Microsoft®
SQL Server™ 2005

ADO.NET 2.0의 XML데이터 형식 지원: SQL Server 2005에서 XML 처리하기

Microsoft®

요약

Microsoft ADO.NET 2.0 및 Microsoft SQL Server 2005의 향상된 XML 지원을 함께 사용하여 응용 프로 그램의 XML 데이터를 보다 쉽게 처리할 수 있는 방법에 대해 알아봅니다.

Contents

소개

Microsoft SQL Server 2005의 큰 변화 중 하나는 XML 데이터 형식이 포함되었다는 점입니다. 이 데이터 형식은 INT 또는 VARCHAR와 같은 최고급 형식이며, SQL Server 2005에서는 일련의 XML 전용 함수를 사용하여 이 데이터 형식을 내부에서 쿼리 및 처리할 수 있습니다. 또한 XML 스키마 컬렉션을 데이터베이스에 저장할 수 있으므로 데이터베이스 기반 스키마 유효성 검사가 가능합니다. 더욱이, SQL Server 2005는 XML 컴퍼지션(SELECT ... FOR XML dialect)의 기능을 크게 확장하고, OpenXML( ) XML 분해 함수를 확장하며, 보다 규모가 작은 분해를 위해 XML 데이터 형식에 대한 새 nodes( ) 함수를 제공합니다.

이와 같이 데이터베이스 서버의 새롭고 향상된 XML 기능으로 인해 Microsoft ADO.NET 2.0의 SqlClient 데이터 공급자 역시 향상되었다는 것은 당연한 일입니다. 또한 ADO.NET DataSet가 XML 형식의 DataColumn을 지원하도록 변경되었으며, System.Data 및 System.Xml 사이의 "통합 지점"이 확대되었습니다. 이 기사에서는 클라이언트에서의 SQL Server 2005 XML 데이터 형식 사용에 대해 알아봅니다.

SQL Server 2005가 생성할 수 있는 XML 출력에는 두 가지 유형이 있습니다. SELECT * FROM AUTHORS FOR XML AUTO 문은 열과 행이 하나씩 있는 행 집합이 아닌 XML 스트림을 생성합니다. 이 출력 형식은 SQL Server 2000에서 바뀌지 않았습니다. XML 스트림 출력이 SQL Server 쿼리 분석기에서 열과 행이 하나씩 있는 행 집합으로 표시되는 것은 쿼리 분석기 도구의 제한 때문일 뿐입니다. 고유한 특정 식별자 이름 "XML_F52E2B61-18A1-11d1-B105-000805F49916B"로 이 스트림을 "일반" 열과 구별할 수 있습니다. 사실 이 이름은 해당 열이 일반 행 집합과 같이 전송되는 대신 클라이언트로 스트림되어야 함을 나타내는 기본 TDS(SQL Server 네트워크 서식인 Tabular Data Stream 파서에 대한 표시기입니다. 클라이언트에서 이 특수한 스트림을 검색하기 위한 특수 메서드는 SqlCommand.ExecuteXmlReader입니다. SQL Server 2005에서 SELECT ... FOR XML dialect는 다양한 방식으로 향상되었습니다. 그 중에서 몇 가지를 언급하면 다음과 같습니다.

1. SQL Server 2000에서는 FOR XML EXPLICIT 모드가 필요했던 대부분의 경우에 사용할 수 있는 새롭고 간편한 FOR XML PATH 모드가 있습니다.
2. TYPE 지시문을 사용하여 스트림 외에 XML 데이터 형식 열을 생성할 수 있습니다.
3. FOR XML 식을 중첩할 수 있습니다.
4. SELECT ... FOR XML을 통해 ROOT 지시문을 사용하여 XML 문서와 XML 조각을 모두 생성할 수 있습니다.
5. 스트림 앞에 표준 XSD 스키마를 추가할 수 있습니다.

