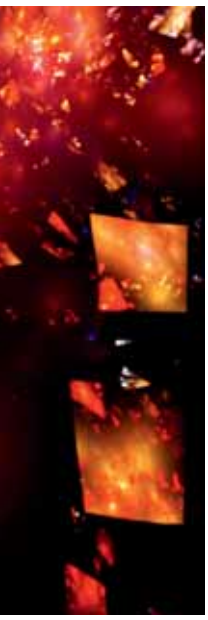


Drie inleesmethoden worden tegen elkaar uitgezet

XML als bron in een Microsoft datawarehouse

Henk Brands

Hoe kan de inhoud van XML bestanden ingelezen worden in een Microsoft datawarehouse? In tegenstelling tot bijvoorbeeld een CSV bestand bevat een XML bestand behalve data ook structuur. Hoe is deze structuur naar relationele tabellen te vertalen? Met SQL Server Integration Services? Of zijn er ook andere manieren?

Met XML (eXtensible Markup Language) is het mogelijk om op een flexibele manier gegevens te beschrijven en op te slaan[1]. Als bron voor het ETL-proces zorgt XML voor een aantal voordelen boven bijvoorbeeld een regulier CSV bestand. Om te beginnen is XML een manier om data gestructureerd op te slaan. XML beschrijft behalve de data ook de relaties tussen de elementen in de data. De inhoud van een XML bestand is leesbaar voor de mens en (bij goed gekozen elementnamen) zelfbeschrijvend.

```
<supermarkten>
  <supermarkt id="1">
    <naam>Appie</naam>
    <adres>
      <straat>Dorpstraat 1</straat>
      <postcode>1234 AB</postcode>
    </adres>
    <producten>
      <product id="1">
        <naam>Hagelslag</naam>
        <verkocht>5</verkocht>
        <prijs>0.98</prijs>
      </product>
      <product id="2">
        <naam>Cola</naam>
        <verkocht>16</verkocht>
        <prijs>1.42</prijs>
      </product>
    </producten>
  </supermarkt>
</supermarkten>
```

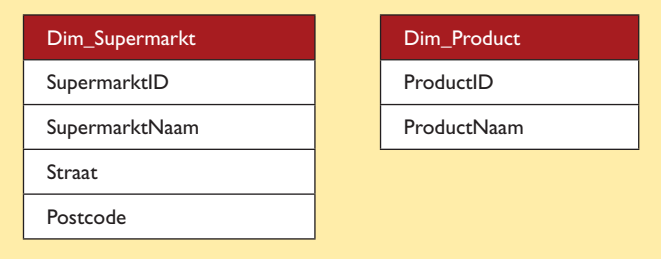
Afbeelding 1: XML voorbeeld.

Verder is het mogelijk om XML te valideren tegen een XML schema (zie kader). Hierbij is het dus mogelijk om een contract op te stellen tussen de aanbieder en de ontvanger van een XML bestand. Dit zorgt ervoor dat er geen misverstand kan bestaan over hoe het XML bestand eruit moet zien.

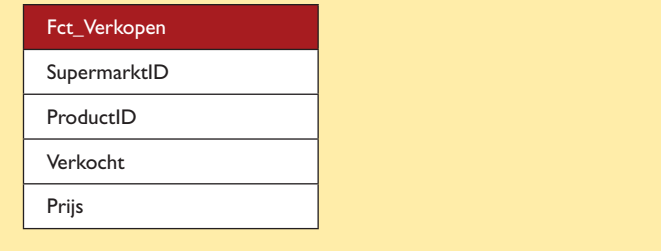
Drie methoden

Welke manieren zijn er om XML in te lezen in een Microsoft datawarehouse? In dit artikel worden drie methoden beschreven om kleine XML bestanden in te lezen in een SQL Server database. Wanneer echt grote XML bestanden (ongeveer 1 GB en groter) ingelezen moeten worden is het aan te raden om eens naar SQL XML Bulk Load te kijken[2]. Dat zal in dit artikel echter niet verder behandeld worden. Als afsluiter is een consumententabel te vinden waarin de verschillende methoden tegen elkaar worden uitgezet.

