Microsoft®
SQL Server™ 2005

Microsoft SQL Server 2005 의 XML 지원

Microsoft®

요약

본 기술 자료에서는 Microsoft® SQL Server™ 2005에 내장된 XML 지원 기능을 개괄적으로 소개하고 이 제품이 .NET Framework V2.0 및 OLEDB나 SQLXML과 같은 원시 코드에서 클라이언트측 프로그래밍 지원과 어떻게 통합이 되는지 설명합니다.

Contents

들어가는 글

현재 널리 채택되고 있는 XML은 플랫폼에 의존하지 않는 데이터 표현 형식입니다. XML은 B2B 응용 프로그램이나 워크플로우 상황에서처럼 느슨하게 연결된 이기종 시스템 간에 정보를 교환하는데 유용합니다. 실제로 XML 기술을 발전시킨 주요 요인이 바로 데이터 상호 교환이었습니다.

XML은 현재 엔터프라이즈 응용 프로그램에서 반구조적 데이터와 비구조적 데이터를 모델링하는데 점점 더 많이 사용되고 있습니다. 이와 같은 응용 프로그램 중 하나가 바로 문서 관리입니다. 전자 메일과 같은 문서는 반구조적인 특성을 가지고 있는데, 문서를 데이터베이스 서버 안에 XML로 저장하게 되면 문서 내용을 근거로 문서를 검색하고 부분적인 내용에 대해 쿼리를 실행(예: 제목에 "Background"라는 단어가 들어 있는 항목을 찾는 것)하고 그 결과를 문서로 만드는 강력한 응용 프로그램이 개발해야 할 것입니다. XML을 생성하거나 사용하는 응용 프로그램이 증가하면서 그런 상황이 생길 가능성이 점점 많아지고 있습니다. 예를 들어, Microsoft® Office 2003 시스템에서는 Microsoft Word, Excel, Visio® 및 Infopath 문서를 XML 마크업 문서로 생성할 수 있습니다.

XML 데이터에 대해 관계형 데이터베이스를 사용하는 이유

XML 데이터를 관계형 데이터베이스에 저장하면 데이터 관리와 쿼리 처리에 도움이 됩니다. SQL Server는 관계형 데이터에 대해 강력한 쿼리 기능과 데이터 수정 기능을 제공하며 XML 데이터를 쿼리하고 수정하도록 기능이 확대되었습니다. 그렇기 때문에 구 버전에 대해 한 투자(예: 비용 기반 최적화나 데이터 스토리지 분야)를 활용할 수 있습니다. 예를 들어, 잘 알려져 있는 관계형 데이터베이스의 인덱싱 기법은 XML 데이터도 인덱싱하도록 확대되었기 때문에 비용 기준 의사결정을 사용하여 쿼리를 최적화할 수 있습니다.

XML 데이터는 기존의 관계형 데이터 및 SQL 응용 프로그램과 상호 호환이 되므로, 데이터 모델링이 필요하게 되면 기존의 응용 프로그램을 중단시키지 않고 XML을 시스템에 포함시킬 수 있습니다. 데이터베이스 서버는 XML 데이터를 관리하는 관리 기능(예: 백업, 복구 및 복제)도 제공합니다.

이러한 기능 때문에 점점 증가하는 XML 사용에 대처하기 위해 SQL Server™ 2005에 XML 지원 기능을 기본으로 포함시켜야 한다고 느끼게 되었습니다. SQL Server 2005에서 XML을 지원하게 되면 엔터프라이즈 응용 프로그램 개발에 도움이 될 것입니다.

이어지는 내용에서는 SQL Server™ 2000과 2005의 XML 지원을 개괄적으로 살펴보고 XML을 사용하게 만드는 몇 가지 시나리오를 설명하고 서버측 XML 기능 세트와 클라이언트측 XML 기능 세트를 상세히 설명할 것입니다. 그리고 제일 마지막 섹션에서 결론을 맺을 것입니다.

SQL Server 2000의 XML 지원

이 섹션에서는 관계형 데이터 모델과 XML 데이터 모델 사이의 데이터 매핑을 다양하게 지원하는 SQL Server 2000 및 그 이후에 나온 SQLXML 클라이언트측 프로그래밍 플랫폼 웹 릴리스들의 XML 지원을 간략하게 개괄적으로 설명합니다.

서버측 지원

서버의 경우 SELECT 문에서 FOR XML 구를 사용하여 테이블 및 쿼리 결과에서 XML 데이터를 생성합니다. 이 방법은 데이터 상호 교환 및 웹 서비스 응용 프로그램에서 이상적입니다. FOR XML을 뒤집어 놓은 것이 OpenXML이라고 하는 관계형 행 집합 생성자 함수입니다. 이 함수는 XPath 1.0 표현식을 평가하여 XML 데이터에서 값을 추출하여 행 집합 열에 넣습니다. OpenXML은 수신되는 XML 데이터를 테이블 안에 집어넣거나 Transact-SQL 언어를 사용하여 쿼리를 실행하는 응용 프로그램에서 사용합니다.

클라이언트측 지원

SQL Server 2000을 위한 클라이언트 프로그래밍 지원 기능은 SQLXML이라고 합니다. 이 기술의 핵심은 XML 뷰입니다. XML 뷰는 XML 스키마와 관계형 테이블의 양방향 매핑을 가리킵니다. SQL Server 2000은 XDR 스키마 매핑만 지원합니다. 물론 XSD 지원도 나중에 나온 두 개의 릴리스에서 추가되었습니다. XML 뷰에서는 XPath 1.0 서브셋을 사용하여 쿼리를 실행할 수 있습니다. 이 경우 매핑은 경로 표현식을 기반 테이블의 SQL 쿼리로 변환하는데 사용되며 쿼리 결과는 XML 결과에 함께 포함됩니다.

SQL XML에서는 동적 섹션이 있는 XML 문서를 만드는데 사용되는 XML 템플릿을 만들 수도 있으며, XML 문서 내에 FOR XML 쿼리와 XPath 1.0 표현식을 매핑 쿼리 위에 내장시킬 수도 있습니다. XML 템플릿을 실행하면 쿼리 블록이 쿼리 결과로 대치됩니다. 이런 식으로 정적인 콘텐트와 데이터를 이용하는 동적인 콘텐트가 함께 존재하는 XML 문서를 만들 수 있습니다.

SQL Server 2000에는 SQLXML 기능을 액세스하는 두 가지 주요 방법이 있습니다.

- SQLXMLOLEDB 공급자 - SQLXMLOLEDB 공급자는 ADO를 통하여 Microsoft SQLXML 기능을 표시하는 OLE DB 공급자입니다.
- HTTP 액세스 - SQL Server 2000의 SQLXML 기능은 SQLXML ISAPI 필터를 사용하여 HTTP를 통해 액세스할 수도 있습니다. Microsoft의 구성 툴을 사용하면 HTTP를 사용하는 XML 뷰를 통해 XML 템플릿, FOR XML 문 및 XPath 1.0 문을 실행하라는 요청을 수신하도록 웹 사이트를 설정할 수 있습니다.

XML 지원의 한계

서버와 클라이언트 프로그래밍 플랫폼은 테이블 형식과 XML 데이터 간의 매핑을 기초로 XML 데이터를 생성하고 사용할 수 있는 풍부한 기능을 제공하며 상당히 구조화된 XML 데이터를 매우 효과적으로 처리합니다. SQLXML에서 쿼리 언어는 XPath 1.0의 서브셋이며 몇 가지 제한 사항을 가지고 있습니다. 예를 들어, descendant axe (//-연산자)는 지원하지 않습니다. 따라서 특정한 솔루션을 개발하는데 제약이 있습니다. 예를 들어 XML 문서는 순서가 그대로 보존되지 않는데 이는 문서 관리와 같은 응용 프로그램에서 매우 중요한 것입니다. 게다가, 재귀 XML 스키마도 지원하지 않습니다. 이러한 한계에도 불구하고, 클라이언트 SQLXML과 서버 XML 기능은 응용 프로그램 개발에서 널리 사용되고 있습니다. SQL Server 2005는 이러한 많은 한계를 해결하고 관계형-XML 상호 교환 기능을 강화하였으며 XML을 기본 요소로 지원합니다.

SQL Server 2005의 XML 지원 개요

이 섹션에서는 SQL Server 2005에 추가된 새로운 XML 지원을 개괄적으로 간략하게 설명합니다. 이는 .NET Framework V2.0 지원과 OLE DB 등과 같은 원시 클라이언트 데이터 액세스 지원으로 보완되었습니다.

XML 데이터 형식

XML 데이터 모델에서는 관계형 데이터 모델과 매핑하는 것이 불가능하지는 않지만 그 특성상 매우 어렵습니다. XML 데이터는 재귀적인 계층 구조를 가지고 있으며 관계형 데이터베이스는 계층형 데이터를 제대로 지원하지 못합니다(외래 키 관계로 모델링함). 문서 순서는 XML 인스턴스의 근본적인 속성이며 쿼리 결과에서 그대로 보존되어야 합니다. 이것은 순서가 없기 때문에 순서 지정 열을 추가하여 순서를 강제로 정해야 하는 관계형 데이터와 대조가 됩니다. XML 데이터를 많은 수의 테이블로 분해하는 실제의 XML 스키마의 경우 쿼리를 실행하는 동안 결과를 다시 정렬하면 많은 비용이 발생합니다.

SQL Server 2005에서는 XML이라고 하는 원시 데이터 형식을 도입하였습니다. 사용자는 관계형 열 이외에 XML 형식의 열이 하나 이상 존재하는 테이블을 생성할 수 있으며, XML 변수와 매개 변수도 사용할 수 있습니다. 문서 순서나 재귀형 구조와 같은 XML 모델 특성을 보다 충실하게 지원할 수 있도록 XML 값은 내부 형식에 BLOB(large binary objects)로 저장됩니다.

SQL Server 2005는 W3C XML 스키마를 메타데이터로 관리하는 수단으로 XML 스키마 컬렉션을 제공합니다. XML 데이터 형식을 XML 스키마 컬렉션에 연결해 놓으면 XML 인스턴스에 스키마 제한 사항을 적용할 수 있습니다. XML 데이터가 XML 스키마 컬렉션과 연결되면 typed XML이라고 하고 그렇지 않으면 untyped XML이라고 합니다. type XML과 untyped XML은 모두 단일 프레임워크 내에 넣을 수 있으며, XML 데이터 모델은 그대로 보존됩니다. 그리고 쿼리 프로세싱 중에 XML 기능이 실행됩니다. 기반 관계형 인프라는 이런 목적으로 널리 사용됩니다. 또한 관계형 데이터와 XML 데이터 간의 호환성이 지원되며 이는 XML 기능을 더욱 널리 확산시키는데 도움이 되고 있습니다.

XML 데이터 형식 쿼리 및 데이터 수정

XML 인스턴스는 Transact-SQL SELECT 문을 사용하여 검색합니다. XML 데이터 형식에는 XML 인스턴스를 쿼리하고 수정하는 5가지 방식이 내장되어 있습니다.

XML 데이터 형식 메서드는 XQuery를 인식합니다. XQuery는 현재 모습을 드러내고 있는 W3C 표준 언어(현재 최종 단계임)이며 탐색 언어인 XPath 2.0을 포함하고 있습니다. 하위 트리를 추가하거나 삭제하고 스칼라 값을 업데이트하는 것과 같은 XML 데이터를 수정하는 언어도 사용할 수 있습니다. 많은 수의 함수 뿐만 아니라 내장된 XQuery와 데이터 수정 언어들을 사용하면 XML 데이터 처리를 다양하게 지원할 수 있습니다.

XQuery 유형 시스템은 W3C XML 스키마 유형의 시스템에 맞추어져 있습니다. 대부분의 SQL 유형은 XQuery 유형 시스템(예: 십진수)과 호환이 됩니다. 몇몇 유형(예: xs:duration)은 내부 형식으로 저장되며 XQuery 유형 시스템과 호환이 되도록 적절하게 변환이 됩니다.

컴파일 단계에서는 XQuery 표현식과 데이터 수정문의 정적 형식 정확성을 확인하고 typed XML인 경우에 형식을 추정하는 데 XML 스키마를 사용합니다. 런타임에 유형 안전성 위반으로 인해 표현식이 실패로 끝나면 정적 유형 오류가 발생합니다.

XML 인덱싱

쿼리를 실행하면 각 XML 인스턴스를 런타임에 처리합니다. 하지만 XML 값이 크거나 한 테이블의 많은 수의 행에서 쿼리를 평가하는 경우 이렇게 하면 비용이 많이 듭니다. 따라서, 쿼리 속도를 향상시킬 수 있도록 XML 열을 인덱싱하는 메커니즘이 마련되어 있습니다.

관계형 데이터를 인덱싱하는데 널리 사용되는 것이 B+tree입니다. XML 열의 주 XML 인덱스는 열에 있는 XML 인스턴스의 모든 태그, 값 및 경로에 B+tree 인덱스를 생성합니다. 이렇게 하면 XML 데이터에 대한 쿼리를 효율적으로 평가할 수 있으며 문서 순서와 문서 구조를 그대로 유지하면서 B+tree에서 XML 결과를 재구성할 수 있습니다.

다양한 종류의 흔히 발생하는 쿼리의 속도를 높이기 위하여 보조 XML 인덱스를 XML 열에 생성하기도 합니다. 즉, 경로 기반 쿼리의 경우 PATH 인데스를 생성하고 property bag 시나리오의 경우 PROPERTY 인덱스를 생성하고 값 기반 쿼리의 경우 VALUE 인덱스를 생성합니다.

XML 스키마 프로세싱

XML 스키마 컬렉션에 따라 XML 열, 변수 및 매개 변수를 선택적으로 입력할 수 있습니다. XML 스키마들은 서로 관련이 있을 수도 있고(예를 들면 〈xs:import〉를 사용함) 서로 관련이 없을 수도 있습니다. 각 typed XML 인스턴스는 정해진 XML 스키마 컬렉션에서 타겟 네임스페이스를 지정합니다. 데이터베이스 엔진은 데이터 할당 및 수정 중에 XML 스키마에 따라 인스턴스를 확인합니다.

XML 정보는 스토리지 및 쿼리 최적화에서 사용됩니다. typed XML 인스턴스의 내부 바이너리 표현 형태에는 XML 인덱스에서처럼 typed 값이 들어 있습니다. 따라서 typed XML 데이터를 효율적으로 처리할 수 있습니다.

관계형 데이터와 XML 데이터의 통합

관계형 데이터와 XML 데이터는 동일한 데이터베이스에 함께 저장할 수 있습니다. 간단히 말해서, 데이터베이스 엔진은 관계형 데이터 모델 뿐만 아니라 XML 데이터 모델을 올바르게 처리하는 방법을 알고 있습니다. 관계형 데이터와 SQL 응용 프로그램은 SQL Server 2005로 업그레이드해도 아무 문제 없이 작동합니다. 파일 및 텍스트나 이미지 열에 존재하는 XML 데이터는 서버의 XML 데이터 형식 열로 옮길 수 있습니다. XML 열은 XML 데이터 형식 메서드를 사용하여 인덱싱하고 쿼리를 실행하고 수정할 수 있습니다.