ADO.NET 2.0의 관계형 datatype 열거를 참조함으로써 이제 XML이 최고급 관계형 데이터베이스 유형임을 처음으로 나타낼 수 있습니다. System.Data.DbType 및 System.Data.SqlDbType에는 각각 DbType.Xml 및 SqlDbType.Xml에 대한 추가 값이 포함되어 있습니다. 또한 System.Data.SqlTypes 네임스페이스에는 SqlXml이라는 새 클래스가 있습니다. 이 클래스는 XML 형식 값을 기반으로 XmlReader 인스턴스용 팩터리 역할을 합니다. 여기서는 간단한 몇 개의 코드를 사용하여 설명하겠습니다. 다음과 같은 SQL Server 테이블이 있다고 가정해 봅니다.

```
CREATE TABLE xmltab (
  id INT IDENTITY PRIMARY KEY,
  xmlcol XML)
```

다음 ADO.NET 2.0 코드를 사용하여 클라이언트에서 이 테이블에 액세스할 수 있습니다.

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

void GetXMLColumn {
// "Generic Coding..."
 기사에는 구성 파일에서
// 연결 문자열을 가져오는 방법이 나와 있습니다.
string s = GetConnectStringFromConfigFile("xmldb");
using (SqlConnection conn = new SqlConnection(s))
using (SqlCommand cmd = new SqlCommand(
      "select * from xmltab", conn))
 {
  conn.Open(    );
  SqlDataReader rdr = cmd.ExecuteReader(    );
  DataTable t = rdr.GetSchemaTable(    );

  while (rdr.Read(    ))
    {
      SqlXml sx = rdr.GetSqlXml(1);
      XmlReader xr = sx.CreateReader(    );
      xr.Read(    );
      Console.WriteLine(xr.ReadOuterXml(    ));
    }
 }
}
```

참고
프로그래머 코멘트는 샘플 프로그램 파일에는 영문으로 제공되며 기사에는 설명을 위해 번역문으로 제공됩니다.

GetSchemaTable에 의해 생성된 DataTable을 탐색할 때 반환되는 열 메타데이터는 열을 올바르게 식별합니다.

```
ProviderType: 25 (25 = XML)
ProviderSpecificDataType: System.Data.SqlTypes.SqlXml
DataType: System.Xml.XmlReader
DataTypeName:
```

이는 SQL Server에서 기본 제공되는 다른 형식과 같습니다. 이 열의 ".NET 형식"은 XmlReader이며, .NET의 경우에 이는 파일에서 로드되거나 XmlDocument 클래스를 통해 생성되는 XML과 같습니다. XML 데이터 형식 열을 ADO.NET 2.0의 저장 프로시저나 매개 변수화된 문에서 매개 변수로 사용하는 것 역시 간단합니다.

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

void AddARow {
// 구성 파일에서 연결 문자열을 가져옵니다.
string s = GetConnectStringFromConfigFile("xmldb");
using (SqlConnection conn = new SqlConnection(s))
using (SqlCommand cmd = new SqlCommand(
      "insert xmltab(xmlcol) VALUES(@x)", conn))
 {
  conn.Open(    );
  cmd.Parameters.Add("@x", SqlDbType.Xml);

  // 매개 변수 값을 파일에 연결합니다.
  XmlReader xr = XmlReader.Create("somexml.xml");
  cmd.Parameters[0].Value = new SqlXml(xr);
  int i = cmd.ExecuteNonQuery(    );
 }
}
```

XML과 문자열 비교

앞의 코드에 나온 두 방법은 모두 SqlTypes의 SQL Server 전용 데이터 형식을 사용합니다. SqlReader의 보다 일반적인 접근자 메서드인 GetValue(   )를 사용하면 값이 상당히 다릅니다. 열은 XmlReader가 아닌 .NET 문자열 클래스로 표시됩니다. 즉, 메타데이터가 열의 .NET 데이터 형식을 XmlReader로 식별하더라도 열을 XmlReader로 캐스트할 수 없습니다. GetSqlXml(   )을 제외한 접근자를 사용하는 경우 문자열이 반환됩니다.

