Rob Lans is werkzaam bij The Competence Group.

XML-integratie in Microsoft SQL Server 2005

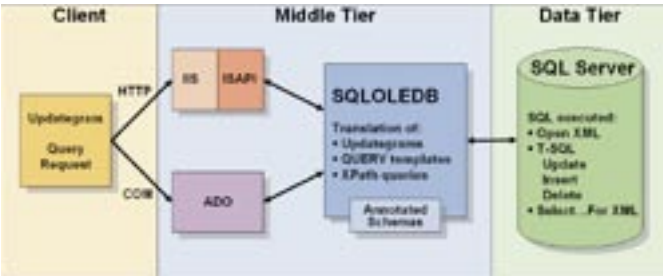
MICROSOFT ZET GROTE STAP MET XML DATATYPE EN XQUERY IN SQL SERVER 2005

Al enkele jaren heeft Microsoft het gebruik van XML hoog in het vaandel staan. XML moet de integratie van verschillende producten en platformen gemakkelijker maken. Nieuwe producten of nieuwe versies van bestaande producten bezitten dan ook allemaal in meer of mindere mate XML-gerelateerde functionaliteit. Ook SQL Server gaat mee in deze trend. In Service Pack 3 werd de XML-functionaliteit van SQL Server 2000 al uitgebreid. In SQL Server 2005 wordt opnieuw een grote stap op dit gebied gezet. De opvallendste nieuwe features op dit gebied zijn de introductie van XML als eigen datatype en de ondersteuning van XQuery (een vraagtaal voor XML-documenten) door de queryprocessor.

In dit artikel zullen we eerst een kort overzicht geven van de XML-mogelijkheden die we reeds van SQL Server 2000 kennen. Daarna gaan we gedetailleerder kijken naar XML als datatype in SQL Server. Wat zijn de implicaties en waarom is het een verbetering ten opzichte van de oude situatie? En als we XML in de database hebben, hoe kunnen we dan die gegevens manipuleren? Waar komt XQuery in dit plaatje kijken? Dit artikel vereist enige kennis van XML en XPath.

XML in SQL Server 2000

Het schema in afbeelding 1 schetst grofweg hoe SQL Server gebruikt wordt in combinatie met XML. In veel gevallen moeten we denken aan e-commerce-achtige applicaties waarbij gegevensuitwisseling met klanten en leveranciers via XML-documenten geschiedt. In dit soort situaties kunnen we SQL Server direct via HTTP benaderen of bijvoorbeeld vanuit ASP via het ADO-objectmodel. In beide gevallen wordt een groot deel van het werk uitgevoerd door de SQLOLEDB-laag. Deze laag kent bijvoorbeeld het verschijnsel UpdateGrams, een XML-document waarin zowel een 'before state' als een 'after state' is opgenomen. Aan de hand hiervan genereert de SQLOLEDB-provider de juiste update-, insert- en delete-commando's. Een andere functionaliteit die in het schema in afbeelding 1 wordt genoemd bij de 'middle tier' is XPath. Als we gebruikmaken van http-access kunnen we SQL Server benaderen via een URL. In deze URL kunnen we een XPath-expressie gebruiken om aan te geven in welke informatie we geïnteresseerd zijn.



Afbeelding 1. Het gebruik van SQL Server in combinatie met XML.

XPath-expressies zijn een manier om in een XML-document aan te geven welke node of nodes of welke attributen we willen zien.

In SQL Server zelf zien we ook de nodige ondersteuning voor XML. Het duidelijkst merken we deze ondersteuning aan de FOR XML-clausule van het SELECT-statement. Als laatste onderdeel van een SELECT-statement, dus nog na de ORDER BY, kunt u de clausule FOR XML meegeven. Dit heeft tot gevolg dat het resultaat van de query niet in het standaardformaat van een tabel (recordset) terugkomt, maar in de vorm van XML. Via XSL kunt u hier opmaak (of andere transformaties) op loslaten en het resultaat zo direct in een browser tonen.