데이터베이스는 기존의 관계형 인프라를 활용하며 XML 프로세싱을 위한 스토리지 엔진, 쿼리 프로세서 등과 같은 엔진 구성 요소도 활용합니다. 예를 들어, XML 인덱스는 B+tree를 생성하며 쿼리 계획은 Showplan을 실행하여 볼 수 있습니다. 관계형 프레임워크에 통합되어 있는 백업/복구 및 복제와 같은 데이터 관리 기능도 XML 데이터에 대해 사용할 수 있습니다. 마찬가지로, 데이터베이스 미러링이나 스냅샷 격리와 같은 새로운 데이터 관리 기능들도 XML 데이터 형식에 대해 사용할 수 있으므로 자연스러운 사용자 환경이 형성됩니다.

구조적 데이터는 테이블 및 관계형 열에 저장해야 합니다. XML 데이터 형식은 응용 프로그램이 정밀한 쿼리와 데이터 수정을 수행해야 하는 경우 XML을 사용하는 반구조적인 마크업 데이터에 적합한 선택입니다.

FOR XML 및 OpenXML 보완 기능

기존의 FOR XML 기능은 여러 가지 방식으로 보강되었습니다. 즉, FOR XML 기능은 XML 데이터 형식 인스턴스 및 [n]varchar(max)와 같은 다른 새로운 SQL 유형에서 사용할 수 있습니다. 향상된 FOR XML 기능에 대한 자세한 내용은 "Microsoft SQL Server 2005에서 새로워진 FOR XML"을 참조하십시오.

새로운 TYPE 지시어를 사용하면 XML 열, 변수 또는 매개 변수에 할당하거나 XML 데이터 형식 메서드를 사용하여 쿼리할 수 있는 XML 데이터 형식 인스턴스가 생성됩니다. 이를 통해 SELECT … FOR XML TYPE 문의 중첩이 가능해집니다.

PATH 모드에서는 XML 트리에서 열의 값이 나타나야 하는 경로를 지정할 수 있습니다. 이것을 앞에서 언급한 중첩 기능과 함께 사용하면 FOR XML EXPLICIT보다 명령문을 만드는데 더 편리합니다.

ELEMETS와 함께 사용되는 지시어 XSINIL은 속성이 xsi:nil="true"인 요소에 NULL을 매핑합니다. 또한 새로운 ROOT 지시어를 사용하면 FOR XML의 모든 모드에서 루트 노드를 지정할 수 있습니다. 이 새 XMLSCHEMA 지시어는 XSD 인라인 스키마를 생성합니다. SQL Server 2005에서 FOR XML을 사용하면 FOR XML RAW 모드에서 기본 〈row〉를 대체할 요소 이름을 지정할 수 있습니다.

OpenXML의 보강된 기능은 sp_preparedocument에서 XML 데이터 형식을 사용할 수 있게 된 것과 rowset에서 XML 및 새로운 SQL 유형 열을 생성할 수 있게 된 것으로 구성됩니다.

클라이언트의 XML 데이터 형식 액세스

클라이언트가 서버의 XML 데이터를 액세스하는 방법은 여러 가지가 있습니다. ODBC나 OLE DB를 사용하는 원시 SQL 클라이언트 액세스는 XML 데이터를 유니코드 문자열 형식으로 전달합니다. OLE DB에서는 유니코드 데이터를 스트리밍할 수 있는 XML 데이터 형식에 대해 ISequentialStream 액세스를 할 수 있습니다.

.NET Framework V2.0에서 ADO.NET을 통한 관리된 액세스는 XML 데이터를 SqlXml라고 하는 새로운 클래스로 전달합니다. 여기서 지원하는 CreateReader( )라고 하는 메서드는 실행 결과 XML을 읽는 XmlReader 인스턴스를 만들어줍니다. 마찬가지로 DataSet는 XML 데이터 형식의 인스턴스를 미드티어의 열에 로드할 수 있으며, 이 인스턴스를 XML 문서로 편집하여 SQL Server에 다시 저장할 수 있습니다. 이 두 가지 방법 모두 SQL 쿼리를 서버로 보내어 미드티어에서 처리할 XML 열을 검색할 수 있습니다.

SQL Server 2005에서 HPPT 엔드포인트에 대한 직접적인 SOAP 액세스는 XML 데이터를 쿼리, 검색 및 수정하는데 사용할 수 있습니다.

원시 기술과 관리된 클라이언트 기술은 모두 XML 스키마 컬렉션을 검색하여 XML 열의 형식을 정하는 새 인터페이스를 제공합니다.

XML 스토리지의 흥미진진한 시나리오

XML 데이터가 점점 널리 퍼지고 있고 데이터를 규정하는 XML 스키마를 사용하든 않든 관계없이 XML 데이터가 고객 데이터가 될 것입니다. XML 데이터와 XML 스키마는 모두 함께 관리해야 합니다. 대체로 실제로 사용하는 응용 프로그램의 XML 스키마는 복잡합니다. 그런 XML 스키마를 테이블이나 열에 매핑하는 것은 복잡한 작업이며 시간이 경과하면서 XML 스키마가 바뀌거나 시스템에 새로운 스키마가 추가될 때 그런 매핑을 유지 관리하는 것은 성가신 일입니다. XML 데이터가 파일 시스템이나 데이터베이스 서버의 텍스트 열에 저장되는 경우도 자주 있습니다. 텍스트 열은 복제나 백업/복구와 같은 데이터 관리에 편리하지만 데이터의 XML 구조를 기준으로 하는 쿼리는 지원하지 않습니다. XML이 기본적으로 지원되면 XML을 사용하는 응용 프로그램 개발이 더 빨라질 것입니다.

전용 속성 관리

사용자 인터페이스 소프트웨어와 같은 일부 응용 프로그램에서는 사용자가 정해진 속성 중에서 선택할 수 있습니다. 하지만 사용자가 원하는 속성을 정의할 수 있는 응용 프로그램도 있습니다. 그런 전용 속성은 XML 형식으로 저장하면 관리하기 더 좋습니다. 응용 프로그램은 스칼라 속성 이상을 지원할 수 있습니다.

- 여러 개의 전화 번호와 같이 객체에서 값이 여러 개인 속성을 사용할 수 있습니다.
- 복잡한 속성도 지원합니다. 예를 들면 문서의 저작자 속성은 저작자의 연락처 정보가 될 수 있습니다.

객체 속성은 XML 데이터 형식 열에 저장할 수 있으며 인덱싱하여 쿼리를 효율적으로 처리하게 할 수 있습니다.

데이터 교환 및 워크플로우

XML에서는 플랫폼에 의존하지 않고 응용 프로그램 간에 데이터를 상호 교환할 수 있습니다. 그런 데이터는 XML 마크업을 사용하여 메시지로 모델링할 수 있습니다. XML 메시지를 끊임 없이 폐기하고 생성하는 것보다는 메시지를 XML 형식으로 저장하는 것이 현명합니다. 이것은 데이터 흐름 요건에 잘 맞습니다. 워크플로우 단계에 도달하는 XML 메시지는 현재 상태를 전달합니다. 각 메시지가 처리되면, 그 진행 상태를 XML 내용(예: 상태 변경)에 기록한 다음 워크플로우 프로세싱의 그 다음 단계로 XML 데이터를 넘깁니다. 메시지는 다양한 형식이며 반구조적인 메시지일 수도 있을 뿐만 아니라 다양한 XML 스키마가 연결되어 있기 때문에 메시지를 테이블에 매핑하는 것이 항상 쉬운 일을 아닙니다.

XML 기반 표준은 재무 데이터나 지리 정보 데이터와 같이 다양한 수직형 영역에서 모습을 드러내고 있습니다. 이런 표준들은 쿼리하고 업데이트할 수 있는 인스턴스 데이터를 기반으로 데이터 구조를 규정합니다. 실제 데이터는 바이너리 형식인데 XML 데이터는 그 데이터에 대한 메타데이터 정보를 제공하는 경우도 자주 있습니다.

간단한 예로, 입력 매개 변수 테이블을 저장된 프로시저나 함수에 전달하기 위하여, 응용 프로그램은 데이터를 XML로 변환한 다음 그것을 XML 데이터 형식 매개 변수로 전달합니다. 저장 프로시저나 함수 내에서 이 행 집합은 XML 매개변수로부터 다시 생성됩니다.

문서 관리

예를 들어 콜센터에서 환자 기록과 대화 내용을 XML 문서로 관리한다고 합시다. 환자의 전화를 받을 때 콜센터는 이전의 대화 내용을 보고 상황을 판단하려고 할 것입니다. XML 마크업을 쿼리하면 그렇게 할 수 있으며, 이것은 응용 프로그램에 도움이 됩니다. 게다가, 이전에 대화한 내용이 상세하면 현재 대화 내용을 기록하기 쉽습니다.

전자 메일과 같은 문서는 반구조적인 특성을 가지고 있습니다. XML 마크업이 있는 문서는 Microsoft Office 2003 등에서 만들기가 점점 더 쉬워지고 있습니다. 이러한 XML 문서는 XML 열에 저장할 수 있으며, 인덱싱, 쿼리 및 업데이트도 가능합니다. 따라서, XML이 기본적으로 지원되면 개발자들이 더 많은 일을 할 수 있습니다.

SQL Server 2005의 서버측 XML 프로세싱

SQL Server 2005 지원은 관계형 데이터와 XML 데이터를 저장하는 한 데이터베이스를 제공하는 것으로 이루어집니다.

XML 데이터 형식

일반적인 CREATE TABLE 문을 사용하면 XML 열이 있는 테이블을 만들 수 있습니다. 그 다음에 XML을 특별한 방식으로 인덱싱할 수 있습니다.

Untyped XML

SQL Server 2005 XML 데이터 형식은 ISO SQL-2003 표준 XML 데이터 형식을 구현합니다. 그렇기 때문에, 잘 구성된 XML 1.0 문서 뿐만 아니라 텍스트 노드와 여러 개의 최상위 요소들이 존재하는 소위 말하는 XML 콘텐트 프래그먼트들도 저장할 수 있습니다. 데이터가 제대로 구성되어 있는지 검사를 하고 잘 구성되지 않은 데이터는 거부하지만, XML 데이터 형식이 XML 스키마와 반드시 연결되어 있어야 하는 것은 아닙니다.

untyped XML은 미리 스키마를 알지 못하여 매핑 기반 솔루션이 가능하지 않은 경우에 유용합니다. 스키마를 알고 있기는 하지만 관계형 데이터 모델에 매핑하는 것이 매우 복잡하고 관리하기 어려운 경우, 또는 최근에 외부 요구 사항을 근거로 여러 스키마를 데이터에 연결시킨 경우에도 untyped XML이 도움이 됩니다.

예제: 테이블에 포함된 untyped XML 열

다음 명령문을 실행하면 정수 기본키는 "pk" 이고 untyped XML 열은 "xCol" 인 "docs" 라고 하는 테이블이 만들어집니다.

```
CREATE TABLE docs (pk INT PRIMARY KEY, xCol XML not null)
```

기본키 존재에 관계없이 XML 열이나 관계형 열이 하나 이상 존재하는 테이블을 생성할 수도 있습니다.

Typed XML

XML 스키마 컬렉션에 XML 데이터를 규정하는 XML 스키마가 있으면, 그 XML 스키마 컬렉션을 XML 열에 연결시켜 type XML이 되게 할 수 있습니다. XML 스키마는 데이터를 검증하고 쿼리 컴파일이나 데이터 수정 명령문 중에서 untyped XML보다 정밀하게 형식 검사를 수행하고 스토리지와 쿼리 프로세싱을 최적화하는데 사용됩니다.

typed XML 열, 매개 변수 및 변수는 XML 문서나 콘텐트를 저장할 수 있습니다. 문서인지 콘텐트인지는 선언할 때 옵션(DOCUMENT나 CONTENT, CONTENT가 기본값임)으로 지정할 수 있습니다. 또한, XML 스키마 컬렉션을 제공해야 합니다. 각 XML 인스턴스에 최상위 요소가 정확하게 하나이면 DOCUMENT를 지정하고 그렇지 않으면 CONTENT를 사용하십시오. 쿼리 컴파일러는 형식 검사에서 DOCUMENT 플래그를 사용하여 최상위 요소가 하나임을 알려줍니다.

예제: 테이블의 typed XML 열

XML 열, 변수 및 매개 변수는 XML 스키마 컬렉션에 연결시킬 수 있습니다(자세한 내용과 예제는 본 백서 뒷 부분의 "XML 스키마 처리" 섹션 참조). 그런 컬렉션 중 하나의 이름이 myCollection이라고 합시다. 아래 명령문을 실행하면 XmlCatalog 테이블이 생성되고 XML 열 문서는 myCollection을 사용하여 형식이 설정됩니다. typed XML 열은 XML 문서만이 아니라 XML 프래그먼트도 받아들이도록 지정됩니다.

```
CREATE TABLE xmltab (
        id INT IDENTITY PRIMARY KEY,
        Document XML(CONTENT myCollection))
```

XML 데이터 형식 열 제한

XML 열의 형식을 설정하는 것뿐만 아니라 typed XML과 untyped XML 데이터 형식 열에서 관계형 (열이나 행) 제한 규칙을 사용할 수 있습니다. 대부분의 SQL 제한 조건은 XML 열에도 적용되지만, 잘 알려진 예외 중에는 고유 키, 기본 키 및 외래 키 제한 조건입니다. XML 데이터 형식 인스턴스는 비교할 수 없기 때문입니다. 그렇기 때문에, XML 열을 null 사용 가능이나 null 사용 불가능으로 지정할 수 있으며, 기본값을 지정하고 열에 대해 CHECK 제한 조건을 정의할 수도 있습니다. 예를 들어, untyped XML 열에 CHECK 제한 조건을 지정하여 저장된 XML 인스턴스가 XML 스키마에 일치한지 확인하게 할 수 있습니다.

다음과 같은 경우 제한 조건을 사용하십시오.

- 비즈니스 규칙을 XML 스키마로 표현할 수 없는 경우입니다. 예를 들어, 꽃집의 배달 주소는 사업장 위치에서 50 마일 이내이어야 하는 경우, XML 열에 제한 조건으로 지정할 수 있습니다. 제한 조건에는 XML 데이터 형식 메서드가 관련될 수 있습니다.
- 또는 테이블의 다른 XML 열이나 비 XML 열이 관련될 수도 있습니다. 예를 들어 XML 인스턴스에 나오는 고객 ID(/Customer/@CustId)를 정수 CustomerID 열의 값에 일치시킬 수 있습니다.