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

```
void GetXMLColumn {
// 구성 파일에서 연결 문자열을 가져옵니다.
string s = GetConnectStringFromConfigFile("xmldb");
using (SqlConnection conn = new SqlConnection(s))
using (SqlCommand cmd = new SqlCommand(
      "select * from xmltab", conn))
 {
  conn.Open(    );
  SqlDataReader rdr = cmd.ExecuteReader(    );
  // "System.String"을 인쇄합니다.
  Console.WriteLine(rdr[1].GetType(    ));

  // 실패했습니다. 잘못된 캐스트입니다.
  XmlReader xr = (XmlReader)rdr[1];

  // 제대로 작동합니다.
  string s = (string)rdr[1];
 }
}
```

SqlReader.GetProviderSpecificValue( ) 메서드를 사용하더라도 문자열이 반환됩니다. GetProviderSpecificFieldType이 System.Sql.Types.SqlXml을 올바르게 반환하므로, 이는 다소 예외적인 것입니다. 공급자의 현재 베타 버전 문제인 듯하므로, 여기서는 이 메서드를 사용하지 않겠습니다.

```
// System.Data.SqlTypes.SqlXml
Console.WriteLine(rdr.GetProviderSpecificFieldType(1));

// System.Data.SqlTypes.SqlString
Object o = rdr.GetProviderSpecificValue(1);
Console.WriteLine(o.GetType(    ));
```

SqlClient는 XML 매개 변수를 위한 대칭 기능을 제공합니다. 따라서 String 데이터 형식과 함께 이러한 매개 변수를 사용할 수 있습니다. XML 형식을 사용해야 하는 문자열(NVARCHAR)을 전달할 수 있는 것은 SQL Server에서 VARCHAR 또는 NVARCHAR을 XML 데이터 형식으로 자동 변환하는 기능이 제공되기 때문입니다. 또한 다음 예제에서 볼 수 있는 것처럼 이러한 변환은 클라이언트 쪽에서도 수행될 수 있습니다. 저장 프로시저를 insert_xml이라고 가정하면 string/NVARCHAR을 XML로 자동 변환하는 두 방법은 모두 작동합니다.

```
-- T-SQL stored procedure definition
CREATE PROCEDURE insert_xml(@x XML)
AS
INSERT xmltab(xmlcol) VALUE(@x)

// 클라이언트 쪽 코드
```

```
using System;
using System.Data;
using System.Data.SqlClient;

void InsertXMLFromClient {
// 구성 파일에서 연결 문자열을 가져옵니다.
string s = GetConnectStringFromConfigFile("xmldb");
using (SqlConnection conn = new SqlConnection(s))
using (SqlCommand cmd1 = new SqlCommand(
      "INSERT xmltab(xmlcol) VALUES(@x)", conn))
using (SqlCommand cmd2 = new SqlCommand(
      " insert_xml", conn))
 {
  string str = "<somedoc/>";

  conn.Open(    );

  // 서버 쪽 변환
  cmd1.Parameters.Add("@x", SqlDbType.NVarChar);
  cmd1.Parameters[0].Value = str;
  cmd1.ExecuteNonQuery(    );

  // 클라이언트 쪽 변환도 작동합니다.
  cmd2.CommandType = CommandType.StoredProcedure;
  cmd2.Parameters.Add("@x", SqlDbType.Xml);
  cmd2.Parameters[0].Value = s;
  cmd2.ExecuteNonQuery(    );
 }
}
```

문서, 조각 및 FOR XML 지원

SQL Server 2005의 XML 데이터 형식은 XML 문서와 XML 문서 조각을 모두 지원합니다. 조각은 다중 최상위 수준 요소와 최소한의 텍스트 노드를 포함할 수 있다는 점에서 문서와 다릅니다. 형식화된 XML 열/변수/매개 변수를 사용하면 조각 허용 여부를 DOCUMENT(조각이 허용되지 않음) 또는 CONTENT(조각이 허용됨) 지정을 사용하여 결정할 수 있습니다. 기본값은 CONTENT이며, 형식화되지 않은 XML이 조각을 허용합니다. 다음 T-SQL 코드는 조각 지원을 보여 줍니다.

```
CREATE TABLE xmltab (
  id INT IDENTITY PRIMARY KEY,
  xmlcol XML)
GO

-- insert a document
INSERT xmltab VALUES('〈doc/〉')
-- fragment, multiple top-level elements
INSERT xmltab VALUES('〈doc/〉〈doc/〉')
-- fragment, bare text node
INSERT xmltab VALUES('Hello World')
-- even this fragment works
INSERT xmltab VALUES('〈doc/〉sometext')
```