Bij het inlezen van XML bestanden wordt gebruik gemaakt van Microsoft SQL Server Integration Services (SSIS) en de Microsoft SQL Server database engine (hierna aangeduid als SQL Server). De eerst beschreven methode maakt voor het gehele proces gebruik van SSIS en slaat alleen het resultaat op in de database. De twee andere methoden gebruiken XML functionaliteit die door SQL Server wordt aangeboden. De drie methoden worden beschreven aan de hand van hetzelfde voorbeeld: een fictieve door de auteur verzonnen buurtsupermarkt met de naam Appie. Afbeelding 1 toont een eenvoudig XML bestand, dat de supermarkt beschrijft en de producten die hier verkocht zijn. Het doel is om dit XML bestand te vertalen naar een eenvoudig datawarehouse. De data met de relaties kunnen vertaald worden naar twee dimensietabellen en één feitentabel: Dim_Supermarkt, Dim_Product en Fct_Verkopen. Afbeelding 2 en afbeelding 3 geven respectievelijk weer hoe deze tabellen eruit zien.



Afbeelding 2: Dimensietabellen.



Afbeelding 3: Feitentabel.

SQL Server Integration Services

De eerste methode, en misschien wel de meest voor de hand liggende, is het volledig gebruik maken van SSIS. SSIS kent de XML Source component. Zoals de naam aangeeft is deze component geschikt om XML bestanden in te lezen. Bij het configureren neem je een verwijzing naar het in te lezen XML bestand en het bijbehorende schema op. De metadata (eigenschappen zoals naam, datatype en lengte van de kolommen) van de XML component worden aangemaakt met behulp van de informatie in het schema.
Het doet vermoeden dat het ingestelde schema wordt gebruikt voor de validatie van de gegevens, en dat het schema hier rechtstreeks voor gebruikt wordt. Door een wijziging in het schema te maken kan de validatie worden aangepast. Dit werkt echter niet compleet naar verwachting. In werkelijkheid is het zo simpel als reeds hiervoor beschreven. Het schema wordt gebruikt om de metadata aan te maken. De validatie van het XML bestand dat wordt ingelezen wordt gedaan tegen de standaard metadatavalidatie van SSIS en niet rechtstreeks tegen het opgegeven schema. Als het schema wijzigt levert dit een design- en runtime-fout van de package op.
Als er geen schema beschikbaar is kan SSIS deze ook genereren op basis van een bestaand XML bestand. De ervaring leert echter dat het verstandiger is om zelf een schema te maken, omdat het door SSIS gegenereerde schema zo ruim is opgezet dat het altijd de lading dekt. Een strak ingericht schema zorgt ervoor dat de datatypen exact op maat opgenomen kunnen worden.

In afbeelding 4 is afgebeeld hoe een SSIS dataflow er uitziet, waarbij het supermarkt XML bestand wordt ingelezen. Te zien is dat er vanuit de XML Source een viertal stromen (groene pijlen: product, producten, supermarkt en adres) komt. Elke stroom staat voor een element dat subelementen bevat (een parent element). Zo bevat de adresstroom de velden straat en postcode. Omdat de uiteindelijke tabellen onderdelen van meerdere stromen nodig hebben, zijn deze aan elkaar gekoppeld door middel van Merge Joins.
Per stroom is een error flow (rode pijl) beschikbaar. Wanneer er in het element prijs een stuk tekst staat, terwijl het ingestelde schema aangeeft dat dit een getal moet zijn, zal alleen in de productstroom een fout optreden. Als er een error flow is ingesteld (dit is niet het geval in het gegeven voorbeeld) dan gaan de afgekeurde gegevens door die stroom heen. Dit betekent echter niet dat het hele XML bestand wordt afgekeurd, hier moeten extra stappen voor worden ondernomen. Om echte schemavalidatie te bewerkstelligen is de XML Source component dan ook niet geschikt. Voeg hiervoor in de control flow een XML Task toe. Hierin is het mogelijk om het XML bestand te valideren en een nette melding te laten geven waarom de validatie niet geslaagd is.