예제: XML 열 제한하기

〈book〉의 〈author〉의 〈last-name〉이 〈author〉의 〈first-name〉과 다르게 하려면, 다음 CHECK 제한 조건 CK_name을 지정합니다. XML 데이터 형식 메서드는 사용자 지정 함수 안에 넣어야 합니다. 이렇게 하도록 udf_Check_Names( )이 사용됩니다.

```
CREATE FUNCTION udf_Check_Names (@xmlData XML)
RETURNS int AS
BEGIN
RETURN (SELECT @xmlData.exist( '/book/author[first-name = last-name]' ))
END
GO

 CREATE TABLE docs (pk INT PRIMARY KEY,
xCol XML not null
CONSTRAINT CK_name CHECK (udf_Check_Names(xCol) = 0))
GO
```

텍스트 인코딩

SQL Server 2005는 XML 데이터를 유니코드(UTF-16)로 저장합니다. 서버에서 검색된 XML 데이터도 UTF-16 인코딩이 되어 나옵니다. 다른 방식의 인코딩을 원한다면 캐스팅을 하거나 미드티어에서 데이터를 검색한 후에 필요한 변환을 해야 합니다. 예를 들어, 서버에서 XML 데이터를 varchar 형식으로 캐스팅할 수 있습니다. 이런 경우 데이터베이스 엔진은 XML을 varchar를 비교하여 결정된 인코딩과 연속으로 내보냅니다.

XML 데이터 저장하기

XML 열, 매개 변수 또는 변수에 대해 XML 값을 공급하는 방법은 여러 가지입니다.

- XML 데이터 형식으로 함축적으로 변환된 문자 또는 바이너리 SQL 형식으로 공급합니다.
- 파일의 내용으로 공급합니다.
- XML 데이터 형식 인스턴스를 생성하는 type 지시어를 사용한 XML 게시 메커니즘 FOR XML의 실행 결과로 공급합니다.

공급된 값은 적절하게 구성되었는지 확인을 거치며 XML 문서와 XML 프래그먼트를 모두 저장할 수 있습니다. 데이터가 구성 검사를 통과하지 못하면, 거부되며 적절한 오류 메시지가 나옵니다.

typed XML의 경우, 공급된 값은 XML 열의 형식을 정하는 XML 스키마 컬렉션에 등록된 XML 스카마에 일치한지 검사합니다. 이 검사를 통과하지 못하면 XML 인스턴스를 거부합니다. 더 나아가 typed XML의 DOCUMENT 플래그는 XML 문서에 승인된 값을 제한하는 반면, CONTENT의 경우 XML 문서와 콘텐트를 모두 공급할 수 있습니다.

예제: untyped XML 열에 데이터 삽입하기

다음 명령문은 새 행을 docs 테이블에 삽입하고 정수열 pk의 값은 1로 정하고 XML에는 〈book〉 인스턴스를 넣습니다. 문자열로 공급된 〈book〉 데이터는 함축적으로 XML 데이터 형식으로 변환되어 삽입하는 동안 제대로 구성되었는지 검사를 거칩니다.

```
INSERT INTO docs VALUES (1,
 '〈book genre=" security"  publicationdate=" 2002"  ISBN=" 0-7356-1588-2" 〉
        〈title〉 Writing Secure Code 〈/title〉
        〈author〉
                〈first-name〉 Michael 〈/first-name〉
                〈last-name〉 Howard 〈/last-name〉
        〈/author〉
        〈author〉
                〈first-name〉 David 〈/first-name〉
                〈last-name〉 LeBlanc 〈/last-name〉
        〈/author〉
        〈price〉 39.99 〈/price〉
〈/book〉
INSERT INTO docs VALUES (2,
 '〈doc id=" 123" 〉
    〈sections〉
        〈section num=" 1" 〉 〈title〉 XML Schema 〈/title〉 〈/section〉
        〈section num=" 3" 〉 〈title〉 Benefits 〈/title〉 〈/section〉
        〈section num=" 4" 〉 〈title〉 Features 〈/title〉 〈/section〉
    〈/sections〉
〈/doc〉' )
```

예제: 파일에서 untyped XML 열에 데이터 삽입하기

아래 나오는 INSERT 문은 OPENROWSET을 사용하여 C:\temp\xmlfile.xml 파일의 내용을 BLOB로 읽습니다. docs 테이블에 새 행이 삽입되고 기본키의 값은 10, XML 열 xCol의 값은 BLOB로 지정됩니다. 파일 내용이 XML 열에 할당될 때 구성이 제대로 되었는지 검사합니다.

```
INSERT INTO docs
SELECT 10, xCol
FROM            SELECT * FROM EmpCTE
                (BULK  'C:\temp\xmlfile.xml' ,
                SINGLE_BLOB) AS xCol) AS R(xCol)
```

예제: typed XML 열에 데이터 삽입하기

typed XML 열에서는 XML 인스턴스 데이터가 형식을 지정하는데 사용된 XML 스키마의 타겟 네임스페이스를 지정해야 합니다 (네임스페이스는 비어 있을 수 있음). 아래 예에서처럼, 네임스페이스 선언문 xmlns=http://myDVD을 사용하면 그렇게 할 수 있습니다.

```
INSERT XmlCatalog VALUES(2,
 '<?xml version="1.0"?>
<dvdstore xmlns="http://myDVD">
  <dvd genre="Comedy" releasedate="2003">
    <title>My Big Fat Greek Wedding</title>
    <price>39.99</price>
  </dvd>
</dvdstore>' )
```

예제: TYPE 지시어로 FOR XML을 사용하여 생성한 XML 데이터 저장하기

실행 결과를 XML 데이터 형식 인스턴스로 생성하도록 TYPE 지시어로 FOR XML을 보강하였습니다. 그 결과 만들어진 XML은 XML 열, 변수 또는 매개 변수에 할당할 수 있습니다. 다음 명령문에서, FOR XML TYPE을 사용하여 생성한 XML 인스턴스는 XML 데이터 형식 변수 @xVar에 할당이 됩니다. 이 변수는 XML 데이터 형식 메서드를 사용하여 쿼리할 수 있습니다.

```
DECLARE @xVar XML
SET    @xVar = (SELECT * FROM docs FOR XML AUTO,TYPE)
```

스토리지 표현 형태

XML 데이터 형식 인스턴스는 스트리밍이 가능하고 효율적으로 분석할 수 있도록 최적화된 내부의 바이너리 표현 형태로 저장됩니다. 태그는 정수값으로 매핑되고 매핑된 값은 내부 표현 형태로 저장됩니다. 이렇게 하면 데이터가 약간 압축되는 효과도 있습니다.

untyped XML의 경우, 노드 값은 유니코드(UTF-16) 문자열로 저장되므로 동작을 수행하려면 타임 유형 변환을 해야 합니다. 예를 들어, /book/price 〉 9.99를 평가할 수 있도록, book의 가격 값이 10진수로 변환됩니다. 반면, typed XML의 경우, 값은 XML 스키마에 지정된 형식으로 인코딩됩니다. 이렇게 하면 데이터 분석이 훨씬 더 효율적이 되며 런타임 변환을 하지 않아도 됩니다.

저장된 바이너리 형태는 XML 인스턴스 당 2GB로 제한되지만, 이 정도이면 대부분의 XML 데이터를 수용할 수 있습니다. 게다가, XML 계층 구조의 깊이는 128 레벨로 제한됩니다.

XML 데이터의 InforSet 콘텐트는 그대로 보존됩니다(InfoSet에 대한 자세한 내용은 http://www.w3.org/TR/xml-infoset 참조). 하지만 의미 없는 빈칸, 속성 순서, 네임스페이스 접두어 및 XML 선언 등의 정보가 그대로 보존되지 않기 때문에 텍스트 XML과 똑같은 복사본은 아닐 수도 있습니다.

데이터 모델링 관련 고려 사항

대체로, 관계형 데이터 형식 열과 XML 데이터 형식 열을 함께 사용하는 것이 데이터 모델링에 적합합니다. XML 데이터에서 나온 값의 일부는 관계형 열에 저장하고 나머지 또는 전체 XML 값은 XML 열에 저장할 수 있습니다. 이렇게 하면 성능과 잠금 특성이 더 향상됩니다.

XML 데이터 내의 값은 단일 값(즉, 단일 값 속성)의 동일한 테이블에서 계산된 열로 넘길 수 있습니다. 다중 값 속성의 경우 해당 속성에 대해 별도의 테이블을 사용해야 하며, 이 테이블의 내용은 트리거를 사용하여 입력하고 관리해야 합니다. 쿼리는 속성 테이블에 직접 기록해야 합니다.

XML 열에 저장된 XML 데이터의 세분화는 잠금 및 업데이트 특성에서 매우 중요합니다. SQL Server는 XML 데이터와 비 XML 데이터 모두에 대해 동일한 잠금 메커니즘을 사용합니다. 세분화가 큰 경우 업데이트를 위해 큰 XML 인스턴스를 잠그면 다중 사용자 환경에서 처리량이 줄어들게 됩니다. 반면에, 심하게 분해하면 객체 캡슐화가 사라져 재구성 비용이 증가하게 됩니다.

XML 데이터 쿼리 및 수정

XML 열에 저장된 XML 인스턴스를 쿼리하려면 XML 열에 있는 바이너리 XML 데이터를 분석해야 합니다. 바이너리 XML을 분석하는 것은 XML 데이터의 텍스트 형태를 분석하는 것보다 훨씬 빠릅니다. XML 인덱싱을 하면 재분석하지 않아도 됩니다. 이 점은 "XML 데이터 인덱싱" 섹션에서 설명합니다.

XML 데이터 형식의 메서드

원하는 경우 전체 XML 값을 검색할 수도 있고 XML 인스턴스의 일부를 검색할 수도 있습니다. 사용할 수 있는 네 가지 XML 데이터 형식 메서드는 **query**( ), **value**( ), **exist**( ) 및 **nodes**( )입니다. 이 메서드들은 XQuery 표현식을 인수로 사용합니다. 다섯 번째 메서드인 **modify**( )은 XML 데이터를 수정하는데 사용되며 XML 데이터 수정 명령문을 입력으로 받아들입니다.

query( ) 메서드는 XML 인스턴스의 각 부분을 추출하는데 유용합니다. XQuery 표현식은 XML 노드 목록을 평가합니다. 각 노드에서 나온 하위 트리가 문서 순서로 되돌아옵니다. 결과 형식은 untyped XML입니다.

value( ) 메서드는 XML 인스턴스에서 스칼라 값을 추출합니다. 이 메서드는 XQuery 표현식이 평가할 노드의 값을 알려줍니다. 이 값은 **value**( ) 메서드의 두 번째 인수로 지정된 Transact-SQL 형식으로 변환됩니다.

exist( ) 메서드는 XML 인스턴스에서 존재 검사를 하는데 유용합니다. XQuery 표현식이 비 null 노드 목록을 평가하는 경우 실행 결과는 1이고 그렇지 않으면 0입니다.

nodes( ) 메서드는 특수 XML 데이터 형식의 인스턴스를 생성하며, 각 인스턴스의 환경은 XQuery 표현식이 평가하는 다른 노드로 설정됩니다. 특수 XML 데이터 형식은 **query**( ), **value**( ), **nodes**( ) 및 **exist**( ) 메서드를 지원하며, **count**(*) 집계와 NULL 검사에서 사용할 수 있습니다. 그 외의 다른 경우에는 오류가 발생합니다.

modify( ) 메서드에서는 XML 인스턴스의 각 부분을 수정할 수 있습니다. 예를 들면, 하위 트리를 추가하거나 삭제하는 것, 또는 9.99에서 39.99까지 정해진 책의 가격과 같은 스칼라 값을 대치하는 것이 있습니다.

예제: query( ) 메서드 사용

docs 테이블의 XML 열 xCol에서 id가 123인 〈doc〉 요소 밑에서 〈section〉 요소를 추출하는 다음 쿼리를 고려해 봅시다. 이 쿼리는 정수 기본키 열에서도 값을 검색합니다. SELECT 목록의 **query**( ) 메서드는 테이블의 각 행에 대해 평가를 하여 일련의 〈section〉 요소를 생성합니다. 이 요소들은 해당 하위 트리와 함께 문서 순서로 검색됩니다. id가 123인 〈doc〉 요소가 없거나 그 아래에 〈section〉 요소가 없는 각 XML 인스턴스는 아무 결과도 돌려주지 않습니다. 즉, **query**( ) 메서드의 실행 결과 값은 empty XML입니다.

```
SELECT pk, xCol.query( '/doc[@id = 123]//section' )
FROM   docs
```

empty XML 값은 외부의 SELECT 문에서 필터링할 수 있습니다. 또는, 다음 예제에 나오는 것처럼 **exist**( ) 메서드를 사용할 수 있습니다.

예제: exist( ) 메서드

아래 쿼리에 docs 테이블의 XML 열 xCol에 있는 **query**( ) 메서드와 **exist**( ) 메서드가 관련되어 있다고 합시다. **exist**( ) 메서드는 경로 표현식 /doc[@id = 123] 을 평가하여 값이 123인 id라고 하는 속성이 있는 최상위 〈doc〉 요소가 존재하는지 확인합니다. 그런 각 행에 대해, SELECT 구의 **query**( ) 메서드를 평가합니다. 이 예제에서, **query**( ) 메서드를 실행하면 〈doc〉 요소 밑에 있는 〈section〉 요소가 나옵니다. **exist**( ) 메서드를 실행하여 결과가 0이 나오는 모든 행은 건너뜁니다.

```
SELECT xCol.query( '/doc[@id = 123]//section' )
FROM   docs
WHERE  xCol.exist ( '/doc[@id = 123]' ) = 1
```

예제: value( ) 메서드

다음 쿼리는 **value**( ) 메서드를 사용하여 문서의 세 번째 섹션의 제목을 유니코드 문자열로 추출합니다. 실행 결과의 SQL 형식 nvarchar(max)은 **value**( ) 메서드의 두 번째 인수로 지정됩니다. XQuery 함수 **data**( )는 〈title〉 노드에서 스칼라 값을 추출합니다.

```
SELECT xCol.value(
    'data((/doc//section[@num = 3]/title)[1])' , 'nvarchar(max)' )
FROM            docs
```

예제: XML 데이터 형식 메서드에서 GROUP BY 사용하기

XML 데이터 형식 메서드는 Transact-SQL GROUP BY 구에서 허용되지 않습니다. 하지만, 하위 쿼리의 XML 열에서 값을 추출하고 그룹 열의 별칭을 지정하고 그 별칭을 GROUP BY 구에서 사용할 수 있습니다. 다음 쿼리는 이름이 동일한 저작자가 출간한 책의 수를 계산하여 이것을 보여줍니다.

```
SELECT Fname, count(Fname)
FROM
  SELECT nref.value( 'first-name[1]' ,  'nvarchar(50)' ) FirstName,
   FROM   docs CROSS APPLY xCol.nodes( '/book' ) T(nref)
   WHERE  nref.exist( '.[first-name != "David"]' ) = 1
GROUP BY FName
ORDER BY FName
```

예제: sqlcmd 실행

XQuery 및 XML 데이터 수정 명령문을 실행하려면 연결 옵션 QUOTED_IDENTIFIER가 ON이어야 합니다. SQLCMD에서 이 옵션의 기본 값은 OFF입니다. 이 값을 -I 스위치를 사용하여 ON으로 변경해야 합니다.

```
sqlcmd -E -I -d 〈database〉 -Q "SELECT xCol.query( '//author' ) FROM docs"
```

예제: cast( ) 메서드와 value( ) 메서드 사이의 동작 방식 차이

query( ) 메서드는 XML 데이터 형식 인스턴스를 돌려주며, 특수 XML 문자는 문자열 형식으로 변환될 때 엔티티가 됩니다. 반면에 value( ) 메서드는 SQL 형식 값을 돌려주며, 특수 문자를 엔티티로 만들지 않습니다. 이 차이는 이벤트 통지에서 분명하게 나타납니다. 물론 아래의 첫 번째 쿼리에는 엔티티화된 캐리지 리턴이 포함되어 있을 수 있습니다.

```
SELECT EVENTDATA( ).value( '
(/EVENT_INSTANCE/TSQLCommand/CommandText/text( ))[1]' ,' nvarchar(max)' )
```

두 번째 쿼리는 그렇지 않습니다.

```
SELECT CAST (EVENTDATA( ).query( '
 /EVENT_INSTANCE/TSQLCommand/CommandText/text( )' ) AS nvarchar(max))
```

XQuery 언어

파일 시스템, 웹 서비스 또는 구성 파일 등에 저장된 Office 문서에서 XML이 생성되는 곳은 무수하게 많습니다. 사실, XML 형식이든 가상적인 XML 문서이든 간에 생성되는 데이터는 점점 증가하고 있습니다 이처럼 증가하는 데이터에 대처할 수 있도록 강력한 쿼리 언어인 XQuery가 만들어졌습니다 XQuery를 만들게 된 이유는 http://www.w3.org/TR/xquery에 나오는 다음과 같은 XQuery 언어 규격에 설명되어 있습니다.