XML 조각은 또한 SELECT ... FOR XML에 의해서도 생성됩니다. SELECT job_id, min_lvl, max_lvl FROM jobs FOR XML AUTO 문은 다음 출력을 생성합니다. 다중 루트 요소를 확인하십시오.

```
〈jobs job_id="1" min_lvl="10" max_lvl="10" /〉
〈jobs job_id="2" min_lvl="200" max_lvl="250" /〉
〈jobs job_id="3" min_lvl="175" max_lvl="225" /〉
〈jobs job_id="4" min_lvl="175" max_lvl="250" /〉
〈!-- some jobs rows deleted for compactness --〉
```

SqlXml을 사용하여 문서와 조각을 지원합니다. SqlXml CreateReader( ) 메서드는 다음과 같이 새 XmlReaderSettings 클래스를 사용하여 조각을 지원하는 XmlReader를 항상 만듭니다.

```
// SqlXml.CreateReader의 pseudocode
   Stream stm = stm; // 열에서 채워진 스트림(코드 생략)
   XmlReaderSettings settings = new XmlReaderSettings( );
   settings.ConformanceLevel = ConformanceLevel.Fragment;
   XmlReader xr = XmlReader.Create(
     stm, String.Empty, null, null,settings);
```

XmlReader를 동일한 방법으로 구성하는 경우 입력 매개 변수에서 XML 조각을 사용할 수 있습니다. SqlXml 형식을 사용할 때 조각 지원이 기본 제공되지만 조각을 포함하는 XmlReader를 처리할 때는 주의해야 합니다. XmlReader.GetOuterXml( )을 호출하면 첫 번째 조각만 제공됩니다. XmlReader를 배치하여 이후의 조각을 가져오려면 XmlReader Read 메서드를 다시 호출해야 합니다. 이에 대한 설명은 이 기사의 뒷부분에 나와 있습니다.

T-SQL의 "SELECT ... FOR XML"은 열과 행이 하나씩 있는 행 집합이 아닌 XML 스트림을 생성했습니다. 또한 XML의 표준 serialization이 아니라 이진 형식으로 XML이 제공되었습니다. 형식의 차이와 "SELECT ... FOR XML"이 항상 조각을 생성한다는 사실 때문에 이를 사용하려면 특수한 메서드가 필요했습니다. SqlClient는 이를 위해 SqlCommand.ExecuteXmlReader라는 공급자 전용 메서드를 구현합니다.

SQL Server 2000 및 ADO 1.0/1.1에서는 문자열 연결이 필요한 상당히 까다로운 해결 방법을 사용하지 않는 한 ExecuteXmlReader를 사용하여 FOR XML 쿼리의 결과를 얻어야 합니다. SQL Server 2005에서는 향상된 FOR XML이 제공되고 데이터 형식으로 XML이 지원되기 때문에 ExecuteXmlReader를 사용하여 SQL Server에서 단일 XML 스트림만 가져오면 됩니다. SQL Server 2000용으로 작성된 모든 코드가 이 메서드를 사용했으므로, ADO.NET 2.0에서 이 메서드가 지원 및 향상됩니다.

이전 버전과 마찬가지로 ExecuteXmlReader를 사용하여 "FOR XML" 쿼리에서 스트림을 검색할 수 있습니다. 또한 이 메서드를 사용하면 일반 SELECT 문에 의해 생성된 XML 데이터 형식 열을 검색할 수 있습니다. 한 가지 주의해야 할 사항은 일반 SELECT 문이 둘 이상의 행을 반환할 경우 ExecuteXmlReader는 첫 번째 행의 내용만 반환한다는 것입니다. 다음 예제는 이전 예제와 동일한 테이블을 사용하여 이러한 사실을 보여 줍니다.

