가지 조건을 고려하여 인덱싱된 뷰가 전체 쿼리를 처리할 것인지, 아니면 쿼리 일부만 처리할 것인지를 결정합니다. 쿼리의 단일 FROM 구문에 해당하는 이들 조건은 다음과 같이 구성됩니다.

- 쿼리 FROM 구문의 테이블은 인덱싱된 뷰 FROM 구문에 있는 테이블의 상위 집합이어야 합니다.
- 쿼리의 조인 조건은 뷰 조인 조건의 상위 집합이어야 합니다.
- 쿼리의 집계 행은 뷰 집계 행의 상위 집합에서 추론할 수 있어야 합니다.
- 선택 쿼리 목록의 모든 표현식은 선택 뷰 목록 또는 뷰 정의에 포함되지 않은 테이블에서 추론할 수 있어야 합니다.
- 해당 열의 상위 집합이 다른 것과 일치하는 경우, 하나의 조건자가 다른 것을 포함할 수 있습니다. 예를 들어, "T.a=10" 은 "T.a=10 및 T.b=20" 을 포함합니다. 모든 조건자는 스스로를 포함합니다. 한 테이블의 값을 제한하는 뷰의 조건자 부분은 동일 테이블을 제한하는 쿼리의 조건자 부분을 포함해야 합니다. 뿐만 아니라 SQL Server가 확인할 수 있는 방식으로 수행되어야 합니다.
- 뷰 정의의 테이블에 속한 쿼리 검색 조건자의 모든 열은 뷰 정의 내에 다음 중 1개 이상의 형태로 나타나야 합니다.
  a. GROUP BY 목록
  b. 선택 뷰 목록(GROUP BY가 없는 경우)
  c. 뷰 정의와 동일하거나 동급의 조건자
  (1)과 (2)의 경우, SQL Server는 뷰의 행을 더욱 엄격하게 제한하기 위해 뷰의 행에 쿼리 조건자를 적용할 수 있습니다. (3)은 열에서 필터링이 필요하지 않은 특별한 경우이기 때문에 열을 뷰 내에 표시할 필요가 없습니다.

쿼리에 1개 이상의 FROM 구문(하위 쿼리, 추론된 테이블, UNION)이 포함되어 있으면 최적화 프로그램이 여러 개의 인덱싱된 뷰를 선택해서 쿼리를 처리하고 이를 서로 다른 FROM 구문에 적용할 수 있습니다.2
이러한 상황을 보여주는 쿼리의 예제가 본 자료의 마지막 부분에 나와 있습니다. 쿼리 최적화 프로그램이 쿼리 실행 계획에서 사용할 인덱스를 결정하도록 하는 것이 가장 좋은 방법입니다.

NOEXPAND 뷰 힌트 사용

SQL Server가 이름으로 뷰를 참조한 쿼리를 처리하면 일반적으로 기본 테이블만 참조할 때까지 뷰의 정의를 확장합니다. 이러한 프로세스를 뷰 확장(view expansion)이라고 부르며 매크로 확장의 한 형식이라고 할 수 있습니다.
NOEXPAND 뷰 힌트는 쿼리 최적화 프로그램이 클러스터된 인덱스를 통해 일반 테이블과 마찬가지로 뷰를 처리하도록 합니다. 이는 뷰 확장을 차단합니다. FROM 구문에서 인덱싱된 뷰가 직접 참조되는 경우에만 NOEXPAND 힌트를 적용할 수 있습니다. 예를 들면 다음과 같습니다.
SELECT Column1, Column2, ... FROM Table1, View1 WITH (NOEXPAND) WHERE ...
기본 테이블에서 데이터 대신 뷰 자체를 읽어 SQL Server가 쿼리를 처리하도록 하려면 NOEXPAND를 사용하십시오. 뷰 사용을 선호하며 몇 가지 이유로 SQL Server가 기본 테이블에 대한 쿼리를 처리하는 쿼리 계획을 선택한 경우라면 NOEXPAND 사용을 고려하십시오. SQL Server가 인덱싱된 뷰에 대한 쿼리를 직접 처리하도록 하려면 Developer 및 Enterprise Edition을 제외한 모든 버전의 SQL Server에서 NOEXPAND를 사용해야 합니다. SQL Server Management Studio 도구인 Display Estimated Execution Plan 기능을 사용하면 명령에 대해 SQL Server가 선택한 계획을 그래픽으로 볼 수 있습니다. 그렇지 않고 SHOWPLAN_ALL, SHOWPLAN_TEXT, SHOWPLAN_XML을 사용하면 그래픽 이외의 형태로 볼 수 있습니다. 다양한 버전의 SHOWPLAN에 대한 논의는 SQL Sever Books Online을 참조하십시오.

EXPAND VIEWS 쿼리 힌트 사용

이름으로 뷰를 참조하는 쿼리를 처리할 때 뷰 참조에 NOEXPAND 힌트를 추가하지 않는 한 SQL Server는 항상 뷰를 확장합니다. 또한 쿼리의 마지막 부분에 있는 OPTION 구문에 EXPAND VIEWS 쿼리 힌트를 지정하지 않는 한 계속해서 인덱싱된 뷰를 확장된 쿼리에 일치시키려고 시도합니다. 예를 들어, 데이터베이스에 인덱싱된 뷰인 View1이 있다고 가정해 봅시다. 다음 쿼리에서 View1은 논리적 정의(CREATE VIEW 명령)에 따라 확장되고 EXPAND VIEWS 옵션은 View1에 대한 인덱싱된 뷰가 쿼리 해결을 위한 계획에서 사용되는 것을 막습니다.
SELECT Column1, Column2, ... FROM Table1, View1 WHERE ...

OPTION (EXPAND VIEWS)
인덱싱된 뷰에 액세스하는 대신 쿼리가 참조한 기본 테이블에서 데이터에 직접 액세스함으로써 SQL Server가 쿼리를 처리하도록 하려면 EXPAND VIEWS를 사용하십시오. EXPAND 뷰는 때때로 인덱싱된 뷰에서 나타날 수 있는 잠금 경합(lock contention)을 제거하는 데 도움이 될 수 있습니다. NOEXPAND와 EXPAND VIEWS 모두 응용 프로그램을 테스트할 때 인덱싱된 뷰의 사용 여부에 따라 성능을 평가할 수 있도록 도와줍니다.

SQL Server 2005의 인덱싱된 뷰가 제공하는 새로운 기능

SQL Server 2005는 SQL Server 2000보다 인덱싱된 뷰를 위한 많은 향상된 기능이 포함되어 있습니다. 인덱스 가능한 뷰 세트는 다음을 토대로 향상된 기능을 포함하도록 확장되었습니다.