De FOR XML-clausule kent drie vormen, namelijk RAW, AUTO en EXPLICIT. RAW is het eenvoudigste in het gebruik, maar ook de meest beperkte optie qua functionaliteit. De optie EXPLICIT is juist het lastigste in het gebruik, maar we kunnen de uitvoer wel volledig sturen.

```
SELECT Customers.CustomerID, Companyname, OrderID, OrderDate
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
WHERE Customers.CustomerID = 'ALFKI'
FOR XML AUTO

------------------------------------------------------------ Resultaat

<Customers CustomerID="ALFKI" Companyname="Alfreds Futterkiste">
   <Orders OrderID="10643" OrderDate="1997-08-25T00:00:00"/>
   <Orders OrderID="10692" OrderDate="1997-10-03T00:00:00"/>
   <Orders OrderID="10702" OrderDate="1997-10-13T00:00:00"/>
   <Orders OrderID="10835" OrderDate="1998-01-15T00:00:00"/>
   <Orders OrderID="10952" OrderDate="1998-03-16T00:00:00"/>
   <Orders OrderID="11011" OrderDate="1998-04-09T00:00:00"/>
</Customers>

(6 row(s) affected)
```

Codevoorbeeld 1.

De optie RAW heeft tot gevolg dat voor elk record in het resultaat van de query een element <*row*> wordt gemaakt met voor elke geselecteerde kolom een attribuut. Als we AUTO gebruiken hebben de elementen de naam van de tabel. Bovendien zullen er subelementen worden gemaakt indien we een JOIN opnemen in de query. Daarnaast kunnen we ook nog eens met de extra optie ELEMENTS aangeven dat we de kolommen als elementen willen zien (element centric) in plaats van als attributen (attribute centric). De derde mogelijkheid is het gebruik van de EXPLICIT-mode. Hiermee kunnen we elk gewenst formaat maken via het gebruik van universal tables; zie books online voor details. In SQL Server 2005 zijn de mogelijkheden van FOR XML nog aanzienlijk uitgebreid. Zo is er een nieuwe vorm, PATH, bijgekomen, kunnen we well-formed XML maken, kunnen we zelf de elementnamen bepalen, is er ondersteuning voor NULL-waarden en kunnen we gebruikmaken van het nieuwe XML-datatype. Het gaat te ver om al deze mogelijkheden hier te bespreken. Codevoorbeeld 1 is een voorbeeld van een query met de FOR-XML clausule en het bijbehorende resultaat.

Naast het gebruik van FOR XML om uitvoer in de vorm van XML te krijgen in plaats van als tabel, maken we veel gebruik van de OPENXML rowset-functie in combinatie met de stored procedure sp_xml_preparedocument. Codevoorbeeld 2 gaat uit van een XML-document met een root element PurchaseOrder. Daarbinnen vinden we één of meer Order-elementen. Een Order-element heeft een OrderId-attribuut en enkele elementen zoals OrderDate en Customer. Bovendien vinden we binnen een Order-element één of meer OrderDetail-elementen met elementen voor het product, de prijs, het aantal en de belasting.

Het zojuist beschreven XML-document wordt als parameter meegegeven aan de stored procedure van de code in codevoorbeeld 2. Het eerste dat we moeten doen als we XML willen verwerken, is de stored procedure sp_xml_preparedocument aanroepen. Deze stored procedure 'parst' de XML en slaat de inhoud intern op in een boomstructuur. We krijgen een handle naar de XML terug. Deze handle gebruiken we vervolgens bij de aanroep van OPENXML als eerste parameter. OPENXML is een rowset-functie die de boomstructuur 'vertaalt' naar een tabelformaat. Dat houdt in dat we OPENXML kunnen gebruiken in de FROM-clausule van het SELECT-statement zoals we in codevoorbeeld 2 zien gebeuren.