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

void UseExecXmlReader {
// 구성 파일에서 연결 문자열을 가져옵니다.
string s = GetConnectStringFromConfigFile("xmldb");
using (SqlConnection conn = new SqlConnection(s))
using (SqlCommand cmd1 = new SqlCommand(
      "select * from pubs..authors for xml auto,root('root')", conn))
using (SqlCommand cmd2 = new SqlCommand(
      "select * from pubs..authors for xml auto", conn))
using (SqlCommand cmd3 = new SqlCommand(
      "select * from xmltab", conn))
 {
  conn.Open(    );
  // 문서를 포함합니다.
  XmlReader xr1 = cmd1.ExecuteXmlReader(    );
  // 조각을 포함합니다.
  XmlReader xr2 = cmd2.ExecuteXmlReader(    );
  // xmltab의 첫 번째 행 내용만 포함합니다.
  XmlReader xr3 = cmd3.ExecuteXmlReader(    );
  // 그런 다음 XmlReaders를 사용합니다.
  xr1.Dispose(    ); xr2.Dispose(    ); xr3.Dispose(    );
 }
}
```

ADO.NET 2.0에서 XML을 가져오는 방법에 대한 논의를 끝내기 전에 다양한 사용 시나리오의 XmlReader 콘텐츠 수명에 대해 언급하는 것이 도움이 될 것입니다. 또한 XmlReader의 수명을 살펴보면 SqlClient가 수행하는 버퍼링과 성능 극대화를 위한 이 데이터의 사용 방법을 이해할 수 있습니다. XmlReader는 리소스를 사용하는데, 이러한 리소스를 해제하려면 SqlConnection, SqlCommand 및 SqlDataReader와 마찬가지로 Close( ) 또는 Dispose( ) 메서드를 호출해야 합니다. SqlDataReader를 통해 XML 열을 읽는 경우에는 각 행에 할당된 XmlReader가 존재할 수 있습니다. 동일한 행의 열에서 뒤로 이동하거나 다음 행으로 이동하는 것을 지원하기 위해 XmlReader의 내용은 메모리에서

버퍼링됩니다. SqlCommand의 CommandBehavior.SequentialAccess를 사용할 때는 전체 XmlReader가 메모리에서 버퍼링되지 않지만, 이 액세스 메서드는 보다 주의하여 사용해야 합니다. CommandBehavior.SequentialAccess를 사용할 때는 행 집합에서 다음 열로 이동하기 전에 열과 연관된 XmlReader를 모두 소비해야 합니다. 다음 열로 이동한 후에도 XmlReader는 유효한 것으로 표시되지만 해당 Read( ) 메서드를 호출해도 데이터가 생성되지 않습니다. ExecuteReader 대신에 ExecuteXmlReader 또는 ExecuteScalar를 사용할 때는 이 동작에 크게 신경쓰지 않아도 되지만, 여기서도 XmlReader를 닫거나 삭제해야 합니다.

클라이언트에서 XML 스키마 지원 사용

SQL Server 2005는 엄격한 형식의 XML을 지원합니다. 이는 XML이 XML 스키마나 XML 스키마 집합을 따라야 함을 의미합니다. SQL Server에서 XML 스키마 컬렉션을 사용하여 이 지원을 설정합니다. XML 스키마 컬렉션은 다른 SQL Server 개체처럼 정의되며 XML 스키마는 SQL Server에 저장됩니다. T-SQL DDL CREATE 문과 XML 스키마 컬렉션은 다음과 같은 방식으로 사용됩니다.