- GROUP BY 없이 SUM 및 COUNT_BIG를 포함한 스칼라 집계
- 스칼라 표현식 및 UDF(User-Defined Functions). 예를 들어 테이블 T(a int, b int, c int)와 스칼라 UDF dbo.MyUDF(@x int)가 주어진 경우 T에서 정의된 인덱싱된 뷰에는 a+b나 dbo.MyUDF(a) 같이 계산된 열이 포함될 수 있습니다.
- 지속되는 정확하지 않은 열. float 또는 real 유형의 열이나, float 또는 real 열에서 추론된 계산된 열이 부정확한 열에 해당됩니다. SQL Server 2000에서는 인덱스 키에 포함되지 않은 경우, 인덱싱된 뷰의 select 목록에서 부정확한 열을 사용할 수 있었지만 WHERE 또는 FROM 구문 등과 같은 뷰 정의 내 다른 부분에서는 사용할 수 없었습니다. SQL Server 2005에서는 해당 열이 기본 테이블에서 지속되는 경우, 키나 뷰 정의 내부에 포함시킬 수 있도록 했습니다. 지속적 열에는 일반 열과 PERSISTED라고 표시된 계산된 열이 포함됩니다.
- 부정확하며 지속되지 않는 열이 인덱스나 인덱싱된 뷰에 포함될 수 없는 근본적인 이유는 한 시스템에서 데이터베이스 연결을 해제하고 다른 시스템으로 연결할 수 있어야 하기 때문입니다. 이동한 후 새 하드웨어에서도 기존 하드웨어와 동일한 방식으로 인덱스나 인덱싱된 뷰에 저장된 계산된 모든 열 값을 개별 비트까지 추론할 수 있어야 합니다. 그렇지 않으면 새로운 하드웨어에서 인덱싱된 뷰는 논리적으로 손실될 수 있습니다. 이러한 손실 문제 때문에 새 하드웨어에서 인덱싱된 뷰에 대한 쿼리가 뷰 데이터 추론을 위해 쿼리 계획에서 인덱싱된 뷰를 사용하느냐 기본 테이블을 사용하느냐에 따라 서로 다른 답변을 반환할 수 있었습니다. 뿐만 아니라 인덱싱된 뷰는 새로운 시스템 상에서 제대로 유지될 수 없었습니다. 불행히도 서로 다른 시스템(같은 제조업체의 동일한 프로세서 아키텍처를 기반으로 하더라도)상의 부동 소수점 하드웨어는 프로세서 버전에 따라 달라질 수 있습니다. 일부 부동 소수점 값인 a와 b에서 새 하드웨어의 (a*b)는 기존 하드웨어의 (a*b)와 같지 않기 때문에 펌웨어 업그레이드가 필요합니다. 예를 들어 결과값이 아무리 유사하다 하더라도 LSB(Least Significant Bit)는 서로 다릅니다. 모든 표현식은 데이터베이스 업데이트와 인덱스/인덱싱된 뷰의 유지 관리를 수행하는 동안 같은 시스템상에서 평가되기 때문에, 인덱싱에 앞서 부정확한 계산 값을 지속하여 이러한 연결 해제/연결 불일치 문제를 해결할 수 있습니다.
- CLR(Common Language Runtime) 유형. SQL Server 2005에서 새로 선 보이는 주요 기능으로 CLR 기반의 UDT(User-defined Types)와 UDF 지원을 들 수 있습니다. 열이나 표현식이 명확하고 정확하며 지속적이라면 CLR UDT 열이나 이들 열에서 추론된 표현식에서 인덱싱된 뷰를 정의할 수 있습니다. CLR 사용자 정의 집계는 인덱싱된 뷰에서 사용할 수 없습니다.

쿼리를 인덱싱된 뷰에 일치시키고 쿼리 계획에서 이를 사용할 수 있도록 하는 최적화 프로그램 기능은 다음을 포함할 수 있도록 확장되었습니다.

- 쿼리/뷰의 SELECT 목록이나 조건의 새로운 표현식 유형에는 다음이 포함됩니다.
  - (a+b)/2 같은 스칼라 표현식
  - 스칼라 집계
  - 스칼라 UDF

- 간격 포함. 최적화 프로그램은 인덱싱된 뷰 정의의 간격 조건에 해당되는 범위를 탐지하거나 쿼리에 간격 조건을 "포함"할 수 있습니다. 예를 들어, 최적화 프로그램은 "a〉10 및 a〈20"가 "a〉12 및 a〈18"을 포함한다고 판단할 수 있습니다.
- 표현식 동치. 구문상 다르더라도 동일하게 보일 수 있는 특정 표현식은 동일하게 처리됩니다. 예를 들어, "a=b 및 c〈〉10"은 "10〈〉c 및 b=a"와 동치로 간주됩니다.

또한 데이터베이스에 많은 수의 인덱싱된 뷰가 있을 경우, 뷰가 정의된 테이블의 컴파일 성능은 일반적으로 SQL Server 2000 보다는 SQL Server 2005가 훨씬 우수합니다.

설계 고려 사항

데이터베이스 시스템에 적절한 인덱스 세트를 파악하는 것은 복잡한 작업일 수 있습니다. 일반 인덱스를 설계할 때도 수 많은 가능성을 고려해야 하지만 스키마에 인덱싱된 뷰를 추가하면 설계가 훨씬 복잡해지며 가능한 결과의 수도 증가합니다. 예를 들어 인덱싱된 뷰는 다음에서 사용될 수 있습니다.

- 쿼리에서 참조된 테이블의 모든 하위 집합
- 해당되는 테이블 하위 집합에 대한 쿼리 조건의 모든 하위 집합
- 그룹화 열.
- 집계 함수(예: SUM)

각 구조에서 최상의 결과를 도출하기 위해서는 테이블의 인덱스와 인덱싱된 뷰를 동시에 설계해야 합니다. 인덱스와 인덱싱된 뷰 모두 주어진 쿼리에 유용할 수 있기 때문에 이들을 각각 따로 설계하면 중복된 권장 사항으로 높은 스토리지 및 유지 관리 오버헤드를 야기할 수 있습니다. 데이터베이스의 물리적 설계를 조정하는 동안 다양한 쿼리 세트의 성능 요구 사항과 데이터베이스 시스템이 지원해야 하는 업데이트 간에 상충이 발생합니다. 따라서 인덱싱된 뷰에 대한 적절한 물리적 설계를 파악하는 것이 무엇보다 중요한 과제이기 때문에 가능한 Database Tuning Advisor를 사용하도록 합니다.
쿼리 최적화 프로그램이 특정 쿼리에 대한 많은 인덱싱된 뷰를 고려할 수 있기 때문에 쿼리 최적화 비용이 크게 상승할 수 있습니다. 쿼리 최적화 프로그램은 쿼리의 모든 테이블 하위 집합에 정의되어 있는 모든 인덱싱된 뷰를 고려할 수 있습니다. 거부되기 전에 각 뷰의 대체 가능성에 대해 조사해야 합니다. 주어진 쿼리에 대한 수백 개의 뷰가 존재할 경우 같은 시간이 소요될 수 있습니다.
뷰에 클러스터된 고유 인덱스를 생성하기 위해서는 여러 요구 사항을 충족해야 하며 설계 단계에서 다음과 같은 요구 사항을 고려해야 합니다.

- 뷰와 뷰에서 참조된 모든 테이블은 동일한 데이터베이스 내에 있어야 하며 소유자가 같아야 합니다.
- 인덱싱된 뷰는 최적화 프로그램이 사용하게 될 쿼리에서 참조된 모든 테이블을 포함할 필요가 없습니다.
- 뷰에서 기타 다른 인덱스를 만드려면 먼저 클러스터된 고유 인덱스를 만들어야 합니다.
- 기본 테이블, 뷰 및 인덱스가 생성될 때, 그리고 기본 테이블의 데이터와 뷰가 수정될 때마다 특정 SET 옵션(뒤에서 논의)을 올바르게 설정해야 합니다. 또한 이들 SET 옵션이 정확하지 않은 경우에는 쿼리 최적화 프로그램이 인덱싱된 뷰를 고려하지 않을 것입니다.
- 스키마 바인딩을 사용해서 뷰를 생성해야 하며 SCHEMABINDING 옵션을 이용하여 뷰에서 참조되는 모든 사용자 정의 함수를 생성해야 합니다.
- 인덱싱된 뷰에서 정의된 데이터를 보관하기 위한 추가 디스크 공간이 필요할 것입니다.

설계 지침

인덱싱된 뷰를 설계할 때 다음과 같은 지침을 고려하십시오.