- XML 구조를 사용하는 쿼리 언어는 실제로 XML로 저장된 것이든 미들웨어를 통하여 XML로 표시되는 것이든 간에 모든 종류의 데이터에서 쿼리를 지능적으로 표현할 수 있습니다. 이 규격에서는 많은 종류의 XML 데이터 소스에서 폭넓게 적용할 수 있도록 만들어진 XQuery라고 하는 쿼리 언어에 대해 설명합니다.
- XQuery는 W3C XML Query Working Group XML Query 1.0 요구 사항에서 밝힌 내용 및 XML Query Use Cases의 사용 예에 맞게 만들어졌습니다. 이것은 쿼리를 간결하게 만들고 쉽게 이해할 수 있는 언어가 되도록 설계되었습니다. 이 언어는 데이터베이스와 문서를 모두 포함하여 매우 방대한 XML 정보 소스를 충분히 쿼리할 수 있는 유연성을 갖추고 있습니다.

• XQuery를 요약하면 다음과 같이 표현할 수 있습니다. XQuery 언어와 XML의 관계는 SQL 언어와 관계형 데이터베이스의 관계와 같습니다.

Transact-SQL에 내장된 XQuery 서브셋(http://www.w3.org/TR/xquery/)은 XML 데이터 유형을 쿼리하는 것을 지원하는 언어입니다. 이 언어는 현재 W3C(World-wide Web Consortium)에서 개발 중(최종 단계)이며 Microsoft를 포함한 모든 주요 데이터베이스 업체들이 참여하고 있습니다. 본사의 구현 형태는 2004년 7월 버전 XQuery에 맞추어진 것입니다.

XQuery에는 탐색 언어인 XPath 2.0가 포함되어 있습니다. SQL Server 2005에서 구현한 XQuery는 노드에 대한 반복 동작(for), 노드 검사(where), 반환값(return) 및 정렬(order by)을 위한 구문을 제공합니다. 또한 쿼리 중에 데이터를 재구성할 수 있는 요소 구성도 제공합니다.

SQL Server 2005은 XML 데이터 형식의 데이터 수정(DML)을 위한 언어 구조도 제공합니다(자세한 내용은 본 문서 뒷 부분의 "데이터 수정" 섹션 참조). 다음 예제는 XML 데이터 형식에서 XQuery를 사용하는 방법을 보여줍니다.

예제: XQuery에서 다양한 언어 구문 사용하기
아래 쿼리는 함께 사용되는 다양한 XQuery 언어 구문을 보여줍니다. 이 쿼리를 실행하면 id가 123인 문서에서 섹션 번호가 3 이상인 섹션들의 제목을 새로운 태그인 〈topic〉 안에 넣어서 표시합니다.

```
SELECT pk, xCol.query( '
        for $s in /doc[@id = 123]//section
        where $s/@num 〉= 3
        return 〈topic〉{data($s/title)}〈/topic〉' )
FROM   docs
```

"for"는 id가 123인 〈docs〉 요소의 모든 〈section〉 요소를 반복하여 찾아내어 각 〈section〉을 변수 $s에 연결시킵니다. "where"는 섹션 번호(〈section〉 요소의 @num 속성)가 3 이상이 되게 합니다. 쿼리는 〈topic〉이라고 하는 구문 요소에 들어 있는 문서 순서대로 섹션 〈title〉의 값을 돌려줍니다.

쿼리 컴파일 및 실행

SQL 명령문은 SQL 구문 분석기로 분석합니다 SQL 구문 분석기는 XQuery 표현식이 나오면 XQuery 컴파일러로 전환하여 그 XQuery 표현식을 컴파일합니다. 이렇게 하면 전체 쿼리의 트리에 연결된 쿼리 트리가 생깁니다.
전체 쿼리 트리는 쿼리 최적화를 거친 다음 물리적인 쿼리 계획을 생성합니다. 이 쿼리 계획은 비용 기준 추정에 근거하여 선택된 것입니다. Showplan 실행 결과에는 대체로 관계형 연산자가 나오며 XML 프로세싱을 위한 UDX와 같은 몇 가지 새로운 연산자도 나옵니다.

쿼리 실행은 관계형 프레임워크의 나머지 부분에서처럼 튜플 중심적입니다. WHERE 구는 docs 테이블의 각 행에서 평가되며, 그 중에는 런타임에 XML blob를 분석하여 XML 데이터 형식 메서드를 평가하는 것이 포함됩니다. 조건이 일치하면, 해당 행을 잠그고 그 행에서 SELECT 구를 평가합니다. 실행 결과는 query( ) 메서드에 맞는 XML 데이터 형식으로 생성되어 value( ) 메서드의 지정된 타겟 형식으로 변환됩니다.

반면에 WHERE 구에서 행이 조건에 일치하지 않으면, 그 행은 건너뛰고 그 다음 행으로 넘어갑니다.

XML 데이터 수정

SQL Server 2005에는 XQuery를 보완한 데이터 수정 구문이 있습니다. 하위 트리는 지정된 노드 앞이나 뒤에 삽입할 수도 있고, 제일 왼쪽 자식 노드나 제일 오른쪽 자식 노드로 삽입할 수도 있습니다. 게다가, 하위 트리를 부모 노드에 삽입할 수 있습니다. 이 경우 하위 트리는 부모 노드의 제일 오른쪽 자식 노드가 됩니다. 속성, 요소 및 텍스트 노드 삽입도 모두 지원합니다. 하위 트리 삭제도 지원합니다. 이 경우, 전체 하위 트리를 XML 인스턴스에서 삭제합니다.
스칼라 값은 새 스칼라 값으로 대치할 수 있습니다.

예제: XML 인스턴스에 하위 트리 삽입

다음 예제는 **modify**( ) 메서드를 사용하여 새 〈section〉 요소를 번호가 1인 〈section〉 요소 오른쪽에 삽입하는 것을 보여줍니다.

```
UPDATE docs SET xCol.modify( '
  insert
        〈section num=" 2" 〉
〈title〉Background〈/title〉
〈/section〉
  after (/doc//section[@num=1])[1]' )
```

예제: 책의 가격을 $49,99로 업데이트

다음 업데이트 명령문은 ISBN이 1-8610-0311-0인 책의 〈price〉를 $49.99로 대치합니다. XML 인스턴스의 형식은 XML 스키마 http://myBooks, 즉 XML 데이터 수정 명령문의 네임스페이스 선언에 따라 설정됩니다.

```
UPDATE XmlCatalog
SET   Document.modify ( '
        declare namespace bk =  "http://myBooks" ;
        replace value of (/bk:bookstore/bk:book
[@ISBN=" 1-861003-11-0" ]/bk:price)[1] with 49.99' )
```

형식 검사 및 정적 오류

XQuery에서는 형식 검사가 새로 도입되었습니다. 컴파일 단계에서는 XQuery 표현식과 데이터 수정문의 정적 형식 정확성을 확인하고 typed XML인 경우에 형식을 추정하는 데 XML 스키마를 사용합니다. 그렇기 때문에 형식 안전성 위반으로 런타임에 표현식이 실패하면 정적 형식 오류가 발생합니다. 정적 오류의 예를 들면 정수에 문자열을 추가하는 것, 값이 하나이어야 하는데 여러 값이 나오는 것, 그리고 형식이 설정된 데이터에 대해 존재하지 않는 노드를 쿼리하는 것 등이 있습니다. 형식 불일치로 발생하는 정적 오류를 해결하는 방법은 적절한 형식으로 명시적으로 캐스팅하는 것입니다. XQuery 런타임 오류는 빈 시퀀스로 변환됩니다.

단일 값을 요구하는 로케이션 스텝, 함수 매개 변수 및 연산자(예: eq)는 컴파일러가 런타임에 단일 값이 보장되는지 여부를 판단할 수 없는 경우 오류를 돌려줍니다. 이 문제는 형식이 지정되지 않은 데이터에서도 종종 발생합니다. 예를 들어 속성 찾아보기에서는 단일 부모 요소가 필요하므로 단일 부모 노드를 선택하는 서수가 적합합니다.

예제: value( ) 메서드의 형식 검사

untyped XML 열에 대한 아래 쿼리의 경우 //author/last-name에서 서수를 지정해야 합니다. value( ) 메서드가 첫 번째 인수로 단일 노드를 기대하기 때문입니다. 이것이 없으면 컴파일러는 런타임에 하나의 〈last-name〉 노드만 생길 것인지 아닌지 판단할 수 없습니다.

```
nref.value( 'last-name[1]' , 'nvarchar(50)' ) LastName
FROM   docs
```

node( )-value( ) 구성을 평가하여 속성 값을 추출하는 경우 다음 예제에 나오는 것처럼 서수를 지정할 필요가 없을 수도 있습니다.

예제: 알려진 단일 값

아래 나오는 nodes( ) 메서드는 각 〈book〉 요소마다 개별적인 행을 생성합니다. 〈book〉 노드에 대해 평가한 value( ) 메서드는 @genre의 값을 추출합니다. 이 값은 속성이므로 단일 값입니다.

```
SELECT nref.value( '@genre' , 'varchar(max)' ) LastName
FROM   docs CROSS APPLY xCol.nodes( '//book' ) AS R(nref)
```

교차 영역 쿼리

데이터가 관계형 데이터 열과 XML 데이터 형식 열로 구성된 곳에 들어 있는 경우, 관계형 데이터 프로세싱과 XML 데이터 프로세싱을 결합한 쿼리를 만들어야 할 것입니다. FOR XML를 TYPE 지시어와 함께 사용하여 관계형 열과 XML 열의 데이터를 XML 데이터 형식 인스턴스로 변환한 다음 XQuery를 사용하여 쿼리할 수 있습니다. 또는, XML 값에서 행 집합을 생성한 다음 Transact-SQL을 사용하여 쿼리할 수도 있습니다.

교차 영역 쿼리를 만드는 보다 편리하고 효율적인 방법은 XQuery 또는 XML 데이터 수정 표현식 내에서 SQL 변수나 열의 값을 사용하는 것입니다.

- sql:variable( )을 사용하여 XQuery나 XML DML 표현식에 SQL 변수의 값을 적용하십시오.
- sql:column( )에서 XQuery나 XML DML 문맥의 관계형 열에서 나온 값을 사용하십시오.

이 방식을 사용하면 다음 예제에 나오는 것처럼 응용 프로그램이 쿼리의 매개 변수를 지정할 수 있습니다. Sql:column( )은 비슷한 방식으로 사용되지만 장점이 더 많습니다. 열에 대한 인덱스는 효율성 때문에 사용되며 비용 기반 쿼리 옵티마이저에 의해 결정됩니다. 더 나아가, 계산된 열을 사용할 수 있습니다.

XML 및 사용자 정의 형식은 **sql:variable**( ) 및 **sql:column**( )에서 사용할 수 없습니다.

예제: sql:variable( )을 사용한 교차 영역 쿼리

이 쿼리에서 〈book〉 요소의 ISBN은 SQL 변수 @isbn을 사용하여 전달합니다. 상수를 사용하는 대신 sql:variable( )을 사용하여 ISBN의 값을 전달할 수 있으며 이 쿼리는 ISBN이 0-7356-1588-2인 항목 뿐만 아니라 모든 ISBN을 검색하는데 사용할 수 있습니다.

```
DECLARE @KeyName nvarchar(20)
SET   @isbn =  '0-7356-1588-2'
SELECT  xCol
FROM   docs
WHERE   xCol.exist ( '/book[@ISBN = sql:variable("@isbn")]' ) = 1
```

XML 데이터에서 행 집합 생성

전용 속성 관리나 데이터 상호 교환 상황에서는 응용 프로그램이 XML 데이터의 일부를 행 집합에 매핑하는 일이 자주 있습니다. 예를 들어, 입력 매개 변수 테이블을 저장된 프로시저나 함수에 전달하기 위하여, 응용 프로그램은 데이터를 XML로 변환한 다음 그것을 XML 데이터 유형 매개 변수로 전달합니다. 저장 프로시저나 함수 내에서 이 행 집합은 XML 매개변수로부터 다시 생성됩니다.

SQL Server 2000은 이런 목적으로 **OpenXml( )**을 제공합니다. 이것은 행 집합에 대해 관계형 스키마를 지정하고 XML 인스턴스 내의 값이 행 집합의 열에 매핑되는 방식을 지정하여 XML 인스턴스에서 행 집합을 생성하는 기능입니다.

또는, **nodes( )** 메서드를 사용하여 XML 인스턴스 내에 노드 환경을 생성하고 **value( )**, **query( )**, **exist( )** 및 **nodes( )** 메서드에서 그 노드 상황을 사용하여 원하는 행 집합을 생성할 수도 있습니다. **nodes( )** 메서드는 XQuery 표현식을 받아서 XML 열의 각 XML 인스턴스에서 평가하며, XML 인덱스를 효과적으로 사용합니다. 다음 예제는 행 집합을 생성할 때 **nodes( )** 메서드를 사용하는 것을 보여줍니다.