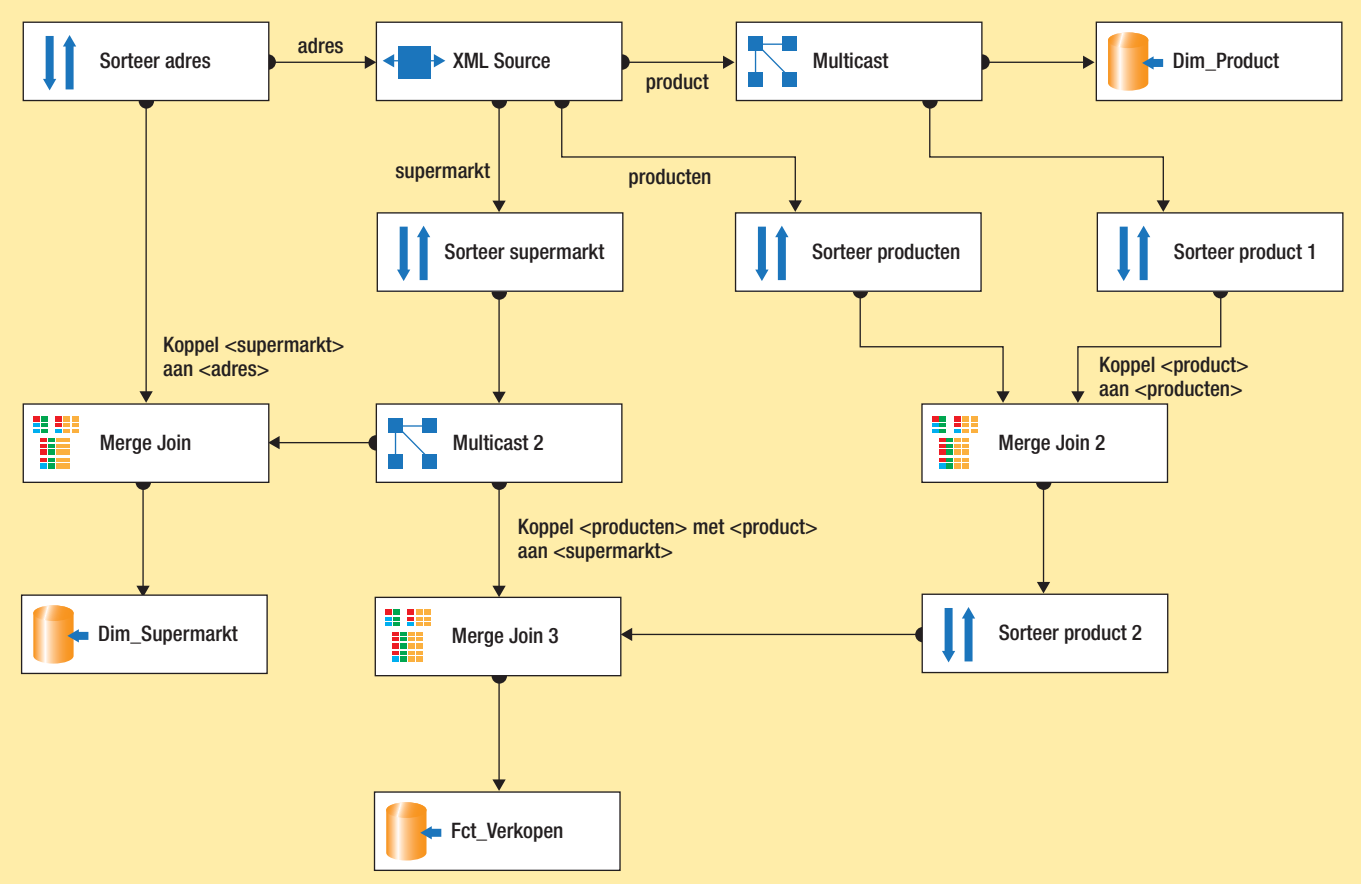
Een gevaar bij het gebruiken van de XML Source is dat de dataflow al snel zeer complex en onderhoudsgevoelig wordt bij een wat uitgebreider XML bestand. Uit de praktijk blijkt dat wanneer er een wijziging in het XML bestand plaatsvindt, de dataflow er door alle fouten in de metadata uitziet als een mooi verlichte kerstboom op kerstavond. Om dit lichtfestijn te voorkomen is het ook een optie om de aparte stromen rechtstreeks in tussen- of stage-tabellen in te lezen en de relaties hiertussen door middel van een query in SQL Server op te lossen.
Afbeelding 5 geeft weer hoe een dataflow die de XML bestanden in stage-tabellen laadt, er uitziet. Hoewel er nog een extra dataflow nodig is om de stage-tabellen in de dimensietabellen en feitentabel te laden, is deze dataflow een stuk beter te onderhouden. Een keerzijde is dat wanneer het XML bestand veel parentelementen bevat er tevens veel stage-tabellen nodig zijn.