- 여러 쿼리 또는 여러 작업에서 사용할 수 있는 인덱싱된 뷰를 설계하십시오.
- 예를 들어, 열에 대한 SUM과 COUNT_BIG을 포함하고 있는 인덱싱된 뷰는 SUM, COUNT, COUNT_BIG, AVG 함수를 포함하고 있는 쿼리에 의해 사용될 수 있습니다. 기본 테이블에서 전체 행을 검색하지 않고 뷰에서 소수의 행만을 검색할 수 있으며, AVG 함수를 실행하는 데 필요한 일부 계산이 이미 완료되었기 때문에 훨씬 신속하게 쿼리가 수행됩니다.
- 인덱스 키의 크기를 작게 유지하십시오.
- 인덱스 키에서 최소한의 열과 바이트를 사용하면 인덱싱된 뷰 행을 좁혀 인덱싱된 뷰의 행에 보다 효율적으로 액세스할 수 있으며 보다 광범위한 키를 이용할 때보다 키 비교를 신속하게 수행할 수 있습니다. 뿐만 아니라 클러스터된 인덱스 키는 인덱싱된 뷰에 정의된 모든 클러스터되지 않은 인덱스에서 행 로케이터로 사용됩니다. 인덱스 키가 커지면 뷰의 클러스터되지 않은 인덱스 수에 비례해 비용이 상승합니다.
- 만들어진 인덱싱된 뷰의 크기를 고려하십시오.
- 순수 집계의 경우 인덱싱된 뷰의 크기가 원본 테이블의 크기와 비슷하면 큰 폭의 성능 향상이 이루어지지 않을 수도 있습니다.
- 프로세스의 일부를 가속화하는 여러 개의 소형 인덱싱된 뷰를 설계하십시오.
- 전체 쿼리를 지원하는 인덱싱된 뷰를 설계할 수 없을 수도 있습니다. 이 경우 각 쿼리의 일부를 수행하는 인덱싱된 뷰를 여러 개 생성하는 방법을 고려해 보십시오.

다음 예제를 검토해 보십시오.

- 자주 실행되는 쿼리는 한 데이터베이스에서 데이터를 집계하고 다른 데이터베이스에서 데이터를 집계한 다음, 그 결과를 조인합니다. 인덱싱된 뷰는 1개 이상의 데이터베이스에서 테이블을 참조할 수 없기 때문에 전체 프로세스를 수행하는 단일 뷰를 설계할 수 없습니다. 하지만 각 데이터베이스에 인덱싱된 뷰를 만들어 해당 데이터베이스에 대한 집계를 수행할 수 있습니다.
- 최적화 프로그램이 기존 쿼리에 대해 인덱싱된 뷰를 일치시킬 수 있다면 기존 쿼리를 재코딩하지 않고도 최소한 집계 처리를 보다 신속하게 수행할 수 있습니다. 조인 처리 작업이 보다 신속하게 실행되지 않더라도 인덱싱된 뷰에 저장된 집계를 사용하기 때문에 전체 쿼리 속도가 빨라집니다.
- 자주 실행되는 쿼리는 여러 테이블에서 데이터를 집계하고 UNION을 사용해 그 결과를 종합합니다. 인덱싱된 뷰에서는 UNION이 지원되지 않습니다. 개별 집계 작업을 수행하도록 각각의 뷰를 설계할 수 있습니다. 따라서 최적화 프로그램은 인덱싱된 뷰를 선택하여 쿼리 재코딩 작업 없이 쿼리 속도를 높일 수 있습니다. UNION 처리는 향상되지 않지만 개별 집계 프로세스는 향상됩니다.

인덱싱된 뷰의 선택을 지원하는 도구

DTA3(Database Tuning Advisor)는 데이터베이스 관리자가 물리적 데이터베이스 설계를 조정할 수 있도록 돕는 SQL Server 2005 기능입니다. DTA는 기본 테이블의 인덱스와 테이블 및 인덱스 파티셔닝 전략은 물론, 인덱싱된 뷰를 권장합니다. DTA를 사용하면 데이터베이스에 대해 실행된 일반적인 쿼리들의 성능을 최적화하하는 인덱스, 인덱싱된 뷰 및 파티셔닝 전략 등을 결정하는 관리자의 능력을 한층 강화할 수 있습니다. DTA는 매우 다양한 인덱싱된 뷰를 권장할 수 있습니다. 여기에는 "SQL Server 2005의 인덱싱된 뷰가 제공하는 새로운 기능" 에서 설명된 SQL Server 2005의 인덱싱된 뷰를 위한 새로운 기능을 활용하는 뷰도 포함됩니다. DTA를 사용한다고 해서 데이터베이스 관리자가 물리적 스토리지 구조를 설계할 때 올바른 판단을 내리기 위해 노력할 필요가 없다는 것을 의미하는 것은 아니며 물리적 데이터베이스 설계 프로세스를 간소화할 수 있습니다. DTA는 가상의 인덱스, 인덱싱된 뷰, 파티션 구조를 제안하여 비용 기반의 쿼리 최적화 프로그램과 연동할 수 있으며 최적화 프로그램을 사용하여 파티션 구조를 채택한 경우와 그렇지 않은 경우의 워크로드 비용을 예측하고 전체 비용이 낮은 구조를 권장합니다. DTA는 필요한 모든 SET 옵션을 적용하기 때문에(올바른 결과 세트 보장을 위해) 인덱싱된 뷰를 성공적으로 작성할 수 있습니다. 그러나 옵션이 요구에 맞게 설정되지 않은 경우, 응용 프로그램이 뷰를 제대로 활용할 수 없습니다. SET 옵션이 제대로 설정되지 않으면 인덱싱된 뷰 정의에 포함된 테이블에서 삽입, 업데이트, 삭제 작업이 수행되지 않을 수 있습니다.

데이터 업데이트가 인덱싱된 뷰에 미치는 영향

SQL Server는 기본 테이블 데이터의 변경 시 다른 인덱스와 유사한 방식으로 인덱싱된 뷰를 자동 유지 관리합니다. 일반 인덱스의 경우, 단일 테이블에 각각의 인덱스가 직접 연결됩니다. 원본으로 사용하는 테이블에서 각각의 INSERT, UPDATE 또는 DELETE 작업이 수행되고 이에 따라 인덱스가 업데이트되기 때문에 인덱스에 저장된 값을 테이블과 항상 일치시킬 수 있습니다.

인덱싱된 뷰도 이와 유사하게 유지 관리되지만, 뷰가 여러 테이블을 참조할 경우 테이블 업데이트를 위해서는 인덱싱된 뷰가 필요할 수 있습니다. 일반 인덱스와 달리 인덱싱된 뷰에서는 참가 테이블에 단일 행을 삽입하면 여러 행이 변경될 수 있습니다. 단일 행이 다른 테이블의 여러 행을 조인할 수 있기 때문입니다. 업데이트나 삭제 작업에서도 마찬가지입니다. 결론적으로 인덱싱된 뷰의 유지 관리는 테이블 인덱스 유지 관리보다 훨씬 많은 비용이 듭니다. 반대로 뷰가 참조하는 기본 테이블에 대한 삽입, 삭제 및 업데이트 작업은 대부분 뷰에 영향을 미치지 않기 때문에 고도의 선택 상황에서는 인덱싱된 뷰의 유지 관리 비용이 테이블 인덱스 유지 관리 비용 보다 훨씬 적습니다. 다른 데이터베이스 데이터를 액세스하지 않고도 인덱싱된 뷰와 관련해 이러한 작업을 제거할 수 있습니다.
SQL Server에서는 일부 뷰를 업데이트할 수 있습니다. 뷰 업데이트가 가능하면 INSERT, UPDATE 및 DELETE 명령문을 사용하는 뷰를 통해 원본으로 사용하는 기본 테이블이 직접 수정할 수 있습니다. 뷰에서 인덱스를 생성해도 뷰 업데이트를 막을 수는 없습니다. 인덱싱된 뷰를 업데이트하면 실제로 뷰의 원본으로 사용하는 기본 테이블이 업데이트됩니다. 이러한 업데이트는 인덱싱된 뷰 유지 관리의 일환으로서 인덱싱된 뷰까지 자동으로 전달됩니다. 업데이트 가능 뷰에 대한 자세한 내용은 SQL Server 2005를 위한 SQL Server Books Online의 "뷰를 통한 데이터 수정" 을 참조하십시오.

유지 관리 비용 고려 사항

인덱싱된 뷰를 설계할 때 다음과 같은 사항을 고려해야 합니다.