De tweede parameter van OPENXML is een XPath-expressie. XPath-expressies zijn vergelijkbaar met paden van bestanden. Gebruikmakend van de hiërarchische opbouw van XML duidt een XPath-expressie een element of een attribuut aan binnen een XML-document. Zo geeft de tweede parameter van de OPENXML-functie van de eerste query in codevoorbeeld 2 aan dat de OPENXML alle Order-elementen moet verwerken die vallen onder het root element PurchaseOrder. Net zo worden in het tweede SELECT-statement alle OrderDetail-elementen verwerkt.
De extra WITH-clausule die volgt op de OPENXML-functie geeft aan hoe de elementen uit het XML-document vertaald moeten worden naar kolommen in de relationele tabel en welke SQL-datatypes gebruikt moeten worden. De derde parameter van de OPENXML-functie heeft hier ook betrekking op. Met deze parameter kunt u aangeven of er sprake is van element centric mapping of van attribute centric mapping. Zie voor meer details de Books Online van SQL Server.

De laatste opdracht in codevoorbeeld 2 is de aanroep van *sp_xml_removedocument*. Daar waar de *sp_xml_preparedocument* de XML 'parst' en in een boomstructuur intern opslaat, geeft de *sp_xml_removedocument* de gebruikte resources weer vrij.

XML-datatype

Uit de bovenstaande voorbeelden volgt dat we in SQL Server 2000 voortdurend bezig zijn vertaalslagen te maken tussen de relationele

```
--Stored procedure die een XML order aan de database toevoegt
CREATE PROC ImportOrder @xml ntext
AS
DECLARE @idoc int
EXEC sp_xml_preparedocument @idoc OUTPUT, @xml
-- Haal de order gegevens uit de XML en zet de gegevens in de Order tabel
  INSERT Orders
    SELECT * FROM OpenXML(@idoc, 'PurchaseOrder/Order', 3)
    WITH (OrderNo integer,
   OrderDate datetime,
   RetailerID varchar(30),
   SupplierID varchar(30),
   Customer varchar(40),
   DeliveryAddress varchar(255),
   OrderStatus varchar(40)
   )
 -- Haal de details uit de XML en zet de gegevens in de OrderDetails tabel
  INSERT OrderDetails
    SELECT * FROM OpenXML(@idoc, 'PurchaseOrder/Order/OrderDetail', 2)
    WITH (OrderNo integer '../@OrderNo',
   ProductID int,
   Quantity int,
   UnitPrice money,
   Tax money
   )

EXEC sp_xml_removedocument @idoc
```

Codevoorbeeld 2.

manier van dataopslag, en de XML-manier van datarepresentatie dan wel data-uitwisseling. De gegevens zitten als kolommen in tabellen en via de FOR XML-clausule maken wij er XML van in de uitvoer. En als we XML aangereikt krijgen, gebruiken we de OPENXML om er op een relationele manier mee te kunnen werken. Als we de XML willen opslaan zoals het aangeleverd wordt, kunnen we het in een kolom van het type *text* of *ntext* kwijt. SQL Server heeft dan echter geen weet van het feit dat het hier XML betreft. De enige manier om dan bijvoorbeeld te zoeken naar een element of attribuut met een bepaalde waarde, is via het gebruik van full-text indices.

SQL Server 2005 introduceert het XML-datatype om bovenstaand probleem op te lossen. Dat wil zeggen dat we kolommen kunnen maken van het type XML. Uiteraard kunnen we ook variabelen en parameters van dit type maken. Het XML-datatype bestaat in twee varianten, typed en untyped XML. Een untyped XML-kolom is eigenlijk gewoon een tekstkolom in de zin dat er tot 2 GB aan tekst in opgeslagen kan worden. In het geval van een typed XML-kolom kennen we ook een XSD (XML schema definition) aan de kolom toe. Deze definitie werkt alsof het een CHECK-constraint betreft. Zodra we data aan de kolom willen toekennen, zal SQL Server controleren of deze data wel voldoen aan de definitie zoals beschreven in de XSD. Zo niet, dan worden de data geweigerd. Op deze manier creëren we dus een echte XML datakolom die heel specifiek hoort bij de business solution die de database ondersteunt.