```
CREATE XML SCHEMA COLLECTION books_xsd
AS
-- one or more XML schemas here
GO

CREATE TABLE typed_xml (
  id INT IDENTITY PRIMARY KEY,
  -- require books_col content to be schema-valid
  books_col XML(books_xsd)
)
-- validated here
INSERT typed_xml VALUES(' <!-- some document --> ')
-- validated here too
UPDATE typed_xml
  SET books_col.modify(' <!-- some XQuery DML --> ')
  WHERE id = 1
```

클라이언트에서 SQL Server 2005에 엄격한 형식의 XML 데이터를 사용하면 클라이언트가 아니라 서버에서 유효성 검사가 수행됩니다. 예를 들어, 앞의 예제에 나온 AddARow 메서드를 사용하여 행을 typed_xml 테이블에 추가하면 유효성 검사가 수행되기 전에 데이터는 네트워크를 통과하여 SQL Server로 보내집니다. 그러나 약간의 작업만으로 SQL Server XML SCHEMA COLLECTION에서 XML 스키마를 검색하여 클라이언트에 이러한 스키마를 보관함으로써 클라이언트 쪽 유효성 검사를 수행할 수 있습니다. 이렇게 하면 사용자나 웹 서비스가 SQL Server에 저장하기 위해 잘못된 스키마의 XML을 클라이언트로 보내지 못하도록 함으로써 작업이 다소 줄어들기는 하지만, 두 가지 주의해야 할 사항이 있습니다. 첫째로, SQL Server에서 가져온 XML 스키마 정보를 사용하는 것은 캐시된 클라이언트 쪽 메타데이터를 사용하는 것과 같습니다. 누군가가 T-SQL ALTER XML SCHEMA 문을 사용하여 스키마 컬렉션을 변경했을 수도 있고, 나아가 스키마 컬렉션을 제거하고 다시 만들었을 수 있습니다. 이러한 경우 클라이언트 쪽 검사는 무의미해집니다. CREATE/ALTER/DROP XML SCHEMA DDL 문에서 새 EVENT NOTIFICATION을 사용하여 이를 뒤늦게 발견하는 것을 방지할 있습니다. 그런 다음 쿼리 알림과 함께

SqlNotificationRequest를 사용할 때와 비슷하게 사용자 지정 코드를 사용하여 Service Broker(서비스 브로커) SERVICE를 모니터링할 수 있습니다. EVENT NOTIFICATION은 SQL Server 2005의 새로운 기능이지만 저의 이전 기사인 Query Notifications in ADO.NET 2.0 에서 언급했던 쿼리 알림과 비슷합니다. 둘째로, 클라이언트 쪽에서 XML 스키마 유효성 검사를 수행하더라도 SQL Server는 서버에서 검사를 다시 실행합니다. 현재로서는 SQL Server 스키마 형식 XML의 해당 인스턴스가 유효한 스키마이므로 직접 검사할 필요가 없음을 SQL Server에 알릴 수 있는 방법이 없습니다.

클라이언트 쪽 XML 스키마 유효성 검사를 수행하려면 T-SQL 함수 xml_schema_namespace( )를 사용하여 SQL Server XML SCHEMA COLLECTION을 검색합니다. XML 스키마 컬렉션이 데이터베이스 스키마 범위이므로 이를 위해서는 XML 스키마 컬렉션 이름과 데이터베이스 스키마 이름이 필요합니다. 이러한 이름을 프로그램으로 하드코드하거나 클라이언트 쪽 행 집합 메타데이터에서 추출할 수 있습니다. 위의 SqlDataReader GetSchemaTable 메서드를 사용하거나 새 SqlMetaData 클래스를 사용하면 특정 열과 연관된 스키마 컬렉션 이름을 쉽게 가져올 수 있습니다. 다음은 간단한 코드 예제입니다.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.Sql;

void GetCollectionInfo {
// 구성 파일에서 연결 문자열을 가져옵니다.
string s = GetConnectStringFromConfigFile("xmldb");
using (SqlConnection conn = new SqlConnection(s))
using (SqlCommand cmd = new SqlCommand(
      "select books_col from typed_xml", conn))
{
  conn.Open( );
  // SQL Server 메타데이터만 가져옵니다.
  SqlDataReader rdr = cmd.ExecuteReader(CommandBehavior.SchemaOnly);

  SqlMetaData md = rdr.GetSqlMetaData(0);
  string database = md.XmlSchemaCollectionDatabase;
  string schema = md.XmlSchemaCollectionOwningSchema;
  string collection = md.XmlSchemaCollectionName;
}
}
```

검색할 XML 스키마 컬렉션을 파악했으므로 T-SQL 함수를 사용하여 해당 컬렉션을 클라이언트 쪽 XmlSchemaSet로 가져올 수 있습니다. 이 예제는 또한 XmlReader를 사용하여 조각을 검색하는 방법도 보여 줍니다. 이 작업이 필요한 것은 XML SCHEMA COLLECTION에 둘 이상의 XML 스키마가 있을 수 있기 때문입니다. xml_schema_namespace 함수는 컬렉션의 모든 XML 스키마를 조각으로 반환합니다.