- 인덱싱된 뷰의 경우 데이터베이스에 추가 스토리지가 필요합니다. 일반적인 테이블 스토리지와 마찬가지로 인덱싱된 뷰의 결과 세트는 물리적으로 데이터베이스에 저장됩니다.
- SQL Server는 뷰를 자동 유지 관리하기 때문에 뷰가 정의되어 있는 기본 테이블이 변경되면 뷰 인덱스에서도 1개 이상이 변경 작업이 수행됩니다. 따라서, 유지 관리 오버헤드가 추가로 발생됩니다.

뷰가 제공하는 총 쿼리 실행 절감 효과 및 뷰 저장/유지 관리 비용에서의 차이가 뷰를 통해 달성할 수 있는 순수 성능 개선이라 할 수 있습니다. 뷰에서 필요한 스토리지를 손쉽게 예측할 수 있습니다. SQL Server Management Studio 도구인 Display Estimated Execution Plan을 통해 뷰 정의를 통해 캡슐화된 SELECT 명령문을 평가하십시오. 이 도구를 통해 쿼리가 반환하는 행의 수와 행 크기를 예측할 수 있습니다. 이 두 값을 곱하면 뷰의 크기를 예측할 수 있습니다. 하지만, 이것은 어디까지나 근사치입니다. 뷰 정의에서 쿼리를 실행하거나 뷰에서 인덱스를 생성해야만 뷰의 실제 인덱스 크기를 정확하게 알 수 있습니다.

SQL Server가 수행하는 자동화된 유지 관리 고려 사항의 관점에서 Display Estimated Execution Plan 기능은 이러한 오버헤드의 영향을 파악할 수 있도록 지원합니다. SQL Server Management Studio를 통해 뷰를 수정한 명령문(뷰에서의 UPDATE, 기본 테이블로의 INSERT)을 평가하면 해당 명령문에 대해 표시된 실행 계획에 유지 관리 작업이 포함됩니다. 프로덕션 환경에서 이 작업이 수행되는 회수와 비용을 고려하면 뷰 유지 관리 비용을 예측할 수 있습니다.
일반 권장사항으로 뷰나 원본으로 사용하는 기본 테이블에 대한 모든 수정 또는 업데이트는 가능한 단독 방식이 아닌 배치 방식으로 수행되어야 합니다. 따라서, 뷰 유지 관리에 있어 오버헤드를 줄일 수 있습니다.

인덱싱된 뷰 생성

인덱싱된 뷰 생성을 위해 거쳐야 하는 단계들은 성공적인 뷰를 구현하는 데 있어서도 매우 중요합니다.

1. 뷰에서 참조하게 될 기존의 모든 테이블에서 ANSI_NULLS가 올바르게 설정되었는지 확인합니다.
2. 새로운 테이블을 생성하기 앞서 아래 표에서와 같이 현재 세션에 대해 ANSI_NULLS가 올바르게 설정되었는지 확인합니다.
3. 뷰를 생성하기 앞서 아래 표에서와 같이 현재 세션에 대해 ANSI_NULLS 및 QUOTED_IDENTIFIER가 올바르게 설정되었는지 확인합니다.
4. 뷰 정의가 명확한지 확인합니다.
5. WITH SCHEMABINDING 옵션을 사용해 뷰를 생성합니다.
6. 뷰에서 클러스터된 고유 인덱스를 생성하기 앞서 아래 표에서와 같이 세션의 SET 옵션이 올바르게 설정되었는지 확인합니다.
7. 뷰에서 클러스터된 고유 인덱스를 생성합니다.
8. 기존 테이블이나 뷰에서 OBJECTPROPERTY 함수를 사용해 ANSI_NULLS 및 QUOTED_IDENTIFIER 값을 확인할 수 있습니다.

일관된 결과를 얻기 위한 SET 옵션 사용

쿼리가 실행될 때 현재 세션에서 다양한 SET 옵션이 지원될 경우 동일한 표현식을 평가해도 SQL Server 2005에서 다른 결과값이 나올 수 있습니다. 예를 들어, SET 옵션인 CONCAT_NULL_YIELDS_NULL이 ON으로 설정된 이후에는 표현식 'abc' + NULL은 NULL 값을 반환합니다. 하지만, CONCAT_NULL_YIEDS_NULL이 OFF로 설정된 후에는 동일한 표현식으로 'abc' 가 생성됩니다. 인덱싱된 뷰는 뷰가 올바르게 유지 관리되고 일관된 결과를 반환할 수 있도록 현재 세션에 대한 여러 SET 옵션과 뷰가 참조한 객체에 있어 고정 값을 요구합니다.
현재 세션의 SET 옵션인 ANSI_NULLS과 QUOTED_IDENTIFIER 은 인덱스를 만드려는 뷰가 생성될 때 모두 ON으로 설정되어야 합니다. 그 이유는 이들 두 옵션이 시스템 카탈로그에서 뷰 정의를 통해 저장되기 때문입니다.

다음과 같은 경우가 발생할 때마다 현재 세션의 SET 옵션을 현재 세션의 Required Value 열에 표시된 값으로 설정해야 합니다.

- 뷰에서 인덱스가 생성
- 인덱싱된 뷰에 참여하고 있는 모든 테이블에서 INSERT, UPDATE 또는 DELETE 작업이 수행
- 쿼리 최적화 프로그램이 쿼리 계획을 개발하기 위해 인덱싱된 뷰 사용

SET 옵션	필요한 값	기본 서버 값	.Net SqlClient, OLE DB 및 ODBC 값	DB LIB 값
ANSI_NULLS	ON	OFF	ON	OFF
ANSI_PADDING	ON	OFF	ON	OFF
ANSI_WARNINGS	ON	OFF	ON	OFF
CONCAT_NULL_YIELDS_NULL	ON	OFF	ON	OFF
NUMERIC_ROUNDABORT	OFF	OFF	OFF	OFF
QUOTED_IDENTIFIER	ON	OFF	ON	OFF

인덱싱된 뷰를 생성하기 위해서는 현재 세션에서 ARITHABORT 옵션이 ON으로 설정되어 있어야 합니다. 하지만, SQL Server 2005의 경우 일단 ANSI_WARNINGS가 ON으로 설정되면 ARITHABORT 옵션도 암묵적으로 ON으로 설정되기 때문에 별도의 명시적 설정이 필요하지 않습니다. .Net SqlClient, OLE DB 또는 ODBC 서버 연결을 사용하면 인덱싱된 뷰의 생성, 사용 및 유지 관리를 위해 기본 설정값에서 SET 옵션을 수정할 필요가 전혀 없습니다. sp_configure를 사용하는 서버 수준에서나 SET 명령어를 사용하는 응용 프로그램에서 모든 DB LIB 값을 올바르게 설정해야 합니다. SET 옵션에 대한 자세한 내용은 SQL Server Books Online의 "SQL Server에서의 옵션 사용" 을 참조하십시오.

확정적 함수 사용

인덱싱된 뷰에 대한 정의는 명확해야 합니다. WHERE 및 GROUP BY 구문을 비롯해 select 목록에 있는 모든 표현식이 명확하면 뷰도 명확합니다. 특정한 입력값 세트를 통해 평가되는 모든 확정적 표현식은 항상 동일한 결과를 반환합니다. 확정적 함수만이 확정적 표현식에 참여하게 됩니다. 예를 들어, DATEADD은 3개의 매개변수에 대해 주어진 모든 인수값 세트에서 동일한 결과를 반환한다는 점에서 확정적 함수라 할 수 있습니다. GETDATE는 동일한 인수를 통해 호출되지만 실행 시 마다 반환 값이 달라진다는 점에서 확정적이라 할 수 없습니다. 자세한 내용은 SQL Server Books Online의 "확정적 함수 및 비확정적 함수" 를 참조하십시오.