예제: XML 인스턴스에서 속성 추출하기

이름이 "David"가 아닌 저작자의 이름과 성을 추출하려고 한다고 합시다. 행 집합은 FirstName 열과 LastName 열로 구성되어 있습니다. nodes( ) 메서드와 value( ) 메서드를 사용하면 다음과 같은 방법으로 그렇게 할 수 있습니다.

```
SELECT nref.value( 'first-name[1]' ,  'nvarchar(50)' ) FirstName,
      nref.value( 'last-name[1]' ,  'nvarchar(50)' ) LastName
FROM   docs CROSS APPLY xCol.nodes( '//author' ) AS R(nref)
WHERE  nref.exist( '.[first-name != "David"]' ) = 1
```

이 예에서, nodes('//author')를 실행하면 각 XML 인스턴스에 대해 〈author〉 요소로 연결되는 참조 행 집합이 만들어집니다. 저작자의 이름과 성은 이 참조 자료를 기준으로 **value( )** 메서드를 평가하여 알아낼 수 있습니다. XML 열을 인덱싱하면 성능이 더 향상되는데, 이 점은 다음 섹션에서 다룰 것입니다.

예제: XML 변수에서 속성 추출하기

위의 쿼리에서 CROSS APPLY 연산자는 XML 변수나 매개 변수에서 속성을 추출하는 경우에는 필요하지 않습니다. 이 예제에서는 XML 변수 @xVar에 〈book〉 인스턴스가 할당되어 있다고 간주하고 저작자의 〈first-name〉과 〈last-name〉을 검색합니다.

```
DECLARE @xVar XML
SET @xVar =
  '〈book genre="security" publicationdate="2002" ISBN="0-7356-1588-2" 〉
      〈title〉Writing Secure Code〈/title〉
      〈author〉
            〈first-name〉Michael〈/first-name〉
            〈last-name〉Howard〈/last-name〉
      〈/author〉
      〈author〉
            〈first-name〉David〈/first-name〉
            〈last-name〉LeBlanc〈/last-name〉
      〈/author〉
      〈price〉39.99〈/price〉
  〈/book〉

SELECT nref.value( 'first-name[1]' , 'nvarchar(50)' ) FirstName,
     nref.value( 'last-name[1]' , 'nvarchar(50)' ) LastName
FROM  @xVar.nodes( '//author' ) AS R(nref)
WHERE  nref.exist( '.[first-name != "David"]' ) = 1
```

XML 데이터 인덱싱

XML 데이터는 내부 바이너리 형태로 저장되며 스토리지에 최대 2GB까지 넣을 수 있습니다. 각 쿼리는 테이블의 각 행에서 XML blob를 런타임에 1회 이상 분석합니다. 따라서 쿼리 처리가 느려집니다. 쿼리가 일반적으로 발생하는 작업인 경우, XML 열을 인덱싱하는 것이 도움이 됩니다. 물론, 데이터 수정 중에 발생하는 XML 인덱스 유지 관리 비용을 고려해야 합니다.

XML 인덱스는 typed XML 열과 untyped XML 열에서 새로운 DDL 명령문을 사용하면 생성됩니다. 이렇게 하면 해당 열의 모든 XML 인스턴스에 대해 B+tree가 생성됩니다. XML 열의 첫 번째 인덱스는 "기본 XML 인덱스"입니다. 이것을 사용할 때 일반적인 종류의 쿼리 속도를 높이기 위하여 XML 열에서 보조 XML 인덱스의 세 가지 유형을 지원합니다. 이 점은 다음 섹션에서 설명합니다.

기본 XML 인덱스

기본 XML 인덱스에는 기본 테이블(즉, XML 열이 정의된 테이블)의 기본 키에 대한 클러스터형 인덱스가 있어야 합니다. 이 인덱스가 있으면 XML 노드의 Infoset 항목의 하위 집합에 B+tree가 생성됩니다. B+tree의 각 열은 요소나 속성 이름, 노드 값 및 노드 유형과 같은 태그에 해당합니다. 각 열은 XML 데이터의 문서 순서와 구조를 포착하고, 경로 표현식을 효율적으로 평가할 수 있도록 XML 인스턴스의 루트에서부터 각 노드까지의 경로를 포착합니다. 기본 테이블의 기본 키는 인덱스 행과 기본 테이블 행을 연결할 수 있도록 기본 XML 인덱스에 복제됩니다.

XML 스키마에 지정된 태그와 유형 이름은 정수 값에 매핑되며 매핑된 값은 스토리지를 최적화하도록 B+tree에 저장됩니다. 인덱스의 경로 열에는 매핑된 값의 연결 관계가 역순, 즉 노드에서부터 XML 인스턴스의 루트까지의 순서로 저장됩니다. 역순으로 표현하면 경로 접미사를 알고 있는 경우(예: //author/last-name과 같은 경로 표현식의 경우), 경로 값을 대응시킬 수 있습니다.

기본 테이블이 파티션 분할되어 있으면, 기본 XML 인덱스도 동일하게 파티션 분할됩니다. 즉, 동일한 파티션 분할 기능과 파티션 분할 방식을 사용합니다.

XML 열의 전체 XML 인스턴스를 검색합니다(예: SELECT * FROM docs 또는 SELECT xCol FROM docs). XML 데이터 형식 메서드가 포함된 쿼리에서는 기본 XML 인덱스를 사용하며, 인덱스 자체에서 스칼라 값이나 XML 하위 트리를 알아냅니다.

예제: 기본 XML 인덱스 생성
다음 명령문은 docs 테이블의 XML 열 xCol에 idx_xCol이라고 하는 XML 인덱스를 생성합니다.

```
CREATE PRIMARY XML INDEX idx_xCol on docs (xCol)
```

보조 XML 인덱스
기본 XML 인덱스가 생성된 다음 워크로드 내의 다양한 종류의 쿼리 속도를 높이기 위하여 보조 XML 인덱스를 생성할 수 있습니다. 세 가지 종류의 보조 XML 인덱스 - PATH, PROPERTY 및 VALUE - 는 각각 경로 기반 쿼리, 전용 속성 관리 시나리오 및 값 기반 쿼리에 도움이 됩니다.

- PATH 인덱스는 기본 XML 인덱스의 열(path, value)에 B+-tree를 생성합니다. 경로의 값은 경로 표현식에서 계산하며 노드의 값이 있으면 그 값도 사용합니다. PATH 인덱스의 주 필드를 알고 있는 경우 PATH 인덱스를 검색하면 경로 표현식 평가 속도가 빨라집니다. 가장 일반적인 경우는 SELECT 문의 WHERE 구에서 XML 열에 대해 exist( ) 메서드를 사용하는 것입니다.
- PROPERTY 인덱스는 기본 XML 인덱스의 열(PK, path, value)에 B+-tree를 생성합니다. 여기서 PK는 기본 테이블의 기본 키입니다. 이 인덱스는 XML 인스턴스 내에서 속성 값 룩업을 하는데 도움이 됩니다.
- VALUE 인덱스는 기본 XML 인덱스의 열(value, path)에 B+tree를 생성합니다. 이 인덱스는 노드의 값을 알고 있지만 경로가 쿼리에 정확하게 지정되지 않은 경우에 도움이 됩니다. 이런 경우는 일반적으로 //author/last-name [. =" Howard"]와 같은 descendnat axe에서 발생합니다. 여기서 〈author〉 요소는 계층 구조의 한 레벨에서 나타납니다. 또한 "와일드카드" 쿼리 (예: /book [@* = "novel"])에서도 나타납니다. 이 쿼리는 일부 속성의 값이 "novel" 인 〈book〉 요소를 찾는 것입니다. 또한, XML 인덱스는 typed XML의 값 기반 범위 스캔에도 도움이 됩니다.

최대 128 레벨의 XML 계층 구조를 사용할 수 있습니다. 더 긴 경로가 들어 있는 XML 인스턴스는 삽입과 수정 중에 거부됩니다. 비슷하게, 노드 값의 처음 128 바이트까지 인덱싱됩니다. 값이 더 길면 시스템에 들어가기는 하지만 인덱싱이 되지 않습니다.

예제: 경로 기반 룩업

아래 쿼리가 워크로드에서 일반적으로 나타난다고 가정해 봅시다.

```
SELECT  xCol
FROM   docs
WHERE  xCol.exist ( '/book[@genre = "security"]' ) = 1
```

경로 표현식 /book/@genre와 값 "security"는 PATH 인덱스의 키 필드에 해당합니다. 따라서, 이 워크로드에서는 type PATH의 보조 XML 인덱스가 도움이 됩니다.

```
CREATE XML INDEX idx_xCol_Path on docs (xCol)
        USING XML INDEX idx_xCol FOR PATH
```

예제: 객체 속성 가져오기

테이블 T의 각 행에서 속성 "genre"과 "title" 및 책의 ISBN을 검색하는 아래 나오는 쿼리를 고려해 봅시다.

```
SELECT xCol.value ( '(/book/@genre)[1]' ,  'varchar(50)' ),
        xCol.value ( '(/book/title)[1]' ,  'varchar(50)' ),
        xCol.value ( '(/book/@ISBN)[1]' ,  'varchar(50)' )
FROM            docs
```

이 경우에는 property 인덱스가 도움이 됩니다. property 인덱스는 다음과 같이 생성됩니다.

```
CREATE XML INDEX idx_xCol_Property on docs (xCol)
        USING XML INDEX idx_xCol FOR PROPERTY
```

예제: 값 기반 쿼리

다음 쿼리에서, descendant axis나 self axis (//-연산자)는 부분적인 경로를 지정하므로 ISBN의 값을 근거로 한 룩업에서 VALUE 인덱스를 사용하면 도움이 됩니다.

```
SELECT  xCol
FROM            docs
WHERE           xCol.exist ( '//book/@ISBN[. = "0-7356-1588-2"]' ) = 1

VALUE 인덱스는 다음과 같이 생성됩니다.
CREATE XML INDEX idx_xCol_Value on docs (xCol)
        USING XML INDEX idx_xCol FOR VALUE
```

내용 인덱싱

XML 열에 텍스트 전용 인덱스를 생성할 수 있습니다. 이 인덱스는 XML 마크업은 무시하고 XML 값의 내용만 인덱싱합니다. 속성 값은 (마크업의 일부로 간주되기 때문에) 텍스트 전용 인덱싱 대상이 아니며 요소 태그는 토큰 경계선으로 사용됩니다. XML 열에 대해 XML 인덱스와 텍스트 전용 인덱스를 모두 생성할 수 있으며, 텍스트 검색을 XML 인덱스 사용 방식과 결합할 수 있습니다. 텍스트 전용 인덱스를 첫 번째 필터로 사용하여 선택의 폭의 좁힌 다음 XQuery를 적용하여 더 선별합니다.

CONTAINS( ) 및 **XQuery contains**( )를 사용하는 텍스트 전용 검색은 서로 의미가 다릅니다. 후자는 하위 문자열 매칭 방식이고 전자는 어근을 사용하는 토큰 매칭 방식입니다.

예제: XML 열에서 텍스트 전용 인덱스 생성하기

XML 열에 텍스트 전용 인덱스를 생성하는 단계는 다른 SQL 형식 열의 경우와 거의 동일합니다. 기본 테이블에 고유한 키 열이 필요합니다. DDL 명령문은 다음과 같습니다. 여기서 PK__docs__7F60ED59은 테이블의 단일 열 기본 키 인덱스입니다.

```
CREATE FULLTEXT CATALOG ft AS DEFAULT
CREATE FULLTEXT INDEX ON dbo.docs (xCol) KEY INDEX PK__docs__7F60ED59
```

예제: 텍스트 검색과 XML 쿼리의 결합

다음 쿼리는 책 제목에 "Secure" 라는 단어가 들어 있는 XML 값을 검사합니다.

```
SELECT *
FROM   docs
WHERE  CONTAINS(xCol,' Secure' )
AND    xCol.exist( '/book/title/text(    )[contains(.," Secure" )]' ) =1
```

CONTAINS( ) 메서드는 텍스트 인덱스를 사용하여 문서 내의 어디서든지 "Secure" 라는 단어가 들어 있는 XML 값을 하위 집합에 넣습니다. **exist**( ) 구는 책 제목에 "Secure" 라는 단어가 나오는지 확인합니다.

XML 인덱스를 사용한 쿼리 실행

XML 인덱싱을 하면 쿼리 실행 속도가 빨라집니다. XML 열에 기본 XML 인덱스가 있으면 쿼리는 항상 그 인덱스를 기준으로 컴파일됩니다. 전체 쿼리(관계형 부분과 XML 부분 모두)에 대해 단일 쿼리 계획이 생성되며, 데이터베이스 엔진의 비용 기준 옵티마이저를 사용하여 최적화됩니다. 보조 XML 인덱스는 쿼리 옵티마이저의 비용 추정을 기준으로 선택됩니다.

XML 인덱스에 대한 카탈로그 뷰

카탈로그 뷰는 XML 인덱스에 대한 메타데이터 정보를 알려줍니다. 카탈로그 뷰 sys.indexes에는 인덱스 "type" 이 3인 XML 인덱스 해당 항목이 들어 있습니다. "name" 열에는 XML 인덱스의 이름이 들어 있습니다.

XML 인덱스는 카탈로그 뷰 sys.xml_indexes에도 기록됩니다. 이 카탈로그 뷰에는 sys.indexes의 모든 열과 XML 인덱스에 의미 있는 몇 가지 특수 열이 들어 있습니다. "secondary_type" 열의 값이 NULL이면 기본 XML 인덱스를 가리킵니다. 값 'P' , 'R' 및 'V' 는 각각 보조 XML 인덱스인 PATH, PROPERTY 및 VALUE을 가리킵니다. "secondary_type_desc" 열에는 기본 XML 인덱스에 대해 NULL이 포함되어 있으며, 세 가지 보조 XML 인덱스에 대해서는 문자열 "PATH" , "PROPERTY" 및 "VALUE" 가 들어 있습니다.

XML 인덱스의 공간 사용량은 테이블 값 함수 sys.dm_db_index_physical_stats( )에서 알 수 있습니다. 이 함수는 XML 인덱스를 포함한 모든 인덱스 유형 별로 사용된 디스크 페이지 수, 평균 행 크기(바이트), 레코드 수 및 그 외의 정보를 알려줍니다. 이 정보는 각 데이터베이스 파티션 별로 알 수 있습니다. XML 인덱스는 기본 테이블과 동일한 파티션 분할 방식과 파티션 분할 기능을 사용합니다.

예제: XML 인덱스의 공간 사용량

```
SELECT sum(Pages)
FROM        sys.dm_db_index_physical_stats ( 'docs' ,  'idx_xCol_Path' ,
DEFAULT,  'DETAILED' )
```

이렇게 하면 모든 파티션에서 테이블 T의 XML index idx_xCol_Path가 차지하고 있는 디스크 페이지 수를 알 수 있습니다. **sum**( ) 함수가 없으면, 파티션 당 디스크 페이지가 표시됩니다.