Naast het hierboven beschreven voordeel van typed XML, is er nog een tweede voordeel. SQL Server maakt gebruik van de kennis over de opbouw van de XML door de data niet als één lange string op te slaan zoals in het geval van untyped XML, maar er intern een meer relationele manier van opslaan voor te bedenken. Zo kan een getal opgeslagen worden als integer in plaats van als tekst. Op die manier kan veel geheugenruimte bespaard worden.

Uiteraard leveren de hierboven beschreven eigenschappen van het XML-datatype al de nodige voordelen op. Echt leuk wordt het pas met het feit dat het XML-datatype vijf methods kent waarmee we

de XML-data kunnen manipuleren. Deze methods maken allemaal gebruik van XQuery. Voordat we de methods nader gaan bekijken, zullen we eerst XQuery bespreken.

XQuery

XQuery is een vraagtaal voor XML, net zoals SQL een vraagtaal is voor relationele databases. Het is een W3C standaard waarvan de laatste draft te vinden is op http://www.w3.org/XML/Query. XQuery maakt gebruik van XPath-expressies om elementen en attributen aan te duiden. Het formaat van het resultaat van een XQuery-expressie wordt volledig bepaald door de expressie zelf. We kunnen zelf nieuwe elementen en attributen definiëren. Met XPath is dat niet mogelijk omdat we slechts bestaande elementen kunnen 'aanwijzen'. Ook deze functionaliteit van XQuery is te vergelijken met het SELECT-statement van SQL waar we met expressies en aliassen in de select-lijst als het ware een nieuwe (virtuele) tabel maken. Centraal bij XQuery staat het zogenaamde FLWR (flower) statement, waarbij de letters respectievelijk staan voor de FOR-, LET-, WHERE- en RETURN-clausule.We zullen XQuery aan de hand van een aantal voorbeelden illustreren.

Afbeelding 2 is een screenshot van de *XQuery builder*, een onderdeel van de SQL Server Management Studio van SQL Server 2005. Helaas is de XQuery builder in beta 2 verdwenen. Laten we hopen dat hij in latere versies weer terugkomt. Het voorbeeld is gebaseerd op de Instructions-kolom van de ProductPlan-tabel die u kunt vinden in AdventureWorks, de nieuwe voorbeelddatabase van Microsoft. Het screenshot bevat twee delen, een min of meer grafische representatie van de query en de XQuery-expressie zelf. De expressie begint met de declaratie van de namespace *ns*. Dit is niets anders dan een verwijzing naar de XSD van de Instructions-kolom.

Het FLWR-statement begint bij de For-clausule die vergelijkbaar is met de For Each loop uit Visual Basic en C#. Via het dollarteken geven we aan dat we een variabele maken ($WorkCenter1) en één voor één worden alle WorkCenter-elementen hier aan toegekend. Voor elk gevonden element wordt vervolgens het returngedeelte uitgevoerd. Alles wat na de return tussen accolades staat zal geëvalueerd worden, alles wat buiten de accolades staat wordt letterlijk in het resultaat opgenomen. In het voorbeeld creëren we dus nieuwe ToolsAndMaterials-elementen. Elk ToolsAndMaterials-element bevat alle tools- en alle materials-elementen die een child-element zijn van de node waar onze variabele $WorkCenter1 naar verwijst. De expressies tussen de accolades zijn 'gewone' XPath-expressies.

De letter W van FLWR staat voor *where*. Deze clausule werkt identiek aan de gelijknamige clausule in het SQL SELECT-statement. Door de regel

```
Where $WorkCenter1/@LaborHours > 3
```

op te nemen in het XQuery-statement uit het voorbeeld in afbeelding 2, en wel tussen de FOR- en de RETURN-clausule in, worden alleen nog die WorkCenter-elementen verwerkt waarvan het attribuut LaborHours groter is dan 3.

Afbeelding 2. XQuery builder.