표현식이 확정적임에도 불구하고 float 표현식을 포함하고 있을 경우, 정확한 결과는 프로세서 아키텍처나 마이크로코드 버전에 의해 결정될 수 있습니다. SQL Server 2005에서는 시스템 간의 데이터베이스 이동 시 데이터 무결성을 보장하기 위해 이러한 표현식을 인덱싱된 뷰의 키가 아닌 열로서만 참여시키고 있습니다. float 표현식을 포함하지 않은 확정적 표현식을 정확한 표현식이라고 합니다. 지속적인 확정적 표현식이나 정확한 표현식만이 인덱싱된 뷰의 키 열 및 WHERE 또는 GROUP BY 구문에 포함될 수 있습니다. 지속적 표현식은 일반 열과 PERSISTED라고 표시된 계산된 열을 포함한 저장된 열에 대한 참조로 사용됩니다.

COLUMNPROPERTY 함수와 IsDeterministic 속성을 사용하여 뷰 열이 확정적인지 확인하십시오. 또한 COLUMNPROPERTY 함수와 IsPrecise 속성을 사용해 SCHEMABINDING에서 뷰의 확정적 열이 정확한지 확인하십시오. COLUMNPROPERTY는 속성이 TRUE면 1을, FALSE나 유효하지 않은 입력을 뜻하는 NULL이면 0을 반환합니다. 예를 들어 이 스크립트에서

```
CREATE TABLE T(a int, b real, c as getdate(), d as a+b)
CREATE VIEW VT WITH SCHEMABINDING AS SELECT a, b, c, d FROM dbo.T
SELECT object_id( 'VT' ), COLUMNPROPERTY(object_id( 'VT' ),' b' ,' IsPrecise' )
```

b 열이 real 유형이기 때문에 SELECT는 IsPrecise에 대해 0을 반환합니다. COLUMNPROPERTY를 통해 T의 다른 열이 확정적이며 정확한지 확인할 수 있습니다.

추가 요구 사항

인덱싱 가능한 뷰 세트는 가능한 뷰 세트의 하위 집합이라 할 수 있습니다. 인덱싱이 가능한 모든 뷰는 인덱스 사용 여부에 관계없이 존재할 수 있습니다.
뷰에서 클러스터된 고유 인덱스를 생성하기 위해서는 설계 지침의 "일관성있는 결과를 얻기 위한 SET 옵션 사용"과 "확정적 함수 사용" 부분에서 언급한 요구 사항 외에도 아래와 같은 요구 사항을 충족해야 합니다.

기본 테이블 요구 사항

- 뷰가 참조하는 기본 테이블은 테이블 생성 시 SET 옵션 ANSI_NULLS를 올바른 값으로 설정해야 합니다. OBJECTPROPERTY 함수를 사용해서 기존 테이블의 ANSI_NULLS 값을 확인할 수 있습니다.

함수 요구 사항

- 뷰가 참조하는 사용자 정의 함수는 WITH SCHEMABINDING 옵션을 사용하여 생성해야 합니다.

뷰 요구 사항

- WITH SCHEMABINDING 옵션을 사용해 뷰를 생성해야 합니다.
- 테이블은 두 부분으로 된 이름(schemaname.tablename)을 사용하는 뷰에 의해 참조되어야 합니다.
- 사용자 정의 함수는 두 부분으로 된 이름(schemaname.functionname)을 사용하는 뷰에 의해 참조되어야 합니다.
- SET 옵션인 ANSI_NULLS와 QUOTED_IDENTIFIER를 올바른 값으로 설정해야 합니다.

뷰 제약

SQL Server 2005의 뷰에서 인덱스를 생성하려면 뷰 정의에 다음이 포함되지 않아야 합니다:

ANY, NOT ANY	OPENROWSET, OPENQUERY, OPENDATASOURCE
부정확한(float, real) 값에 대한 계산	OPENXML
COMPUTE, COMPUTE BY	ORDER BY
부정확한 결과를 생성하는 CONVERT	OUTER 조인
COUNT(*)	클러스터된 인덱스가 비활성화 되었을 경우 기본 테이블 참조
GROUP BY ALL	다른 데이터베이스의 테이블이나 함수 참조
추론된 테이블(FROM 목록의 하위 쿼리)	다른 뷰 참조
DISTINCT	ROWSET 함수
EXISTS, NOT EXISTS	자체 조인
집계 결과에 대한 표현식(예: SUM(x)+SUM(x))	STDEV, STDEVP, VAR, VARP, AVG
완전 텍스트 형식의 조건자(CONTAINS, FREETEXT, CONTAINSTABLE, FREETEXTTABLE)	하위 쿼리
부정확한 상수(예: 2.34e5)	nullable 표현식의 SUM
인라인 또는 테이블 반환 함수	테이블 힌트(예: NOLOCK)
MIN, MAX	text, ntext, image, filestream, XML 열
비확정적 표현식	TOP
비 유니코드 비교	UNION
SQL Server 2005가 탐지할 수 있는 모순으로 뷰가 비어있다는 것을 의미(예: where 0=1 등 ...)	

주 인덱싱된 뷰는 float 및 real 열을 포함할 수 있지만, 지속되지 않는 계산된 열일 경우에는 클러스터된 인덱스 키에 포함될 수 없습니다.

GROUP BY 제약

GROUP BY가 존재할 경우 VIEW 정의에는 다음과 같은 제약이 따릅니다.

- COUNT_BIG(*)를 포함해야 합니다.
- HAVING, CUBE, ROLLUP 또는 GROUPING()을 포함하지 않아야 합니다.

이러한 제약은 인덱싱된 뷰 정의에만 적용됩니다. 쿼리는 이러한 GROUP BY 제약을 충족하지 않은 경우에도 실행 계획에서 인덱싱된 뷰를 사용할 수 있습니다.

인덱스 요구 사항

- CREATE INDEX 명령문을 실행하는 사용자는 뷰 소유자일 가능성이 높습니다.
- 뷰 정의에 GROUP BY 구문이 포함되어 있으면 클러스터된 고유 인덱스 키가 GROUP BY 구문에 지정되어 있는 열만 참조할 수 있습니다.
- IGNORE_DUP_KEY 옵션이 활성화되어 있는 경우 인덱스를 생성하면 안됩니다.

예제

여기나와 있는 예제는 집계와 조인이라는 2개의 주요 쿼리 그룹에서의 인덱싱된 뷰 사용을 설명하기 위한 것입니다. 또한 인덱싱된 뷰의 적용 가능 여부를 판단할 때 쿼리 최적화 프로그램이 사용하는 조건을 보여주고 있습니다. 전체 조건 목록에 관한 내용은 "쿼리 최적화 프로그램의 인덱싱된 뷰 사용 방법" 을 참조하십시오.
쿼리는 SQL Server 2005에서 제공되는 샘플 데이터베이스인 AdventureWorks의 테이블을 토대로 하며 작성과 함께 실행 가능합니다. SQL Server Management Studio의 Display Estimated Execution Plan 도구를 사용해서 뷰 생성 이전이나 이후에 쿼리 최적화 프로그램이 선택한 계획을 보고자 할 수 있습니다. 여기 나와있는 예제들은 최적화 프로그램이 어떻게 비용이 낮은 실행 계획을 선택하는지 보여주기 위한 것이지만, AdventureWorks 샘플 데이터베이스는 성능 이점을 보여주기에는 너무 작습니다.
예제를 살펴보기 앞서 이러한 명령어를 실행하여 세션에 올바른 옵션이 설정되어 있는지 확인하십시오.

설치

```
SET ANSI_NULLS ON
SET ANSI_PADDING ON
SET ANSI_WARNINGS ON
SET CONCAT_NULL_YIELDS_NULL ON
SET NUMERIC_ROUNDABORT OFF
SET QUOTED_IDENTIFIER ON
SET ARITHABORT ON
```

아래 쿼리는 Sales.SalesOrderDetail 테이블에서 총 할인율이 가장 큰 5개 제품을 반환하는 2가지 방법을 보여주고 있습니다.

Query 1

```
SELECT TOP 5 ProductID, Sum(UnitPrice*OrderQty) -
    Sum(UnitPrice*OrderQty*(1.00-UnitPriceDiscount)) AS Rebate
FROM Sales.SalesOrderDetail
GROUP BY ProductID
ORDER BY Rebate DESC
```

Query 2

```
SELECT TOP 5 ProductID,
SUM(UnitPrice*OrderQty*UnitPriceDiscount) AS Rebate
FROM Sales.SalesOrderDetail
GROUP BY ProductID
ORDER BY Rebate DESC
```

쿼리 최적화 프로그램이 선택한 실행 계획에는 다음이 포함되어 있습니다.