XML 스키마 프로세싱

XML 스키마는 시스템에서 선택적입니다. 앞에서 언급한 것처럼, XML 스키마에 연결되지 않은 XML 데이터 형식은 untyped로 간주됩니다. XML 노드 값은 유니코드 문자열로 저장되며 XML 인스턴스의 구성 상태는 검사 대상입니다. untyped XML 열은 인덱싱할 수 있습니다.

XML의 형식은 XML 스키마 컬렉션에 등록된 XML 스키마를 XML 데이터 형식과 연결하면 설정이 됩니다. 새로운 DDL 명령문을 이용하면 XML 스키마 컬렉션을 생성할 수 있습니다. 이 스키마 컬렉션에는 하나 이상의 XML 스키마를 등록할 수 있습니다. XML 스키마 컬렉션에 연결된 XML 열, 매개 변수 또는 변수는 컬렉션의 모든 XML 스키마에 따라 형식이 설정됩니다. XML 스키마 컬렉션 내에서, 형식 시스템은 해당 타겟 네임스페이스를 사용하여 각 XML 스키마를 구분합니다.

XML 인스턴스의 각 최상위 XML 요소는 자신에게 적용되는 가능성 있는 빈 타겟 네임스페이스를 지정해야 합니다. 데이터는 각 최상위 요소의 타겟 네임스페이스에 따라 삽입 및 수정 중에 검증이 됩니다. 바이너리 XML 표현 형태는 관련된 XML 스키마 정보를 기준으로 형식이 설정된 값을 인코딩하며, 재분석하는 것이 untyped XML에 비해 더 효율적이 될 수 있도록 충분히 규정이 됩니다. XML 인덱스의 값도 적절하게 형식이 설정됩니다(예: XML 스키마에 xs:decimal로 정의되어 있으면 /book/price는 십진수로 저장됩니다).

쿼리를 컴파일하는 동안, XML 스키마는 형식을 검사하는데 사용되며 형식이 일치하지 않으면 정적 오류가 발생합니다. 쿼리 컴파일러는 쿼리 최적화를 위해서도 XML 스키마를 사용합니다.

데이터베이스 엔진의 메타데이터 하위 시스템에는 XML 스키마 컬렉션 및 그 안에 들어 있는 XML 스키마와 같은 XML 형식 정보와 원시 XSD와 관계형 형식 시스템 사이의 매핑이 들어 있습니다. 거의 모든 W3C XML Schema 1.0 규격을 지원합니다. XML 스키마 문서의 주석과 주해는 보존되지 않으며, key/keyref도 지원하지 않습니다.

XML 스키마 컬렉션

XML 스키마 컬렉션은 메타데이터 엔티티이며 관계형 스키마에 의해 범위가 정해집니다. 또한 관련이 있을 수도 있고 (예: 〈xs:import〉를 사용한 경우) 없을 수도 있는 하나 이상의 XML 스키마가 들어 있습니다. XML 스키마 컬렉션 내의 각 XML 스키마는 해당 타겟 네임스페이스로 구분됩니다. XML 스키마 컬렉션은 테이블과 마찬가지로 보안이 설정된 엔티티입니다.

XML 스키마 컬렉션은 CREATE XML SCHEMA COLLECTION 구문을 사용하여 생성하며 하나 이상의 XML 스키마를 제공합니다. XML 스키마 컬렉션을 사용하면 XML 열의 형식을 설정할 수 있습니다. 이런 설계에서는 여러 XML 스키마에 따라 형식이 설정된 XML을 동일한 열에 저장하는 유연한 데이터 모델이 가능합니다. 이것은 특히 XML 스키마의 수가 매우 많은 경우에 편리합니다. 더 나아가, 이 설계는 XML 스키마 변환을 어느 정도까지 지원합니다.

뿐만 아니라 typed XML 열에서 DOCUMENT 옵션이나 CONTENT 옵션을 사용하면 XML 열에 XML 트리를 저장할 수 있는지 아니면 프래그먼트를 저장할 수 있는지 지정할 수 있습니다. 기본 설정은 CONTENT입니다. DOCUMENT의 경우, 각 XML 인스턴스는 최상위 요소의 타겟 네임스페이스를 지정해야 하며, 이 네임스페이스에 따라 인스턴스가 검증되거나 형식이 설정됩니다. 반면에, CONTENT의 경우, 각 최상위 요소는 XML 스키마 컬렉션의 타겟 네임스페이스 중의 하나를 지정할 수 있습니다. XML 인스턴스는 인스턴스에서 나타나는 모든 타겟 네임스페이스에 따라 검증이 되고 형식이 설정됩니다.

예제: XML 스키마 컬렉션 생성

타겟 네임스페이스가 http://myBooks인 XML 스키마를 사용하여 XML 인스턴스의 형식을 설정하려고 한다고 합시다. 아래에 나오는 것처럼 XML 스키마 컬렉션 myCollection을 생성하고 XML 스키마를 myCollection의 내용으로 공급하십시오.

```
CREATE XML SCHEMA COLLECTION myCollection AS
 '〈xsd:schema xmlns:xsd=" http://www.w3.org/2001/XMLSchema"
  xmlns=" http://myBooks"
        elementFormDefault=" qualified"
  targetNamespace=" http://myBooks" 〉
  〈xsd:element name=" bookstore"  type=" bookstoreType"  /〉
  〈xsd:complexType name=" bookstoreType" 〉
        〈xsd:sequence maxOccurs=" unbounded" 〉
          〈xsd:element name=" book"  type=" bookType"  /〉
        〈/xsd:sequence〉
  〈/xsd:complexType〉
  〈xsd:complexType name=" bookType" 〉
        〈xsd:sequence〉
          〈xsd:element name=" title"  type=" xsd:string"  /〉
          〈xsd:element name=" author"  type=" authorName"  /〉
          〈xsd:element name=" price"  type=" xsd:decimal"  /〉
        〈/xsd:sequence〉
        〈xsd:attribute name=" genre"  type=" xsd:string"  /〉
        〈xsd:attribute name=" genre"  type=" xsd:string"  /〉
        〈xsd:attribute name=" genre"  type=" xsd:string"  /〉
```

```
  〈/xsd:complexType〉
  〈xsd:complexType name=" authorName" 〉
       〈xsd:sequence〉
        〈xsd:element name=" title"  type=" xsd:string"  /〉
        〈xsd:element name=" title"  type=" xsd:string"  /〉
       〈/xsd:sequence〉
  〈/xsd:complexType〉
〈/xsd:schema〉'
```

새 메타데이터 엔티티는 XML 스키마가 등록된 myCollection에 맞게 생성됩니다. 다음과 같이 하면 XmlCatalog 테이블에 새 행을 추가할 수 있습니다(본 문서 앞 부분에 나온 "예제: 테이블의 형식이 설정된 XML 열" 참조).

```
INSERT XmlCatalog VALUES(1,  '〈?xml version=" 1.0" ?〉
〈bookstore xmlns=" http://myBooks" 〉
  〈book genre=" autobiography"  publicationdate=" 1981"
                    ISBN=" 1-861003-11-0" 〉
    〈title〉 The Autobiography of Benjamin Franklin 〈/title〉
    〈author〉
      〈first-name〉 Michael 〈/first-name〉
      〈last-name〉 Howard 〈/last-name〉
    〈/author〉
    〈price〉 39.99 〈/price〉
  〈/book〉
  〈book genre=" novel"  publicationdate=" 1967"
                    ISBN=" 1-201-63361-2" 〉
    〈title〉 The Confidence Man 〈/title〉
    〈author〉
      〈first-name〉 Michael 〈/first-name〉
      〈last-name〉 Howard 〈/last-name〉
    〈/author〉
    〈price〉 39.99 〈/price〉
  〈/book〉
  〈book genre=" philosophy"  publicationdate=" 1991"
                    ISBN=" 1-861001-57-6" 〉
    〈title〉 The Gorgias 〈/title〉
    〈author〉
      〈first-name〉 Michael 〈/first-name〉
      〈last-name〉 Howard 〈/last-name〉
    〈/author〉
    〈price〉 39.99 〈/price〉
  〈/book〉
〈/bookstore〉
')
```

XML 스키마 컬렉션 수정

ALTER XML SCHEMA COLLECTION 명령문은 새로운 최상위 스키마 구성 요소를 사용하여 XML 스키마 컬렉션의 XML 스키마를 확장하는 것과 XML 스키마 컬렉션에 새 XML 스키마를 등록하는 것을 지원합니다. 이것은 다음 예제에서 볼 수 있습니다.

예제: XML 스키마 컬렉션 변경

다음 명령문은 타겟 네임스페이스 http://myDVD가 있는 새 XML 스키마를 XML 스키마 컬렉션 myCollection에 추가하는 방법을 보여줍니다.

```
ALTER XML SCHEMA COLLECTION myCollection ADD
 '〈xsd:schema xmlns:xsd=" http://www.w3.org/2001/XMLSchema"
  xmlns=" http://myDVD"
        elementFormDefault=" qualified"
  targetNamespace=" http://myDVD" 〉
  〈xsd:element name=" dvdstore"  type=" dvdstoreType"  /〉
  〈xsd:complexType name=" dvdstoreType" 〉
        〈xsd:sequence maxOccurs=" unbounded" 〉
          〈xsd:element name=" dvd"  type=" dvdType"  /〉
        〈/xsd:sequence〉
  〈/xsd:complexType〉
  〈xsd:complexType name=" dvdType" 〉
        〈xsd:sequence〉
          〈xsd:element name=" title"  type=" xsd:string"  /〉
          〈xsd:element name=" price"  type=" xsd:decimal"  /〉
        〈/xsd:sequence〉
        〈xsd:attribute name=" genre"  type=" xsd:string"  /〉
        〈xsd:attribute name=" genre"  type=" xsd:string"  /〉
  〈/xsd:complexType〉
〈/xsd:schema〉'
```

XML 스키마 컬렉션의 카탈로그 뷰

XML 스키마 컬렉션의 SQL 카탈로그 뷰를 이용하면 각 XML 스키마 네임스페이스의 내용을 재구성할 수 있습니다. XML 스키마 컬렉션은 카탈로그 뷰 sys.xml_schema_collections에서 계수됩니다. XML 스키마 컬렉션 "sys"는 시스템에 의해 정의되며 명시적으로 로드하지 않고도 모든 사용자 정의 XML 스키마 컬렉션에서 사용할 수 있는 사전 정의된 네임스페이스를 넣을 수 있습니다. 이 목록에는 xml, xs, xsi, fn 및 xdt의 네임스페이스가 들어 있습니다.

언급할 만한 가치가 있는 다른 두 가지 카탈로그 뷰는 각 XML 스키마 컬렉션 내의 모든 네임스페이스를 계수하는 sys.xml_schema_namespaces와 각 XML 스키마 내의 모든 XML 스키마 구성 요소를 계수하는 sys.xml_schema_components입니다.

내장된 함수 **XML_SCHEMA_NAMESPACE** ( )는 관계형 스키마와 XML 스키마 컬렉션의 이름을 넣을 수 있으며, 선택에 따라서는 XML 스키마의 타겟 네임스페이스도 넣을 수 있습니다. 이렇게 하면 해당 XML 스키마가 들어 있는 XML 데이터 형식 인스턴스가 만들어집니다. 타겟 네임스페이스 인수를 넣지 않으면 내장 함수는 사전 정의된 XML 스키마를 제외한 XML 스키마 컬렉션의 모든 XML 스키마가 들어 있는 XML 인스턴스를 생성합니다.

예제: XML 스키마 컬렉션의 XML 네임스페이스 계수

XML 스키마 컬렉션 "myCollection"에 대해 다음 쿼리를 사용하십시오.

```
SELECT XSN.name
FROM        sys.xml_schema_collections XSC
 JOIN sys.xml_schema_namespaces XSN ON
(XSC.xml_collection_id = XSN.xml_collection_id)
WHERE       XSC.name =  'myCollection'
```

예제: XML 스키마 컬렉션에서 지정된 XML 스키마 생성

다음 명령문은 (관계형) 스키마 dbo 내의 XML 스키마 컬렉션 "myCollection"에서 타겟 네임스페이스가 "http://myBooks"인 XML 스키마를 생성합니다.

```
SELECT XML_SCHEMA_NAMESPACE (N' dbo' , N' myCollection' ,
N' http://myBooks' )
```

XML 스키마 컬렉션의 액세스 제어

XML 스키마 컬렉션은 SQL Server 2005의 보안 모델을 사용하는 다른 SQL 객체와 똑같이 보호할 수 있습니다. 데이터베이스 내에서 XML 스키마 컬렉션을 생성할 수 있는 권한을 주 서버에 부여할 수 있습니다. 각 XML 스키마 컬렉션은 ALTER, CONTROL, TAKE OWNERSHIP, REFERENCES, EXECUTE 및 VIEW DEFINITION 등의 권한을 지원합니다.

- ALTER 권한은 ALTER XML SCHEMA COLLECTION 명령문을 실행하는데 필요합니다.
- TAKE OWNERSHIP 권한은 ALTER AUTHORIZATION 명령문을 실행하여 한 주 서버에서 다른 주 서버로 XML 스키마 컬렉션의 소유권을 넘기는데 필요합니다.
- REFERENCES 권한은 XML 열이나 매개 변수의 형식을 설정하거나 제한 조건을 적용하는 것과 같이 스키마 바인딩이 필요한 모든 경우에 XML 스키마 컬렉션을 사용할 수 있는 권한을 주 서버에 부여합니다.
- EXECUTE 권한은 XML 스키마 컬렉션을 대상으로 주 서버가 삽입하거나 업데이트한 값을 검증하는데 필요합니다. 이 권한은 XML 데이터 형식을 사용하는 typed XML 열, 변수 및 매개 변수에서 나온 값을 쿼리하는데도 필요합니다.
- VIEW DEFINITION 권한은 카탈로그 뷰에서 XML 스키마 컬렉션, 그 안에 들어 있는 모든 XML 스키마 및 그 XML 스키마에 들어 있는 모든 스키마 구성 요소에 해당하는 행을 액세스할 수 있는 권한을 주 서버에 부여합니다.
- CONTROL 권한은 DROP XML SCHEMA COLLECTION 명령문을 사용하여 XML 스키마 컬렉션을 삭제하는 것을 포함하여 XML 스키마 컬렉션에 대해 동작을 수행할 수 있는 권한을 주 서버에 부여합니다. 이 권한에는 XML 스키마 컬렉션에 대한 다른 권한이 포함됩니다.

테이블이나 XML 열에 대한 다른 권한이 있어도 XML 스키마 컬렉션에 대한 권한이 필요합니다. XML 스키마 컬렉션 C에 따라 형식이 설정된 XML 열 X가 있는 테이블 T를 생성하려면 주 서버가 테이블 생성 권한과 XML 스키마 컬렉션 C에 대한 REFERENCES 권한을 가지고 있어야 합니다. 데이터를 열 X에 삽입할 수 있는 권한이 있는 주 서버는 XML 스키마 컬렉션 C에 대해 EXECUTE 권한이 있는 경우에만 그렇게 할 수 있습니다. 비슷하게, XML 데이터 형식 메서드를 사용하여 열 X의 데이터를 쿼리하려면 주 서버가 열 X에 대한 SELECT 권한과 C에 대한 EXECUTE 권한을 가지고 있어야 합니다. 하지만, SELECT X FROM T나 SELECT * FROM T에서 처럼 열 X에서 전체 XML 값을 검색하는데는 X에 대한 SELECT 권한으로 충분합니다.
권한은 주 서버에서 요청할 수 있으며, SQL Server 2005의 보안 모델에서 허용하는 권한이라도 주 서버에 대해 거부될 수 있습니다.