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

```
using System.Xml;
using System.Xml.Schema;

void GetSchemaSet {
// 구성 파일에서 연결 문자열을 가져옵니다.
string s = GetConnectStringFromConfigFile("xmldb");
using (SqlConnection conn = new SqlConnection(s))
using (SqlCommand cmd = new SqlCommand(
      "SELECT xml_schema_namespace(N'dbo',N'books_xsd')", conn))
 {
  XmlSchemaSet ss = new XmlSchemaSet(    );
  conn.Open(    );
  SqlDataReader rdr = cmd.ExecuteReader(    );
  rdr.Read(    );
  XmlReader xr = rdr.GetSqlXml(0).CreateReader(    );

  do
  {
    ss.Add(XmlSchema.Read(xr, null));
    xr.Read(    );
  }
  while (xr.NodeType == XmlNodeType.Element);
 }
}
```

XmlSchemaSet를 사용하면 다음과 같이 클라이언트 쪽 XML 유효성 검사 코드를 Add 루틴에 통합할 수 있습니다. 그러면 클라이언트 쪽 유효성 검사가 완료됩니다.

```
void ValidateAndStore(XmlSchemaSet ss)
{
  // XmlSchemaSet를 XmlReader에 연결합니다.
  XmlReaderSettings settings = new XmlReaderSettings(    );
  settings.Schemas = ss;
  string s = GetConnectStringFromConfigFile("xmldb");
  using (XmlReader xr = XmlReader.Create(
    "file://c:/temp/somefile.xml", settings))
  // 구성 파일에서 연결 문자열을 가져옵니다.
  using (SqlConnection conn = new SqlConnection(s))
  using (SqlCommand cmd = new SqlCommand(
    "insert typed_xml values(@x)", conn))
  {
    try
    {
      conn.Open(    );
```

```
        // 스키마가 유효하지 않으면 여기에서 예외(exception)를 throw해야 합니다.
        cmd.Parameters.AddWithValue("@x", new SqlXml(xr));
        int i = cmd.ExecuteNonQuery(    );
      }
      catch (Exception e)
      {
        Console.WriteLine(e.Message);
      }
    }
  }
```

클라이언트 쪽 XML 스키마 유효성 검사가 모든 응용 프로그램에 필요한 것은 아니지만, 응용 프로그램에서 필요할 때 사용할 수 있다는 것을 안다면 도움이 될 것입니다. 또한 다음과 같이 SELECT...FOR XML에서 새로운 SQL Server 2005 XMLSCHEMA 옵션을 사용할 때도 XML 스키마가 생성됩니다.

```
SELECT * FROM authors FOR XML AUTO, ELEMENTS, XMLSCHEMA
```

현재 베타에서 실제로 작업을 수행하는 데 약간의 제한이 있기는 하지만 조각을 분해하고 클라이언트 쪽에서 이러한 쿼리 유형에 대해 XmlSchemaSet를 검색할 수도 있습니다.

결론

이 기사에서는 새로운 XML 형식과 FOR XML 기능을 사용하여 Microsoft ADO.NET SqlClient에서 Microsoft SQL Server의 XML을 소비하는 방법과 SqlClient를 사용하여 XML을 SQL Server 테이블에 삽입하는 방법에 대해 설명했습니다. 사실 DataSet에는 여기서 언급하지 않은 향상된 XML 기능이 있으므로, 조만간 ADO 2.0 DataSet의 향상된 기능에 대한 기사가 제공될 것입니다. SQL Server XML 데이터 사용에 대한 자세한 내용은 Shankar Pal, Mark Fussell 및 Irwin Dolobowsky의 MSDN 온라인 기사 XML Support in Microsoft SQL Server 2005 에서 확인할 수 있으며 SELECT...FOR XML의 향상된 기능에 대한 자세한 내용은 Michael Rys의 MSDN 온라인 기사 What's New in FOR XML in Microsoft SQL Server 2005 에서 확인할 수 있습니다. 또한 .NET 2.0의 System.Xml에 대한 자세한 내용은 Mark Fussell의 What's New in System.Xml for Visual Studio 2005 and the .NET Framework 2.0 Release 에서 확인할 수 있습니다. 데이터 모델이 그 어느 때보다도 긴밀하게 통합되므로, XML은 다양한 수준에서 ADO.NET 2.0 및 SqlClient와 통합된다고 할 수 있을 것입니다.

[저자소개]

Bob Beauchemin은 DevelopMentor의 강사, 코스 저자 및 데이터베이스 교육 과정 섭외자로서, 데이터 중심 배포 시스템 개발자, 프로그래머 및 관리자로 25년 이상의 경력이 있습니다. Microsoft Systems Journal 및 SQL Server Magazine 등에서 ADO.NET, OLE DB 및 SQL Server에 대한 기사를 썼으며 A First Look at SQL Server 2005 for Developers 및 Essential ADO.NET 의 저자입니다.