- 행 수가 121,317개로 예측되는 Sales.SalesOrderDetail 테이블의 클러스터된 인덱스 스캔
- GROUP BY 열에 따라 선택된 행을 hash 테이블로 보내고 각각의 행에 대해 SUM 집계를 계산하는 hash 일치/집계 연산자
- ORDER BY 구문을 토대로 하는 상위 5개의 정렬 연산자

View 1

Rebate 열에 필요한 집계를 포함하는 인덱싱된 뷰를 추가하면 Query 1에 대한 쿼리 실행 계획이 변경됩니다. 대규모 테이블(수백만 개의 행)에서 쿼리 성능은 크게 개선됩니다.

```
CREATE VIEW Vdiscount1 WITH SCHEMABINDING AS
SELECT SUM(UnitPrice*OrderQty) AS SumPrice,
SUM(UnitPrice*OrderQty*(1.00-UnitPriceDiscount)) AS SumDiscountPrice,
COUNT_BIG(*) AS Count, ProductID
FROM Sales.SalesOrderDetail
GROUP BY ProductID
GO
CREATE UNIQUE CLUSTERED INDEX VDiscountInd ON Vdiscount1 (ProductID)
```

Query 1에 대한 실행 계획은 최적화 프로그램에서 Vdiscount1 뷰가 사용되고 있다는 것을 보여줍니다. 그러나, SUM(UnitPrice*OrderQty*UnitPriceDiscount) 집계를 포함하고 있지 않다는 점에서 이 뷰는 Query 2에서 사용되지 않을 것입니다. 두 쿼리 모두를 지원하는 또 다른 인덱싱된 뷰를 생성할 수 있습니다.

View 2

```
CREATE VIEW Vdiscount2 WITH SCHEMABINDING AS
SELECT SUM(UnitPrice*OrderQty) AS SumPrice,
SUM(UnitPrice*OrderQty*(1.00-UnitPriceDiscount)) AS SumDiscountPrice,
SUM(UnitPrice*OrderQty*UnitPriceDiscount)AS SumDiscountPrice2,
COUNT_BIG(*) AS Count, ProductID
FROM Sales.SalesOrderDetail
GROUP BY ProductID
GO
CREATE UNIQUE CLUSTERED INDEX VDiscountInd ON Vdiscount2 (ProductID)
```

이러한 인덱싱된 뷰를 사용할 경우, Vdiscount1 삭제 이후 두 쿼리에 대한 쿼리 실행 계획에는 다음이 포함됩니다.

- 행 수가 266개로 예측되는 Vdiscount2 뷰의 클러스터된 뷰 스캔
- ORDER BY 구문을 기반으로 하는 상위 5개의 정렬 함수

쿼리 최적화 프로그램은 쿼리에 의해 참조가 되지 않는 경우에도 최저의 실행 비용을 제공한다는 점에서 뷰를 선택했습니다.

Query 3

Query 3은 이전 쿼리와 유사하지만 ProductID가 뷰 정의에 포함되지 않는 SalesOrderID 열로 교체되었습니다. 이는 쿼리 계획에서 인덱싱된 뷰를 사용하기 위해서는 뷰 정의 시 테이블에 있는 쿼리 select 목록의 모든 표현식은 선택 뷰 목록에서 추론해야 한다는 조건에 위배되는 것입니다.

```
SELECT TOP 3 SalesOrderID,
SUM(UnitPrice*OrderQty*UnitPriceDiscount) OrderRebate
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
ORDER BY OrderRebate DESC
```

이러한 쿼리를 지원하기 위해서는 별도의 인덱싱된 뷰가 필요합니다. SalesOrderID를 포함하도록 Vdiscount2를 수정할 수 있었지만, 그 결과 뷰에는 원래 테이블 만큼이나 많은 열이 포함되었고 이로 인해 기본 테이블 사용과 비교해 성능 개선을 기대하기 어려웠습니다.

Query 4

각 제품에 대한 평균 가격을 산출하는 쿼리입니다.

```
SELECT p.Name, od.ProductID,
AVG(od.UnitPrice*(1.00-od.UnitPriceDiscount)) AS AvgPrice,
SUM(od.OrderQty) AS Units
FROM Sales.SalesOrderDetail AS od, Production.Product AS p
WHERE od.ProductID=p.ProductID
GROUP BY p.Name, od.ProductID
```

복잡한 집계(예: STDEV, VARIANCE, AVG)는 인덱스 뷰의 정의에 포함될 수 없습니다. 하지만, 결합을 통해 복잡한 집계 작업을 수행하는 단순 집계 함수를 포함시킴으로써 AVG를 포함한 쿼리를 실행하기 위해 인덱싱된 뷰를 사용할 수 있습니다.

View 3

이러한 인덱싱된 뷰에는 AVG 함수 실행에 필요한 단순 집계 함수가 포함됩니다. View 3 생성 후 Query 4가 실행되면 실행 계획은 사용 중인 뷰를 표시합니다. 최적화 프로그램은 뷰의 단순 집계 열인 Price와 Count에서 AVG 표현식을 추론할 수 있습니다.

```
CREATE VIEW View3 WITH SCHEMABINDING AS
SELECT ProductID, SUM(UnitPrice*(1.00-UnitPriceDiscount)) AS Price,
COUNT_BIG(*) AS Count, SUM(OrderQty) AS Units
FROM Sales.SalesOrderDetail
GROUP BY ProductID
GO
CREATE UNIQUE CLUSTERED INDEX iv3 ON View3 (ProductID)
```

Query 5

이 쿼리는 1가지 검색 조건을 추가로 포함하고 있다는 점을 제외하고는 Query 4와 동일합니다. 추가 검색 조건이 뷰 정의에 포함되지 않은 테이블의 열만 참조함에도 불구하고 View 3은 이 쿼리를 위해 작동하게 됩니다.

```
SELECT p.Name, od.ProductID,
AVG(od.UnitPrice*(1.00-UnitPriceDiscount)) AS AvgPrice,
SUM(od.OrderQty) AS Units
FROM Sales.SalesOrderDetail AS od, Production.Product AS p
WHERE od.ProductID=p.ProductID AND p.Name like  '%Red%'
GROUP BY p.Name, od.ProductID
```

Query 6

쿼리 최적화 프로그램은 이 쿼리에서 View 3을 사용할 수 없습니다. 추가된 검색 조건 od.UnitPrice〉10에는 뷰 정의 시 테이블의 열이 포함되지만, 이 열은 GROUP BY 목록에 표시되지 않고 검색 조건자도 뷰 정의에 표시되지 않습니다.

```
SELECT p.Name, od.ProductID,
AVG(od.UnitPrice*(1.00-UnitPriceDiscount)) AS AvgPrice,
SUM(od.OrderQty) AS Units
FROM Sales.SalesOrderDetail AS od, Production.Product AS p
WHERE od.ProductID = p.ProductID AND od.UnitPrice 〉 10
GROUP BY p.Name, od.ProductID
```

Query 7

반대로, 새로운 검색 조건인 od.ProductID between 0 and 995에 정의된 열이 뷰 정의의 GROUP BY 구문에 포함되기 때문에 쿼리 최적화 프로그램은 Query 7에서 View 3을 사용할 수 있습니다.

```
SELECT p.Name, od.ProductID,
AVG(od.UnitPrice*(1.00-UnitPriceDiscount)) AS AvgPrice,
SUM(od.OrderQty) AS Units
FROM Sales.SalesOrderDetail AS od, Production.Product AS p
WHERE od.ProductID = p.ProductID AND od.ProductID between 0 and 995
GROUP BY p.Name, od.ProductID
```