We kunnen XQuery pas echt vergelijken met het SQL SELECT-statement als we in staat zijn *joins* te schrijven. Deze mogelijkheid bestaat. De werkwijze is sterk te vergelijken met de manier waarop *joins* beschreven zijn in de ANSI-89 standaard. De manier van *joins* schrijven bestond toen uit het opgeven van een lijst van tabellen in de FROM-clausule, om vervolgens in de WHERE-clausule de *join*-condities op te nemen. In latere standaards is het JOIN-keyword geïntroduceerd. In een XQuery-statement kunt u meer FOR-clausules opnemen. Het resultaat is een cartesisch product van de elementen die uit de verschillende FOR's komen. U gebruikt de WHERE-clausule om aan te geven welke combinaties van elementen verwerkt moeten worden. De WHERE zou er dus bijvoorbeeld uit kunnen zien als:

```
Where $Klant/Klantnummer = $Order/Klantnummer
```

Een andere manier om een *join* te maken is door gebuik te maken van nested queries. Net zoals we in SQL een correlated subquery kunnen maken, mogen we ook bij XQuery nesten. Zoals eerder gezegd wordt in de RETURN-clausule alles wat tussen accolades staat geëvalueerd. Tussen deze accolades mogen we weer een volledig XQuery-statement plaatsen waarbij we in de WHERE-clausule mogen refereren aan variabelen die we in de buitenste expressie hebben gedeclareerd. Op http://www.devx.com/xml/Article/8066 staat een tutorial over XQuery FLWR-statements.

XML-methods

Zoals eerder gezegd zijn in SQL Server 2005 vijf methods gedefinieerd voor het XML-datatype. Deze methods zijn query, value, exists, modify en nodes. Ze zijn gemaakt om de data in de XML-kolom te kunnen manipuleren en maken allemaal gebruik van XQuery-statements. We kunnen gebruikmaken van deze methods via de bekende punt-syntax. We selecteren de XML-kolom en zetten de method daar meteen achteraan met een punt tussen de kolomnaam en de method; zie codevoorbeeld 3.

Het SQL SELECT-statement uit het voorbeeld selecteert de CatalogDescription-kolom uit de ProductModel-tabel. We zijn echter niet geïnteresseerd in de volledige kolom, maar laten een XQuery los op de inhoud van de kolom. In deze XQuery maken we zelf een nieuw Product-element met een attribuut ProductModelID (we noemen deze syntax computed constructor syntax) en een element Weight als die aanwezig is. Dit XQuery-statement wordt als parameter meegegeven aan de query-method van de XML-kolom. Het gevolg is uiteraard wel dat deze XQuery voor elk record in de ProductModel-tabel één maal wordt uitgevoerd. In ons voorbeeld is dat zes maal.
Tevens gebruiken we in codevoorbeeld 3 de exist-method. Ook deze method vraagt een XQuery-statement mee als parameter. De

```
use adventureworks

SELECT CatalogDescription.query('
declare namespace p1="http://schemas.microsoft.com/sqlserver/2004/07/
adventure-works/ProductModelDescription";
  <Product ProductModelID="{ (/p1:ProductDescription/@ProductModelID)[1] }">
    {
      /p1:ProductDescription/p1:Specifications/Weight
    }
  </Product>
') as Result
FROM Production.ProductModel
WHERE CatalogDescription.exist('
declare namespace p1="http://schemas.microsoft.com/sqlserver/2004/07/
adventure-works/ProductModelDescription";
       /p1:ProductDescription
') = 1
```

Codevoorbeeld 3.

```
--insert a new location - <Location 1000/>.
UPDATE Production.ProductModel
SET Instructions.modify('
  declare namespace MI="http://schemas.microsoft.com/sqlserver/2004/07/
          adventure-works/ProductModelManuInstructions";
insert <MI:Location LocationID="1000"  LaborHours="1000"  LotSize="1000" >
           <MI:step>Do something using <MI:tool>hammer</MI:tool></MI:step>
         </MI:Location>
  as first
  into   (/MI:root)[1]
')
WHERE ProductModelID=7

go
SELECT Instructions
FROM Production.ProductModel

go
-- Now replace manu. tool in location 1000
UPDATE Production.ProductModel
SET Instructions.modify('
  declare namespace MI="http://schemas.microsoft.com/sqlserver/2004/07/
          adventure-works/ProductModelManuInstructions";
  replace value of (/MI:root/MI:Location/MI:step/MI:tool)[1]
  with   "screwdriver"
')
WHERE ProductModelID=7

go
SELECT Instructions
FROM Production.ProductModel
```

