

Introduktion till OpenXML i SQL Server

Av: Kristofer Gäfvert

Copyright Information

Copyright © 2003 - 2004 Kristofer Gafvert (kgafvert@ilopia.com). No part of this publication may be transmitted, reproduced, or republished in any way, without written permissions by the author.

Innehållsförteckning

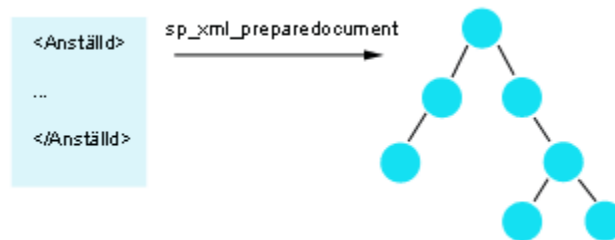
Copyright Information	2
Innehållsförteckning	3
Innan vi börjar	4
Läsa ett XML-dokument	4
Ett mer avancerat exempel	6
Lägga in i en tabell	8

Innan vi börjar

Lika viktigt som att få XML utifrån data i SQL Server, är det att utifrån ett XML-dokument lyckas lägga in data i SQL Server. Detta använder vi en funktion som heter *OpenXML* för, men för att det ska fungera, så finns det ett par saker vi måste tänka på. Innan vi använder *OpenXML* måste XML-dokumentet analyseras och byggas upp som en trädstruktur, med de olika noderna, attributen, texterna med mera i XML-dokumentet, i minnet på datorn. För detta används en lagrad procedur som heter *sp_xml_preparedocument*. Denna lagrade procedur returnerar ett handtag (eng. handle) som används för att komma åt detta dokument, som nu finns i minnet med en egen struktur, som SQL Server (och *OpenXML*) kan tolka.

Eftersom XML-dokumentet finns i minnet, betyder det att vi gärna vill ta bort det när vi är klar med det. Om vi inte gör det riskerar vi att göra slut på minnet i datorn, vilket leder till en minnesläcka. Så även för detta finns det en lagrad procedur, som heter *sp_xml_removedocument*. Om vi nu skulle glömma att ta bort XML-dokumentet från minnet, så kommer det ändå inte finnas där för all framtid, utan tills anslutningen (den anslutning som användes för att lägga in det i minnet) avbrutits.

Om ni får för er att kopiera exempel direkt från denna text, istället för att skriva av, och det inte fungerar, så kan det bero på att jag använder en annan teckentabell i detta dokument än vad SQL Query Analyzer använder. Detta kommer att leda till att SQL Query Analyzer inte kommer att godkänna användningen av dessa tecken.



Läsa ett XML-dokument

Nu när vi vet lite om bakgrunden, så är det dags att börja skriva lite T-SQL. Men vi kommer inte att gå direkt på att lägga in data i tabeller, utan istället börjar vi med att göra lite vanliga *Select* frågor. Dessa frågor liknar de vi gör för existerande tabeller i en databas, men nu ska vi istället göra det mot det XML-dokument vi har.

1: **Declare** @resultDoc int, @xmlDoc varchar(4000)
2:

```
3: Set @xmlDoc = N'  
4: <?xml version="1.0" encoding="ISO-8859-1"?>  
5: <Anstallda>  
6:   <Person>  
7:     <Förnamn>Anna</Förnamn>  
8:     <Efternamn>Andersson</Efternamn>  
9:     <Ålder>26</Ålder>  
10:  </Person>  
11:  <Person>  
12:    <Förnamn>Sivert</Förnamn>  
13:    <Efternamn>Persson</Efternamn>  
14:    <Ålder>35</Ålder>  
15:  </Person>  
16: </Anstallda>  
17:  
18: EXEC sp_xml_preparedocument @resultDoc OUTPUT, @xmlDoc  
19:  
20: SELECT *  
21: FROM OpenXML(@resultDoc, N'/Anstallda/Person', 2)  
22: WITH (  
23:   Förnamn VARCHAR(15),  
24:   Efternamn VARCHAR(15),  
25:   Ålder INTEGER  
26: )  
27:  
28: EXEC sp_xml_removedocument @resultDoc  
29:
```

Det första ni nog funderar över är varför XML-koden är här, ska inte den vara i en fil? Jo, den kan mycket väl finnas i en fil, men för att saker och ting ska bli enklare, så utelämnas detta (normalt sett är denna kod förmodligen sparad som en lagrad procedur, som exekveras från en applikation, som skickar med XML-dokumentet som text, och inte hänvisar till en fil).