View 4

이 뷰는 쿼리의 AVG가 계산되도록 뷰 정의에 SumPrice 및 Count 열을 포함시켜 Query 6에 대한 조건을 충족할 것입니다.

```
CREATE VIEW View4 WITH SCHEMABINDING AS
SELECT p.Name, od.ProductID,
SUM(od.UnitPrice*(1.00-UnitPriceDiscount)) AS SumPrice,
SUM(od.OrderQty) AS Units, COUNT_BIG(*) AS Count
FROM Sales.SalesOrderDetail AS od, Production.Product AS p
WHERE od.ProductID = p.ProductID AND od.UnitPrice 〉 10
GROUP BY p.Name, od.ProductID
GO
CREATE UNIQUE CLUSTERED INDEX VdiscountInd on View4 (Name, ProductID)
```

Query 8

Sales.SalesOrderHeader 테이블에 대한 조인이 추가되는 쿼리에서도 View 4의 동일 인덱스가 사용될 것입니다. 이 쿼리는 쿼리 FROM 구문에 열거된 테이블은 인덱싱된 뷰의 FROM 구문에 있는 테이블의 상위 집합이어야 한다는 조건을 충족합니다.

```
SELECT p.Name, od.ProductID,
AVG(od.UnitPrice*(1.00-UnitPriceDiscount)) AS AvgPrice,
SUM(od.OrderQty) AS Units
FROM Sales.SalesOrderDetail AS od, Production.Product AS p,
    Sales.SalesOrderHeader AS o
WHERE od.ProductID = p.ProductID AND o.SalesOrderID = od.SalesOrderID
    AND od.UnitPrice > 10
GROUP BY p.Name, od.ProductID
```

마지막 2개의 쿼리는 Query 8을 변형한 것입니다. 각각의 변형 쿼리는 최적화 프로그램 조건 가운데 하나에 위배되고 Query 8과 달리 View 4를 사용할 수 없습니다.

Query 8a

뷰 정의의 UnitPrice > 10과 쿼리의 UnitPrice > 25 간의 WHERE 구문 불일치와 UnitPrice가 뷰에 나타나지 않는다는 사실 때문에 Q8a는 인덱싱된 뷰를 사용할 수 없습니다. 이러한 쿼리 검색 조건자는 뷰 정의에 있는 검색 조건자의 상위 집합이 되어야 합니다.

```
SELECT p.Name, od.ProductID, AVG(od.UnitPrice*(1.00-UnitPriceDiscount))
    AvgPrice, SUM(od.OrderQty) AS Units
FROM Sales.SalesOrderDetail AS od, Production.Product AS p,
    Sales.SalesOrderHeader AS o
WHERE od.ProductID = p.ProductID AND o.SalesOrderID = od.SalesOrderID
    AND od.UnitPrice > 25
GROUP BY p.Name, od.ProductID
```

Query 8b

Sales.SalesOrderHeader 테이블이 인덱싱된 뷰 V4 정의에 참여하지 않는다는 사실에 유의하십시오. 그럼에도 불구하고 이 테이블에 조건자를 추가하면 인덱싱된 뷰를 사용할 수 없게 됩니다. 추가된 조건자가 아래 Query 8b에 나타난 집계에 참여하는 추가 행을 변경 또는 제거할 수 있기 때문입니다.

```
SELECT p.Name, od.ProductID, AVG(od.UnitPrice*(1.00-UnitPriceDiscount))
    AS AvgPrice, SUM(od.OrderQty) AS Units
FROM Sales.SalesOrderDetail AS od, Production.Product AS p,
    Sales.SalesOrderHeader AS o
WHERE od.ProductID = p.ProductID AND o.SalesOrderID = od.SalesOrderID
    AND od.UnitPrice > 10 AND o.OrderDate > '20040728'
GROUP BY p.Name, od.ProductID
```

View 4a

View 4a는 select 목록 및 GROUP BY 구문에 UnitPrice 열을 포함시킴으로써 View 4를 확장합니다. 25 이상의 값만 남겨 두도록 10 이상으로 알려진 UnitPrice 값이 추가 필터링되기 때문에 Query 8a는 View 4a를 사용할 수 있습니다. 이는 간격 포함의 한 예입니다.

```
CREATE VIEW View4a WITH SCHEMABINDING AS
SELECT p.Name, od.ProductID, od.UnitPrice,
SUM(od.UnitPrice*(1.00-UnitPriceDiscount)) AS SumPrice,
SUM(od.OrderQty) AS Units, COUNT_BIG(*) AS Count
FROM Sales.SalesOrderDetail AS od, Production.Product AS p
WHERE od.ProductID = p.ProductID AND od.UnitPrice > 10
GROUP BY p.Name, od.ProductID
GO
CREATE UNIQUE CLUSTERED INDEX VdiscountInd
    ON View4a (Name, ProductID, UnitPrice)
```

View 5

View 5는 select 및 GROUP BY 목록에 표현식을 포함하고 있습니다. LineTotal은 계산된 열이기 때문에 그 자체가 표현식이라는 점에 주목하십시오. 이 표현식은 FLOOR 함수에 대한 호출 내에서 중첩됩니다.

```
CREATE VIEW View5 WITH SCHEMABINDING AS
SELECT FLOOR(LineTotal) FloorTotal, COUNT_BIG(*) C
FROM Sales.SalesOrderDetail
GROUP BY FLOOR(LineTotal)
GO
CREATE UNIQUE CLUSTERED INDEX iView5 ON View5(FloorTotal)
```

Query 9

Query 9는 select 및 GROUP BY 목록에 FLOOR(LineTotal) 표현식을 포함하고 있습니다. SQL Server 2005의 표현식으로 뷰 일치가 확장됨에 따라 이 쿼리는 View 5에서 인덱스를 사용할 수 있습니다.

```
SELECT TOP 5 FLOOR(LineTotal), Count(*)
FROM Sales.SalesOrderDetail
GROUP BY FLOOR(LineTotal)
ORDER BY COUNT(*) DESC
```

View 6

View 6은 매월 마지막 3일 간 품목 번호에 대한 정보를 저장합니다. 적은 수의 페이지에 이들 행을 클러스터하기 때문에 이 기간 동안의 Sales.SalesOrderDetail에 대한 쿼리를 신속하게 처리할 수 있습니다.

```
CREATE VIEW View6 WITH SCHEMABINDING AS
SELECT SalesOrderID, SalesOrderDetailID, CarrierTrackingNumber, OrderQty,
    ProductID, SpecialOfferID, UnitPrice, UnitPriceDiscount, rowguid,
    ModifiedDate
FROM Sales.SalesOrderDetail
WHERE ModifiedDate IN ( convert(datetime,  '2004-07-31' , 120),
                convert(datetime,  '2004-07-30' , 120),
                convert(datetime,  '2004-07-29' , 120) )
GO
CREATE UNIQUE CLUSTERED INDEX VEndJulyO4Ind
    ON View6(SalesOrderID, SalesOrderDetailID)
GO
```

Query 10

아래 쿼리는 View 6와 일치하며 시스템은 Sales.SalesOrderDetail 테이블 전체를 스캔하는 대신 뷰에서 VendJuly04Ind 인덱스를 스캔하는 계획을 생성할 수 있습니다. 또한 이는 표현식 동치(뷰가 아닌 쿼리에서 날짜 순서가 다르고 데이터 형식이 서로 다르기 때문)와 조건자 포함(뷰에 저장된 결과의 하위 집합에 대해 쿼리가 실행되기 때문)을 입증합니다.

```
SELECT h.*, SalesOrderDetailID, CarrierTrackingNumber, OrderQty,
  ProductID, SpecialOfferID, UnitPrice, UnitPriceDiscount, d.rowguid,
  d.ModifiedDate
FROM Sales.SalesOrderHeader as h, Sales.SalesOrderDetail as d
WHERE (d.ModifiedDate =  '20040729'  OR d.ModifiedDate =  '20040730' )
and d.SalesOrderID=h.SalesOrderID
```