카탈로그 뷰의 가시성

XML 스키마 컬렉션에 대해 ALTER, TAKE OWNERSHIP, REFERENCES, VIEW DEFINITION 또는 CONTROL 권한이 있는 주 서버는 XML 스키마 컬렉션, 그 안에 들어 있는 XML 스키마 및 해당 XML 스키마 구성 요소에 대해 카탈로그 뷰 행을 액세스할 수 있습니다. 그런 권한 중 하나가 있으면, 주 서버는 내장 함수인 XML_SCHEMA_NAMESPACE( )를 사용하여 FOR XML ... XMLSCHEMA 문으로 XML 스키마 컬렉션의 내용을 액세스할 수 있습니다.

주 서버의 VIEW DEFINITION 권한이 거부되면, 주 서버는 XML_SCHEMA_NAMESPACE( )나 FOR XML ... XMLSCHEMA를 사용하여 카탈로그 뷰의 XML 스키마 컬렉션을 액세스할 수 없습니다.

FOR XML의 향상된 기능

TYPE 지시어를 사용하면 XML 열, 변수 또는 매개 변수에 할당하거나 XML 데이터 유형 메서드를 사용하여 쿼리할 수 있는 XML 데이터 유형 인스턴스가 생성됩니다. 이를 통해 SELECT … FOR XML TYPE 문의 중첩이 가능해집니다.

PATH 모드에서는 XML 트리에서 열의 값이 나타나는 경로를 지정할 수 있습니다. 이것을 앞에서 언급한 중첩 기능과 함께 사용하면 FOR XML EXPLICIT보다 편리합니다. 물론, 계층 구조가 깊은 경우에는 제대로 작동하지 않을 수 있습니다.

ELEMETS와 함께 사용되는 지시어 XSINIL은 속성이 xsi:nil=" true" 인 요소에 NULL을 매핑합니다. 새로운 ROOT 지시어를 사용하면 FOR XML의 모든 모드에서 루트 노드를 지정할 수 있습니다. 이 새 XMLSCHEMA 지시어는 XSD 인라인 스키마를 생성합니다.

성능 관련 지침

XML 데이터 모델은 관계형 데이터 모델보다 더 기능이 풍부하고 더 복잡합니다. XML 데이터 모델에서는 복잡한 데이터를 모델링할 수 있을 뿐만 아니라 데이터 내의 계층 구조 관계와 문서 순서도 그대로 보존이 되어야 합니다. 문서 순서는 XML 노드 식별자를 기준으로 정렬하여 유지됩니다. 이렇게 하면 동시에 계층 관계도 그대로 유지됩니다. 이것은 비교적 복잡한 쿼리 계획에도 도움이 됩니다.

성능을 향상시키려면 구조적 데이터를 테이블의 관계형 열에 저장해야 합니다. 데이터가 반구조적이거나 비구조적이며 XML 마크업도 포함하고 있는 경우 모델링 필요에 맞는 XML 데이터 모델을 선택하십시오. 하지만 성능이 향상될 것으로 기대하지는 마십시오. XML 스키마는 쿼리 최적화에 도움이 됩니다.

SQL Server CLR의 XML 지원

SQL Server CLR 지원 기능을 이용하면 관리되는 코드에 서버측 로직을 포함시켜 비즈니스 규칙을 시행하게 할 수 있습니다. 이 비즈니스 로직을 XML 데이터에 추가하는 방법은 여러 가지입니다.

- 관리되는 코드에 XML 규칙을 넘겨 받을 SQLCLR 함수를 넣고 System.Xml 네임스페이스가 제공하는 XML 프로세싱 기능을 사용할 수 있습니다. 예를 들면, 아래 나오는 것처럼 XML 데이터에 XSL 변환을 적용하는 것입니다. 또는, XML을 해체하여 하나 이상의 관리되는 클래스로 만든 다음 관리되는 코드를 사용하여 실행할 수 있습니다.
- XML 열에서 비즈니스 필요에 맞는 프로세싱을 호출하는 Transact-SQL 저장된 프로시저와 함수를 만들 수 있습니다.

예제: XSL 변환 적용

XML 데이터 형식 인스턴스와 XSL 변환을 위한 파일 경로를 받아서 변환을 XML 데이터에 적용하고 변환된 XML을 돌려주는 CLR 함수 TransformXml.ApplyXslTransform ( )를 생각해 봅시다. C#로 만든 기본 함수는 다음과 같습니다.

```
using System;
using System.Data.SqlTypes;
using System.Xml;
using System.Xml;
using System.Xml;
public class TransformXml
{
   public static SqlXml ApplyXslTransform (
SqlXml XmlData, string xslPath) {
        // Load XSL transformation
        XslCompiledTransform xform = new XslCompiledTransform( );
        xform.Load (xslPath);

        // Load XML data
        XPathDocument xDoc = new XPathDocument (XmlData.CreateReader( ));
        XPathNavigator nav = xDoc.CreateNavigator ( );

        // Apply the transformation
        // using makes sure that we flush the writer at the end
        using (XmlWriter writer = nav.AppendChild( ))
        {
           xform.Transform(XmlData.CreateReader( ), writer);
        }

        // Return the transformed value
        SqlXml retSqlXml = new SqlXml (nav.ReadSubtree( ));
        return (retSqlXml);
  }
}
```

이 어셈블리는 CREATE ASSEMBLY 문을 사용하여 데이터베이스에 등록해야 하며 ApplyTransformXml( )created에 해당하는 사용자 정의 Transact-SQL 함수 SqlXslTransform( )는 CREATE FUNCTION 문을 사용하여 등록해야 합니다. 이 명령문에 대한 자세한 내용은, SQL Server 2005 Books Online을 참조하십시오. 그 다음에 다음 쿼리에서처럼 Transact-SQL에서 이 SQL 함수를 호출할 수 있습니다.

```
SELECT SqlXslTransform (xCol,  'C:\temp\xsltransform.xsl' )
FROM              docs
WHERE  xCol.exist( '/book/title/text(    )[contains(.," secure" )]' ) =1
```

이 쿼리 결과에는 변환된 XML의 행 집합이 들어 있습니다.

SQLCLR를 사용하면서 XML 데이터를 분해하여 테이블이나 속성 프로모션에 넣고 System.Xml 네임스페이스에서 관리된 클래스를 사용하여 XML 데이터를 쿼리하는데 사용할 수 있는 완전히 새로운 세계가 열립니다. 자세한 내용은 SQL Server 2005 Books Online 및 Visual Studio™ 2005 Books Online을 참조하십시오.

SQL Server 2005의 클라이언트측 XML 프로세싱

XML 데이터 형식에 대한 클라이언트측 지원

XML 데이터 형식에 대한 클라이언트 액세스는 ADO.NET, SQL 원시 클라이언트(SQLNCLI) 및 SOAP over HTTP를 사용하여 제공됩니다. 앞의 두 가지는 아래에서 설명합니다. SOAP 액세스에 대해서는 SQL Server Books Online을 참조하십시오.

.NET Framework V2.0의 ADO.NET XML 지원

XML 데이터 형식은 SqlDataReader.GetSqlXml( ) 메서드의 System.Data.SqlTypes 네임스페이스에서 SqlXml 클래스로 표현됩니다. SqlXml.CreateReader( ) 함수를 사용하여 SqlXml 객체에서 XmlReader를 가져올 수 있습니다.
XML 열의 형식을 지정하는 XML 스키마 컬렉션의 3 부분으로 된 이름은 (SqlDataReader 객체에서 GetSchemaTable( ) 나 GetSqlMetaData (int)를 사용하여) XML 열 메타데이터에서 데이터베이스의 이름(XmlSchemaCollectionDatabase), 관계형 스키마(XmlSchemaCollectionOwingSchema) 및 XML 스키마 컬렉션 (XmlSchemaCollectionName)을 가리키는 세 가지 속성으로 가져올 수 있습니다.

새로운 스키마 행 집합 XMLSCHEMA은 클라이언트가 서버에서 XML 스키마를 검색하는데 사용할 수 있습니다.
XMLSCHEMA 행 집합에는 XML 스키마 컬렉션, 타겟 네임스페이스 및 XML 스키마 내용 자체에 해당하는 세 열이 있습니다.
다음 예제는 XML 데이터 형식에 대한 관리된 액세스를 위한 기본 코드입니다.

예제: XML 데이터 형식에 대한 프로세스 내 액세스

아래의 C# 코드는 XML 데이터 형식을 프로세스내 공급자가 액세스하는 방법을 보여줍니다. 아래 나오는 상황 연결 관계를 이용하면 CLR 코드를 호출한 것과 동일한 환경에서 SQL 문을 실행할 수 있습니다. 프로세스 외 액세스의 경우, 데이터베이스로 새로 연결해야 합니다.

```
using System;
using System.Xml;
using System.Data;
using System.Data.SqlTypes;
using System.Data.SqlClient;

class xmldtADONETReadAccessInProc
{
      static void ReadXmlDataType (  ) {
            // in-proc connection to server
            SqlConnection conn =
new SqlConnection("context connection=true");

            // prepare query to select xml data
            SqlCommand cmd = conn.CreateCommand(  );
            cmd.CommandText =
 "SELECT xCol.query( '//section' ) FROM docs" ;
```

```
                    // execute query and retrieve incoming data
                    SqlDataReader rdr = cmd.ExecuteReader(    );
                    r.Read(    );

                    // access XML data type field in rowset
                    SqlXml xml = r.GetSqlXml(0);
                    new XmlTextWriter(Console.Out).WriteNode(
xml.CreateReader(    ), true);
        }
}
```

예제: FOR XML TYPE의 실행 결과 액세스

TYPE 지시어를 사용한 FOR XML을 실행하면 관리된 클라이언트가 SqlXml 클래스를 사용하여 검색할 수 있는 XML 데이터 형식 인스턴스가 생성됩니다. 그렇기 때문에, 위의 예제에 나오는 코드는 cmd.CommandText를 다음과 같이 수정해도 변경되지 않습니다.

```
cmd.CommandText =
   "SELECT xCol.query( '//section' ) FROM docs FOR XML AUTO, TYPE" ;
```

예제: SQL 클라이언트 공급자를 사용한 XML 데이터 형식 열 업데이트

아래 코드는 SQL 클라이언트 공급자를 사용하여 XML 열의 값을 대체하는 WriteXmlDataType( ) 메서드입니다. 프로세스내 공급자의 코드도 비슷합니다.

```
using System;
using System.Xml;
using System.Data;
using System.Data.SqlTypes;
using System.Data.SqlClient;

class xmldtADONETUpdateAccess
{
        static void WriteXmlDataType (    ) {
                    // connection to server
                    SqlConnection conn = new SqlConnection("server=server1;"  +
 " database=XMLtest; Integrated Security=SSPI");
                    conn.Open(    );
        // update XML column at the server
        SqlCommand cmd = conn.CreateCommand(    );
        cmd.CommandText =  "UPDATE docs SET xCol=@x WHERE id=1" ;
// set value of XML parameter
        SqlParameter p = cmd.Parameters.Add("@x" , SqlDbType.Xml);
```

```
p.Value = new SqlXml(new XmlTextReader("<hello/>",
XmlNodeType.Document, null));

// execute update and close connection
        cmd.ExecuteNonQuery(    );
conn.Open(    );
}
}
```

SQL 원시 클라이언트 액세스

새로운 SQL 원시 액세스(SQLNCLI)의 OLE DB 공급자에서, XML 데이터 형식 열은 유니코드 텍스트(DBTYPE_XML, DBTYPE_BYTES, DBTYPE_BSTR, DBTYPE_WSTR 및 DBTYPE_VARIANT)로 검색할 수도 있고 ISequentialStream을 사용하여 유니코드 문자 스트림(DBTYPE_UNKNOWN)으로 검색할 수도 있습니다. 기본값은 DBTYPE_XML입니다. XML 데이터는 UTF-16 인코딩으로 서버로 보낼 수 있습니다. 이 경우 응용 프로그램은 그 데이터가 이미 인코딩된 UTF-16인지 확인해야 합니다. 바이트 순서 표시인 0xFFFE이 첫 번째 바이트로 필요합니다. 응용 프로그램은 데이터베이스 서버의 인코딩과 호환이 되기만 한다면 약간 다른 인코딩이 된 XML 데이터도 서버로 보낼 수 있습니다. XML 데이터 내에서 지정된 인코딩은 인정할 수 있습니다. 그 외의 모든 경우에 XML은 바이너리 데이터로 보내야 합니다.

3 부분으로 된 XML 스키마 컬렉션 이름은 IDBSchemaRowset::GetRowset(   )을 실행하여 나온 COLUMNS 스키마 행 집합의 새로운 세 열에 들어갑니다. 즉, SS_XML_SCHEMACOLLECTION_CATALOGNAME에는 카탈로그 이름이 나오며, SS_XML_ SCHEMACOLLECTION SCHEMANAME에는 XML 스키마 컬렉션이 존재하는 관계형 스키마의 이름이 나오고, SS_XML_SCHEMACOLLECTIONNAME에는 XML 스키마 컬렉션의 이름이 나옵니다. 이 이름들은 DBTYPE_WSTR형식입니다. 이 세 열은 데이터 검색과 업데이트 모두에서 untyped XML 열에 대해 NULL 값을 가집니다.

PROCEDURE_PARAMETERS 스키마 행 집합과 IColumnRowset:GetColumnRowset(   )도 비슷하게 변경되었습니다.
XML 스키마 컬렉션의 내용을 검색하기 위해, 클라이언트는 XML_SCHEMA_NAMESPACE(   ) 호출에서 이 이름들을 사용하여 서버를 개별적으로 액세스하여 XML 스키마를 XML 데이터 형식으로 돌려 받을 수 있습니다. 또는, 새로운 SS_XMLSCHEMA 스키마 행 집합이 포함된 IDBSchemaRowset은 카탈로그 이름, 관계형 스키마 이름, XML 스키마 컬렉션 이름, 타겟 네임스페이스 및 XML 스키마를 돌려줍니다.

SQLNCLI를 사용하는 ODBC 액세스의 경우, XML 데이터 유형은 SQL_SS_XML이라고 하는 유니코드 가변 길이 문자 데이터에 매핑됩니다. 3 부분으로 된 XML 스키마 컬렉션 이름은 XML 열의 SqlColAttribute를 통하여 CharacterAttributePtr로 나타납니다. 데이터베이스, 관계형 스키마 및 XML 스키마 컬렉션의 필드 식별자는 각각 SQL_CA_SS_XML_SCHEMACOLLECTION_CATALOG_NAME, SQL_CA_SS _XML_SCHEMACOLLECTION_SCHEMA_NAME 및SQL_CA_SS _XML_SCHEMACOLLECTION_NAME입니다.