OPENXML

De tweede methode OPENXML is een standaard functie in SQL Server die al sinds SQL Server 2000 beschikbaar is. In dit artikel wordt het XML datatype in combinatie met de OPENXML functie gebruikt. Maar voordat deze functie nader wordt toegelicht,

XML schema

Het XML schema (ook wel aangeduid als XML Schema Definition Language – XSD) is een W3C standaard die de structuur van een XML bestand beschrijft. Het is de opvolger en 'vervanger' van het DTD. Hiermee wordt van een valid XML bestand (XML die voldoet aan de W3C standaard) gegarandeerd dat deze aan een bepaald formaat voldoet. Het XML schema kan daarom dienen als contract tussen de gegevensleverancier en gegevensconsument. Er wordt gesproken over typed XML wanneer een gegeven XML bestand voldoet aan een schema.

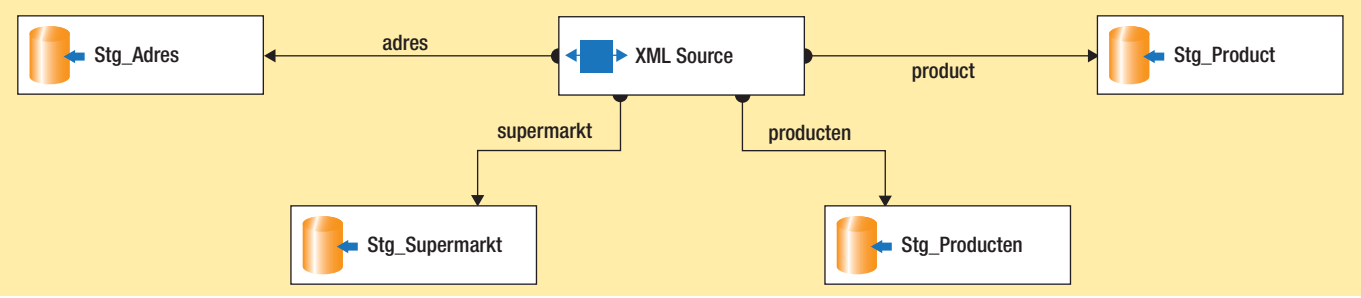


Afbeelding 4: SSIS XML Source dataflow.

maken we eerst een kort uitstapje naar dit XML datatype van SQL Server.

In SQL Server 2005 heeft Microsoft dit datatype en hiermee de nieuwe mogelijkheden geïntroduceerd. Je kunt nu bijvoorbeeld een tabel maken met een kolom van het type XML, wat je in staat stelt om valid XML in een tabel te laden. Dit betekent dat er per rij een compleet XML bestand opgeslagen kan worden. Behalve untyped XML is ook typed XML aan te maken. Typed XML is XML dat gekoppeld is aan een schemadefinitie. In SQL Server kun je dan ook een schema aanmaken en dit toewijzen aan het type XML. Het schema maak je met het SQL commando *CREATE XML SCHEMA COLLECTION*. Hierbij hoef je alleen de naam van het schema op te geven, gevolgd door de schemadefinitie in tekst. In afbeelding 6 staat een weergave van een tabel waarin de XML van de voorbeeldsupermarkt samen met de bestandsnaam ingeladen kan worden. XmlContent is hier van het type XML en moet voldoen aan het XML schema Supermarkten. Het schema Supermarkten is hier niet gedefinieerd, maar sluit aan bij het XML bestand uit afbeelding 1. Dit betekent dat elk XML bestand dat geladen wordt in de tabel XmlFiles voldoet aan het schema Supermarkten.