View 7

또한 개발자들은 때때로 특수한 무결성 제약 조건을 적용하는 데 인덱싱된 뷰를 사용하는 것이 편리하다는 사실을 발견하게 됩니다. 예를 들어, 인덱싱된 뷰에서는 "열에 여러 개의 0 값이 있는 경우를 제외하고 테이블 T의 열 a는 고유하다"는 제약 조건을 적용할 수 있습니다. 아래에 나와있는 인덱싱된 뷰 View7은 이러한 제약 조건을 적용합니다. 다음과 같은 스크립트를 실행하면 View 7은 최종 삽입 작업 이전까지 성공적으로 실행됩니다. 이러한 명령문은 0이 아닌 중복 값을 추가할 수 있기 때문에 허용되지 않습니다.

```
USE tempdb
GO
CREATE TABLE T(a int)
GO
CREATE VIEW View7 WITH SCHEMABINDING
AS SELECT a
FROM dbo.T
WHERE a <> 0
GO
CREATE UNIQUE CLUSTERED INDEX IV on View7(a)
GO
-- legal:
INSERT INTO T VALUES(1)
INSERT INTO T VALUES(2)
INSERT INTO T VALUES(0)
INSERT INTO T VALUES(0) -- duplicate 0

-- dissalowed:
INSERT INTO T VALUES(2)
```

인덱싱된 뷰에 대한 FAQ

Q. 인덱스를 생성할 수 있는 뷰 종류에 제약이 있는 이유는 무엇입니까?

A. 뷰 유지 관리를 점진적으로 확대하는 것이 논리적으로 가능하도록 하고 유지 관리에 대한 부담을 증가시키는 뷰 생성 기능을 제한하며 SQL Server 시스템의 복잡성을 줄이기 위한 것입니다. 대형 뷰 세트가 비확정적이고 컨텍스트 종속적이기 때문에 DML 연산에 따라 컨텐트가 '변경' 됩니다. 이들 뷰는 인덱싱을 실행할 수 없습니다. GETDATE 또는 SUSER_SNAME라고 정의된 모든 뷰를 예로 들 수 있습니다.

Q. 뷰의 첫번째 인덱스가 클러스터되어야 하고 고유해야 하는 이유는 무엇입니까?

A. 인덱싱된 뷰를 유지 관리하는 동안 키 값으로 뷰 레코드를 손쉽게 검색할 수 있도록 하고 유지 관리에 특별 로직이 필요한 복제본이 포함된 뷰가 생성되는 것을 막기 위해서는 고유해야 합니다. 또한 클러스터된 인덱스만이 고유한 값을 갖도록 하고 동시에 여러 행을 저장할 수 있기 때문에 클러스터되어야 합니다.

Q. 쿼리 최적화 프로그램이 쿼리 계획에서의 사용을 위해 인덱싱된 뷰를 선택하지 않는 이유는 무엇입니까?

A. 최적화 프로그램이 인덱싱된 뷰를 선택하지 않는 이유는 크게 3가지로 볼 수 있습니다.

(1) Enterprise 또는 Developer 에디션이 아닌 다른 버전의 SQL Server를 사용하고 있습니다. Enterprise 및 Developer 에디션만이 자동 쿼리 및 인덱싱된 뷰 일치 기능을 지원합니다. 쿼리 프로세서가 기타 모든 에디션에서 인덱싱된 뷰를 사용할 수 있도록 이름으로 인덱싱된 뷰를 참조하고 NOEXPAND 힌트를 포함시키십시오.

(2) 인덱싱된 뷰를 사용하는 비용이 기본 테이블에서 데이터를 입수하는 비용을 넘어서거나, 쿼리가 간단해 기본 테이블에 대한 쿼리를 신속하고 손쉽게 찾을 수 있습니다. 인덱싱된 뷰가 소형 테이블에 정의될 때 종종 이러한 경우가 나타납니다. 쿼리 프로세서가 인덱싱된 뷰를 사용하도록 하려면 NOEXPAND 힌트를 사용할 수 있습니다. 처음에 뷰를 명시적으로 참조하지 않은 경우에는 쿼리를 재작성해야 합니다. NOEXPAND를 통해 쿼리의 실제 비용을 계산하고 이를 뷰를 참조하지 않은 쿼리 계획의 실제 비용과 비교할 수 있습니다. 두 비용에 큰 차이가 없으면 인덱싱된 뷰 사용 여부를 결정하는 것이 그다지 중요하지 않다고 생각하면 됩니다.

(3) 쿼리 최적화 프로그램은 쿼리를 인덱싱된 뷰에 일치시키지 않습니다. 뷰의 정의와 쿼리의 정의를 재확인하여 두 정의가 구조적으로 일치하도록 합니다. CASTS, 변환 및 논리적으로 쿼리를 변경하지 않는 기타 표현식은 이러한 일치에 방해가 될 수 있습니다. 또한 SQL Server가 수행하는 표현식 정규화와 동치(equivalence) 및 포함(subsumption) 관계 테스트에 대한 제한이 있습니다. 몇몇 동치 표현식이 동일하거나 다른 표현식에 의해 논리적으로 포함된 표현식이 실제로 포함되어 있다는 것을 보여 줄 수 없기 때문에 일치에 실패할 수 있습니다.

Q. 일주일에 한 번 데이터 웨어하우스를 업데이트하고 있습니다. 인덱싱된 뷰 때문에 주간에는 쿼리 속도는 향상되었지만 업데이트 속도는 느려졌습니다. 어떻게 해야 합니까?

A. 주 단위의 업데이트에 앞서 인덱싱된 뷰를 삭제하고 이후 이를 다시 생성하십시오.

Q. 뷰 복제본이 있기는 하지만 실제로 이를 유지 관리하고 싶습니다. 어떻게 해야 합니까?

A. 원하는 뷰의 모든 열이나 표현식을 그룹화해서 COUNT_BIG(*) 열을 추가한 뷰를 생성한 다음, 그룹화 열에서 클러스터된 고유 인덱스를 생성하십시오. 그룹화 프로세스는 고유성을 가져야 합니다. 이는 실제 동일한 뷰가 아니지만 사용자의 요구를 충족할 수 있습니다.

Q. 다른 뷰 위에 정의된 뷰를 가지고 있습니다. SQL Server는 상위 레벨 뷰의 인덱싱을 지원하지 않을 것입니다. 어떻게 해야 합니까?

A. 중첩된 뷰에 대한 정의를 상위 레벨 뷰까지 직접 확장하십시오. 그런 다음, 이를 인덱싱하거나 가장 안쪽의 뷰를 인덱싱하거나 뷰를 인덱싱하지 않을 수 있습니다.

Q. 인덱싱된 뷰가 WITH SCHEMABINDING로 정의되어야 하는 이유는 무엇입니까?

A. 사용자의 뷰 액세스에 관계없이

- 뷰가 참조한 모든 객체를 schemaname.objectname을 사용해 명백하게 정의해야 하기 때문입니다.
- 또한 허용되지 않는 방식으로 뷰 정의를 수행하거나 SQL Server가 뷰에서 인덱스를 재생성하도록 강요하는 방법으로 뷰 정의에서 참조된 객체를 변경할 수는 없습니다.

Q. 인덱싱된 뷰에서 OUTER JOIN을 사용할 수 없는 이유는 무엇입니까?

A. 기본 테이블에 데이터를 삽입하면 OUTER JOIN에 따라 인덱싱된 뷰에서 행이 논리적으로 사라질 수 있습니다. 따라서, OUTER JOIN 뷰의 업데이트가 점차 복잡해지고 표준 (INNER) JOIN 기반의 뷰에 비해 업데이트 속도도 느려질 수 있습니다.

자세한 내용

Microsoft SQL Server 2005 Books Online에는 인덱싱된 뷰에 대한 자세한 정보가 포함되어 있습니다. 자세한 내용은 다음을 참조하십시오.

- Microsoft SQL Server 개발자 센터 http://msdn.microsoft.com/sql
- Microsoft SQL Server TechNet 사이트 http://www.microsoft.com/korea/technet/sql
- Microsoft SQL Server 웹사이트 http://www.microsoft.com/korea/sql