Codevoorbeeld 4.

method geeft een 1 terug als het meegegeven statement resultaat oplevert en anders 0, parallel aan de werking van het Exists-keyword van SQL. Deze method is dus bij uitstek geschikt voor gebruik in de WHERE-clausule van een SQL SELECT-statement. We zien dus in codevoorbeeld 3 dat we delen van de opgeslagen XML kunnen opvragen, geformatteerd op een ad hoc gedefinieerde manier, en dat we kunnen filteren op basis van elementen of attributen binnen de XML. Dit biedt duidelijk meer mogelijkheden dan het opslaan van XML in tekstkolommen.

Met de query-method kunnen we weliswaar onze eigen XML-elementen maken en dus precies het gewenste resultaat samenstellen, het resultaat blijft wel XML. Het is in sommige gevallen nodig om geen XML terug te krijgen maar 'gewone' scalaire waardes, zoals een naam of het aantal namen dat voorkomt in de XML. De value-method voorziet in deze behoefte. We geven een XQuery-expressie mee die slechts één waarde oplevert en definiëren met een tweede parameter het datatype van die waarde. Het resultaat is een normaal SQL-attribuut. Ook de nodes-method moeten we zoeken in de hoek van het vertalen van XML naar relationele data. Als een XML-kolom meer gelijke elementen bevat, kunnen we met de nodes-method zorgen dat we een record krijgen voor elk element. Eén record met een XML-kolom waarin drie gelijke elementen voorkomen, kunnen we omvormen tot drie records met een XML-kolom met slechts één element.

Nog interessanter wordt het met de modify-method die ons in staat stelt ook aanpassingen in de XML aan te brengen. Het spreekt bijna voor zich dat we deze method alleen kunnen gebruiken in de SET-clausule van een SQL UPDATE-statement. Bovendien hebben we speciale XML_DML-syntax nodig om aan te geven welke verandering waar doorgevoerd moet worden. Net zoals in SQL zijn de mogelijkheden ook hier beperkt tot insert, delete en replace (is update). Het gaat te ver om de volledige XML_DML hier te bespreken. Codevoorbeeld 4 laat het volgende zien: in de eerste query wordt met het XML_DML insert-statement een nieuw Location-element toegevoegd in de root van de XML van het record met ProductModelID 7. De tweede keer dat we de modify-method aanroepen gebruiken we het 'replace value of'-statement om de waarde van het tool-element te wijzigen van 'hammer' naar 'screwdriver'. Zie voor meer voorbeelden de books online.

Tot slot

Het moge duidelijk zijn dat Microsoft met de introductie van het xml-datatype en de introductie van XQuery in de query-engine een grote stap heeft gezet op het gebied van integratie van XML en relationele data. Voor Windows-applicaties die SQL Server als onderliggende database gebruiken levert dat niet veel extra's op. Zoals in het begin al gezegd zullen vooral e-commerce-applicaties voordelen ondervinden van de integratie van XML in de database. Daarnaast biedt deze XML-integratie ook vele mogelijkheden voor developers van webservices.

Peter ter Braake is productspecialist SQL Server bij Compu'Train. Zijn e-mailadres is pbraake@computrain.nl. Compu'Train (www.computrain.nl) is de grootste CPLS (voorheen CTEC Gold) in Nederland en marktleider op het gebied van SQL Server-trainingen.

Nuttige internetadressen

- http://www.devx.com/xml/Article/8066
- http://www.w3schools.com/xquery
- http://www.w3.org/XML/Query
- http://msdn.microsoft.com/xml
- http://msdn.microsoft.com/sql