Om de bestanden in de tabel te laden kan gebruik worden gemaakt van een SQL expressie in bijvoorbeeld SSIS. Afbeelding 7 geeft een voorbeeld van het bestand 'supermarkt.xml' waarvan de bestandsnaam en de inhoud geladen worden in de tabel XmlFiles. Wanneer het bestand niet voldoet aan het schema Supermarkten dan zal de INSERT falen. Dit is bijvoorbeeld met een TRY CATCH blok netjes af te handelen.



Afbeelding 5: SSIS XML Source met stage-tabellen.

```
CREATE TABLE XmlFiles
(
      XmlFilename varchar(50) NULL
,     XmlContent xml(Supermarkten) NULL
)
```

Afbeelding 6: XML tabel.

```
INSERT INTO  XmlFiles(XmlFilename, XmlContent)
SELECT       'supermarkt.xml', *
FROM         OPENROWSET
             (
                    BULK 'C:\supermarkt.xml'
             ,      SINGLE_BLOB
             ) AS x
```

Afbeelding 7: Insert XML file.

Hoe krijgen we deze gegevens uit de XmlFiles tabel en in onze tabellen? Hier komt OPENXML weer om de hoek kijken. Volgens de MSDN[3] voorziet OPENXML in een relationele laag over de XML. Het is mogelijk om, nadat de XML is geprepareerd, via XPath[4] expressies de data uit het bestand te halen. In afbeelding 8 staat een codevoorbeeld over het gebruik van OPENXML. Het eerste opvallende in de code is dat hier untyped XML gebruikt wordt. Dit komt omdat OPENXML geen typed XML accepteert. De stored procedure sp_xml_preparedocument is een verplichte aanroep. Deze procedure geeft een handle terug die aan OPENXML als eerste argument mee gegeven dient te worden. Het tweede argument bevat een XPath expressie die de af te drukken node identificeert. Via de WITH clausule kun je aangeven welke elementen of attributen weergegeven dienen te worden. Dit dient respectievelijk te gebeuren door de kolomnaam, datatype en Xpath expressie op te geven. Als de handle naar het XML bestand niet meer nodig is, oftewel als er niets meer met XML gedaan hoeft te worden, dient door middel van sp_xml_removedocument de handle opgeheven te worden. Het is via OPENXML alleen mogelijk om één XML document per keer in te lezen. Dit betekent dat wanneer de tabel XmlFiles uit afbeelding 7 meerdere records zou bevatten, de SQL code uit afbeelding 8 voor elk record uitgevoerd moet worden.

XQuery als derde methode

Als je gebruik maakt van het eerder beschreven XML datatype, zijn er extra mogelijkheden ontstaan om deze XML te bevragen. Hiervoor heeft Microsoft de XQuery functies geïntroduceerd in SQL Server[5]. Deze functieset bevat behalve de mogelijkheden voor normale bevraging ook extra functionaliteit. Zo kun je hiermee ook functies zoals het FLWOR statement gebruiken en XML genereren of bewerken voordat deze in de tabellen geladen wordt. Bij het maken van een query kun je XML retourneren, maar ook een relationele resultaatset opleveren die is gebaseerd op de XML. Dat laatste kan OPENXML ook, maar waarin XQuery zich onderscheidt is de mogelijkheid om dit voor een gehele set aan XML documenten ineens te doen. Dit betekent dat wanneer de tabel XmlFiles uit afbeelding 7 meerdere XML bestanden bevat, deze ineens kunnen worden ingelezen in de feiten- en dimensietabellen. Op deze manier hoef je het niet voor

```
DECLARE      @UntypedXML AS xml
DECLARE      @idoc AS int

SELECT       @UntypedXML = XmlContent
FROM         XmlFiles
WHERE        XmlFilename = 'supermarkt.xml'

EXEC sp_xml_preparedocument @idoc OUTPUT,
                                      @UntypedXML

INSERT INTO  Dim_Supermarkt
SELECT       *
FROM         OPENXML (@idoc,
                      '/supermarkten/supermarkt')
WITH         (
               SupermarktID int '@id'
             , SupermarktNaam varchar(50)
                                      './naam'
             , Straat varchar(50)
                                 './adres/straat'
             , Postcode varchar(50)
                               './adres/postcode'
             )

INSERT INTO  Dim_Product
SELECT       *
FROM         OPENXML (@idoc, '/supermarkten/
                    supermarkt/producten/product')
WITH         (
               ProductID int './@id'
             , ProductNaam varchar(50) './naam'
             )

INSERT INTO  Fct_Verkopen
SELECT       *
FROM         OPENXML (@idoc, '/supermarkten/
                    supermarkt/producten/product')
WITH         (
               SupermarktID int '../../@id'
             , ProductID int './@id'
             , Verkocht int './verkocht'
             , Prijs decimal(18,2) './prijs'
             )
EXEC sp_xml_removedocument @idoc
```