사용자들은 SQL Server 2005 릴리스의 데이터베이스 서버 툴이나 클라이언트 툴을 설치하여 SQL 원시 클라이언트가 되어야 합니다.

SQL Server 2005의 XML 데이터는 SQLOLEDB를 기본 DBTYPE_WSTR로 사용하면 MDAC API에서 사용할 수 있습니다. 이전 SQL Server 2005 클라이언트에게는 XML 열을 사용자 정의 함수의 결과로 보낼 수 없습니다. 그렇게 하려고 하면 오류가 발생합니다.

SQLXML - XML 스키마와 관계형 스키마 사이의 매핑

SQLXML 매핑 기술을 사용하면 관계형 데이터의 논리적 XML Views를 생성할 수 있습니다. "매핑" 또는 "주석이 포함된 스키마"라고도 하는 XML View는 임의의 XSD 스키마에 특수한 주석을 추가하면 생성됩니다. 다른 SQLXML 기술은 이 주석이 포함된 스키마를 사용하여 논리적 XML 뷰에 대한 쿼리와 업데이트를 관계형 테이블에 대한 쿼리와 업데이트로 변환할 수 있습니다.

- XML View를 XPath 쿼리와 결합하면, SQLXML이 요청된 데이터를 찾아서 스키마에 지정된 대로 형태를 만드는 FOR XML 쿼리를 생성합니다.
- SQLXML Updategrams은 XML 인스턴스의 변경 내용을 표현합니다. 이 변경 내용은 주석이 포함된 스키마와 결합되어 관계형 변경에 반영되며, 낙관적 동시성을 사용하여 적절한 데이터가 업데이트되게 합니다.
- SQLXML Bulkload는 XML View를 사용하여 XML 데이터를 관계형 테이블에 "넣습니다".

관계형 테이블의 XML View 생성

데이터베이스의 XML View를 생성하려면, XML 데이터의 XSD 스키마에서 시작합니다. 데이터베이스 테이블/뷰의 행은 스키마의 복잡한 형식 요소들로 매핑됩니다. 데이터베이스에서 이 열의 값은 속성 또는 단순 형식 요소로 매핑됩니다.

기본적으로 명시적인 주석이 포함되어 있지 않으면, SQLXML은 복잡한 형식의 요소가 테이블 및 단순 형식 요소로 매핑되며 속성은 열에 매핑된다고 가정합니다. 이것은 요소와 속성의 이름이 데이터베이스의 테이블과 열의 이름과 정확하게 일치해야만 실행이 됩니다.

요소/속성의 이름이 매핑되는 테이블(뷰)나 열 이름과 동일하지 않으면, 명시적인 매핑을 만들어야 합니다. 다음 주석은 XML 문서의 요소나 속성과 데이터베이스의 테이블(뷰)나 열 사이의 매핑을 지정하는데 사용됩니다.

- **sql:relation** - XML 요소를 데이터베이스 테이블에 매핑합니다.
- **sql:field** - 요소나 속성을 데이터베이스 열에 매핑합니다.

XML View에 계층 구조를 생성하는 관계 매핑

데이터베이스에서, 테이블은 외래 키 관계에 의해 연관성을 가집니다. XML에서, 이와 동일한 관계는 요소의 중첩된 계층 구조로 표현됩니다. 매핑에서 적절한 중첩 구조를 만들려면, 요소들이 서로 연결되는 방식을 지정해야 합니다. sql:relationship 주석을 사용하면 매핑하는 스키마 요소들 사이에 이런 관계를 설정할 수 있습니다. 이 주석 내에서 부모 테이블과 자식 테이블을 지정할 수 있으며 각 테이블 내에서 결합을 수행하는데 사용되는 열도 지정할 수 있습니다. 그러면 SQLXML은 이 정보를 사용하여 매핑에 맞는 적절한 중첩 계층 구조를 만듭니다.

오버플로를 사용하여 사용되지 않는 데이터를 저장하기

매핑은 XML 데이터가 일정한 구조를 가질 때 작동합니다. 하지만, XML에는 비구조적 데이터가 있을 수도 있고 특정한 열에 매핑되지 않는 데이터가 있을 수도 있습니다. 이런 데이터를 저장해 두었다가 나중에 검색하려면 sql:overflow 주석을 사용할 수 있습니다. sql:overflow 주석은 사용되지 않은 모든 데이터를 저장할 열을 지정하고 쿼리를 실행할 때 그 데이터를 검색할 위치를 지정합니다.

오버플로 열을 이용하면 데이터베이스에 추가하지 않고도 XML을 확장할 수 있습니다. 데이터베이스에 요소나 속성을 저장할 열을 추가하지 않고도 언제든지 XML 구조에 요소와 속성을 추가할 수 있습니다. 요소나 속성은 오버플로 필드에 저장되며 적절한 때에 검색할 수 있습니다.

추가 정보

XML View를 만드는 것에 대한 자세한 내용과 매핑 예제를 보려면,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sqlxml3/htm/ssxsdannotations_0gqb.asp를 참조하십시오.

XPath를 사용하여 XML View를 쿼리하기

데이터베이스의 XML View가 생성되면 XPath 쿼리 언어를 사용하여 마치 실제 XML 문서가 있는 것처럼 그 뷰를 쿼리할 수 있습니다. SQLXML은 XPath 1.0 쿼리 언어의 서브셋을 지원합니다. 매핑에 대해 XPath 명령을 실행하면, SQLXML는 그 명령을 함께 결합하여 FOR XML EXPLICIT 문을 만들어서 SQL Server로 보냅니다. 적절한 데이터가 검색되면 매핑에 따라 형태를 만듭니다.

XML View에서 지원하는 XPath의 서브셋에 대한 자세한 내용은 SQLXML Documentation을 참조하십시오.

Updategrams를 사용한 XML View 업데이트

데이터베이스의 XML View에 대해 Updategram을 사용하면 XML View를 통해 SQL Server의 데이터베이스를 수정/삽입/업데이트/삭제할 수 있습니다.

Updategram의 구조

updategram은 updategram의 구문을 형성하는 〈sync〉, 〈before〉 및 〈after〉 요소로 구성된 XML 문서입니다. 각 〈sync〉 vlock에는 〈before〉 블록과 〈after〉 블록이 하나 이상 들어 있습니다. 〈before〉는 레코드 인스턴스의 기존 상태("이전 상태"라고도 함)를 가리킵니다. 〈after〉는 데이터가 변경될 새 상태를 가리킵니다. updategram이 레코드 인스턴스를 삭제할 것인지 삽입할 것인지 업데이트할 것인지는 〈before〉 블록과 〈after〉 블록의 내용에 따라 결정됩니다.

삽입 동작

레코드 인스턴스가 〈after〉 블록에 나타나지만 그에 해당하는 〈before〉 블록에는 없으면 updategram은 삽입 동작입니다. 이 경우, updategram은 〈after〉 블록의 레코드를 데이터베이스에 삽입합니다.

삭제 동작

레코드 인스턴스가 〈before〉 블록에는 있지만 〈after〉 블록에 그에 해당하는 레코드가 없는 경우 updategram은 삭제 동작입니다. 이 경우, updategram은 데이터베이스에서 〈before〉 블록의 레코드를 삭제합니다.

updategram에 지정된 요소가 테이블의 하나 이상의 행에 일치하거나 테이블에서 일치하는 행이 하나도 없으면, updategram은 오류 메시지를 보내고 전체 〈sync〉 블록을 취소시킵니다. updategram의 요소가 삭제할 수 있는 레코드는 한 번에 하나입니다.

업데이트 동작

이미 존재하는 데이터를 업데이트하는 경우 〈before〉 블록과 〈after〉 블록을 모두 지정해야 합니다. updategram은 〈before〉 블록에 지정된 요소를 사용하여 데이터베이스에 이미 존재하는 레코드를 찾아냅니다. 〈after〉 블록에서 그에 해당하는 요소는 업데이트 동작을 실행한 후의 레코드를 가리킵니다.

〈before〉 블록의 요소는 데이터베이스에서 한 테이블 행과만 일치해야 합니다. 요소가 여러 테이블 행과 일치하거나 일치하는 행이 전혀 없다면, updategram은 오류 메시지를 보내고 전체 〈sync〉 블록을 취소시킵니다.

추가 정보

Updategrams을 생성하여 XML View를 통해 데이터를 수정하는데 사용하는 방법에 대한 자세한 내용은 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sqlxml3/htm/updategram_5kkh.asp를 참조하십시오.

XML View를 통해 XML 데이터를 대량으로 로드하기

XML Bulk Load는 XML 데이터를 SQL Server 테이블로 로드하는데 쓰이는 COM 객체입니다. INSERT 문과 OPENXML 함수를 사용하면 XML 데이터를 SQL Server 데이터베이스에 삽입할 수 있습니다. 하지만, bulk load 유틸리티를 사용하면 많은 양의 XML 데이터를 삽입해야 하는 경우 성능이 향상됩니다. XML Bulk Load는 매핑 스키마를 해석하여 XML 데이터를 삽입할 테이블을 찾아낸 다음, XML 데이터를 관계형 테이블에 넣습니다.

소스 XML 문서가 큰 경우에는, 대량 로드 프로세싱에서 전체 문서가 메모리에 로드되지 않습니다. 하지만, XML Bulk Load는 XML 데이터를 스트림으로 해석하여 읽습니다. 이 유틸리티는 데이터를 읽으면서 데이터베이스 테이블을 찾아내고 XML 데이터 소스에서 적절한 레코드를 생성한 다음 그 레코드를 SQL Server로 보내어 삽입합니다.

Bulkload의 작동 방식 및 사용 방법에 대한 자세한 내용은 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sqlxml3/htm/bulkload_7pv0.asp을 참조하십시오.

SQLXML 데이터 액세스 메서드

SQL Server 2000이 나온 이후, SQLXML 기능을 액세스하는 다음 두 가지 새로운 방법이 추가되었습니다.

- SQLXML Managed Classes
- SQLXML Web Services

뿐만 아니라, SQL Server에 대한 HTTP 액세스는 템플릿 내에서 Updategrams을 지원하도록 보강되었습니다.

SQLXML Managed Classes

SQLXML Managed Classes는 Microsoft .NET Framework 내에서 SQLXML 3.0의 기능을 노출합니다. SQLXML Managed Classes를 사용하면 SQL Server의 인스턴스에서 XML 데이터를 액세스하여 데이터를 .NET Framework 환경으로 옮겨 놓고 데이터를 처리한 다음 업데이트된 내용을 SQL Server로 보내어 업데이트를 적용하게 하는 C# 응용 프로그램을 만들 수 있습니다.

SQLXML 관리된 클래스를 사용하는 방법에 대한 자세한 내용은
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sqlxml3/htm/dotnet_0jck.asp을 참조하십시오.

결론

본 백서에서는 SQL Server 2005의 XML 보완 기술을 설명합니다. 서버측 기능에는 XML 스토리지, 인덱싱 및 쿼리 프로세싱을 위한 기본 구현 기능이 포함됩니다. FOR XML이나 OpenXML과 같은 기존의 기능도 보강되었습니다. 클라이언트측 지원은 XML 데이터 형식을 지원하도록 ADO.NET를 보강한 것과 System.Xml에서 서로 다른 소스에서 나온 XML을 쿼리하도록 XQuery를 지원한 것으로 구성됩니다. 뿐만 아니라 지금은 SQLXML 매핑 기술의 웹 릴리스 보강 기능이 SQL Server 2005에 포함되어 있습니다.

서버측 지원과 클라이언트측 지원은 다양한 환경에서 유용합니다. XML 데이터 형식에서는 XML 데이터를 untyped XML 열에 삽입하여 XML 데이터를 저장하는 간단한 메커니즘이 마련됩니다. XML 스키마를 사용하여 typed XML을 정의하면 데이터베이스 엔진이 스토리지와 쿼리 프로세싱을 최적화하고 데이터를 검증하는데 도움이 됩니다.

XML 데이터 형식은 문서 순서를 그대로 유지하며 문서 관리와 같은 응용 프로그램에서 유용합니다. 또한 재귀형 XML 스키마도 처리할 수 있습니다. 알려진 스키마를 사용하는 구조적 데이터의 경우에는 아직도 관계형 데이터 모델이 최상의 선택입니다. 정밀한 쿼리와 업데이트가 별 의미가 없는 상황에서는 [n]varchar(max)도 적합합니다.

SQLXML 매핑 기술은 서버에 있는 테이블에 저장된 관계형 데이터에 대해 XML 중심형 프로그래밍 모델을 사용하려는 모든 경우에 유용합니다. 매핑의 기반은 XML 스키마를 XML 뷰로 정의하는 것입니다. 매핑은 XML 데이터를 테이블에 대량 로드하고 XPath 1.0을 사용하여 테이블을 쿼리하는데 사용할 수 있습니다. 문서 순서는 그대로 유지되지 않으므로 이 매핑 기술은 XML 문서 프로세싱보다는 XML 데이터 프로세싱에 유용합니다.

.NET Framework V2.0 릴리스의 System.Xml에 있는 핵심 XML 클래스를 이용하면 XML을 읽고 쓰고 처리하고 변환할 수 있습니다. V2.0 릴리스에서 성능, 유용성, 형식 설정 및 쿼리 등을 향상시키고 XML을 지원하게 되면서 혁신성, 표준 지원 및 사용 편이성에서 계속 업계 선두 자리를 지키고 있습니다.

서버측 기술과 클라이언트측 기술은 서로 보완하는 관계입니다. 매핑 기술은 서버측 기능을 활용하여 문서 순서 유지와 재귀성 스키마와 같은 기능을 강화하고 더 많은 응용 분야를 찾아낼 수 있습니다. 한편, CLR은 XSLT을 XML 데이터 형식으로 변환하는 것과 같은 기존의 XML 툴을 더욱 강화하고 다채롭게 만듭니다.

샹카르 팔(Shankar Pal)은 SQL Server 엔진 담당 프로그램 매니저이며 서버측 XML 기술 관련 작업을 하고 있습니다. 개인 블로그는 http://blogs.msdn.com/spal입니다.

마크 퍼셀(Mark Fussell)은 마이크로소프트 웹데이터 팀의 수석 프로그램 매니저이며, .NET Framework의 System.Xml와 System.Data 네임스페이스 내의 구성 요소들, Microsoft XML Core Services (MSXML) 및 Microsoft Data Access Components(MDAC) 등을 포함하는 마이크로소프트 데이터 액세스 기술을 개발하고 있습니다. 개인 블로그는 http://weblogs.asp.net/mfussell입니다.

어윈 돌로보프스키(Irwin Dolobowsky)는 웹 데이터 팀의 프로그램 매니저입니다.