Afbeelding 8: OPENXML.

```
INSERT INTO  Dim_Supermarkt
SELECT       XmlContent.value('(./supermarkten/supermarkt/@id)[1]',
                              'int')                                    AS SupermarktID
,            XmlContent.value('(./supermarkten/supermarkt/naam)[1]',
                              'varchar(50)')                            AS SupermarktNaam
,            XmlContent.value('(./supermarkten/supermarkt/adres/straat)[1]',
                              'varchar(50)')                            AS Straat
,            XmlContent.value('(./supermarkten/supermarkt/adres/postcode)[1]',
                              'varchar(50)')                            AS Postcode
FROM         XmlFiles

INSERT INTO  Dim_Product
SELECT       a.b.value('@id', 'int')                                   AS ProductID
,            a.b.value('naam[1]', 'varchar(50)')                       AS ProductNaam
FROM         XmlFiles
CROSS APPLY  XmlContent.nodes('supermarkten/supermarkt/producten/product') AS a(b)

INSERT INTO  Fct_Verkopen
SELECT       XmlContent.value('(./supermarkten/supermarkt/@id)[1]', 'int')
                                                                       AS SupermarktID
,            a.b.value('@id', 'int')                                   AS ProductID
,            a.b.value('verkocht[1]', 'int')                           AS Verkocht
,            a.b.value('prijs[1]', 'decimal(18,2)')                    AS Prijs
FROM         XmlFiles
CROSS APPLY  XmlContent.nodes('supermarkten/supermarkt/producten/product') AS a(b)
```

Afbeelding 9: XQuery.

ieder bestand afzonderlijk te doen. Dit kan voordelen hebben wanneer je, bijvoorbeeld door middel van een aggregatie, standgegevens in de feitentabel wilt opslaan. Dit kan dan in één slag opgelost worden, terwijl bij OPENXML eerst de data compleet moeten zijn.

Afbeelding 9 geeft de voorbeeldcode van hoe XQuery te gebruiken is. Voor Dim_Product en Fct_Verkopen wordt een CROSS APPLY clausule gebruikt. Dit is noodzakelijk om alle producten en alle feiten van die producten uit te lezen. Het zorgt ervoor dat dit voor elk record in de XmlFiles tabel gebeurt.

Consumententabel

Wat is nu de beste methode? De punten in afbeelding 10 kunnen helpen om een keuze te maken. Hoewel er enkele minnen in de tabel staan betekent dit niet dat er een slecht(st)e methode bij zit. Kies voor elke taak het juiste gereedschap. Maar het is wel fijn om te weten welke gereedschappen je allemaal in de gereedschapskoffer hebt.

Henk Brands (henkb@infosupport.com) is manager Competence Center Business Intelligence & Data Warehousing bij Info Support.

Noten

1. *http://www.w3.org/XML/*
2. *http://msdn.microsoft.com/en-us/library/ms171993.aspx*
3. *http://msdn.microsoft.com/en-us/library/ms186918(SQL.90).aspx*
4. *http://www.w3.org/TR/xpath*
5. *http://www.w3.org/TR/xquery/*

	SSIS	OPENXML	XQUERY
Untyped XML	+	+	+
Typed XML	+/-	-	+
Onderhoudbaarheid	-	+	+
Meerdere bestanden inlezen	+	-	+
Eenvoud	+	+/-	+/-

Afbeelding 10: Consumententabel.