På första raden deklarerar vi två variabler. Den sista av dessa kommer att innehålla XML-dokumentet. Denna är deklarerad som varchar, och som ni ser så är den också väldigt stor. För detta lilla exempel hade den inte behövt vara så stor, men tänkas bör på att antalet tecken blir otroligt stort när det blir lite större XML-dokument. Om man då anger en för liten storlek på datatypen, kommer detta resultera i underliga fel. Till exempel om jag enbart hade angivit storleken 200 i ovanstående exempel, hade vi fått detta felmeddelande:

XML parsing error: The following tags were not closed: Anstallda, Person, Efternamn.

Anledningen till felmeddelandet är för att när den storlek jag angivit har uppnåtts, kommer den strunta i allt som kommer därefter, vilket resulterar att en massa text i XML-dokumentet fattas (vilket inkluderar sluttaggarna för elementen "Anstallda", "Person" och "Efternamn").

På rad 3 sparar vi XML-dokumentet (som är på raderna 4 till 16) i variabeln @xmlDoc.

På rad 18 kör vi den lagrade proceduren *xp_xml_preparedocument* som läser in XML-dokumentet i minnet, formaterar det, och sedan returnerar ett handtag till det, som vi sparar i variabeln @resultDoc.

Därefter, på raderna 20 till 26 gör vi själva frågan, som kommer resultera i detta:

Utskrift

	Förnamn	Efternamn	Ålder
1	Anna	Andersson	26
2	Sivert	Persson	35

OpenXML har som vi ser tre parametrar, och syntaxen ser ut såhär:

```
OPENXML(idoc int [in],rowpattern nvarchar[in],[flags byte[in]])  
[WITH (SchemaDeclaration | TableName)]
```

Den första parametern (idoc) är handtaget till det som skapades med *sp_xml_preparedocument*. Den andra parametern (rowpattern) används för att tala om vad som är intressant. I vårt fall säger vi åt den att starta i /Anställda/Person (den första / kan utelämnas). Som flagga satte vi nummer 2, vilket betyder att vi var intresserade av element. Om vi hade satt en 1:a istället, hade vi fått attribut (vi saknar attribut helt i vårt exempel, så det hade resulterat i en massa NULL). Det går också kombinera dessa två nummer till en 3:a, vilket gör att vi får både attribut och element. Därefter använder vi WITH för att tala om vilka kolumner vi vill ha, samt vilken datatyp det är. I vårt fall vill vi ha Förnamn, Efternamn och Ålder från XML-filen, så därför skriver vi det.

På rad 28 rensar vi upp efter oss, och tar bort det som vi lade i minnet.

Ett mer avancerat exempel

Nu när vi har använt SQL för att få ut det vi vill från ett XML-dokument, är det dags att ta ett litet mer avancerat exempel.

```
1: Declare @resultDoc int, @xmlDoc varchar(4000)  
2:  
3: Set @xmlDoc = N'  
4: <?xml version="1.0" encoding="ISO-8859-1"?>  
5: <Anställda>  
6:   <Avdelning Id="3">  
7:     <Person Id="1">  
8:       <Förnamn>Anna</Förnamn>  
9:       <Efternamn>Andersson</Efternamn>  
10:      <Ålder>26</Ålder>  
11:     </Person>  
12:     <Person Id="2">  
13:       <Förnamn>Bertil</Förnamn>  
14:       <Efternamn>Johansson</Efternamn>  
15:       <Ålder>48</Ålder>  
16:     </Person>  
17:   </Avdelning>
```

```
18: <Avdelning Id="5">
19:   <Person Id="3">
20:     <Förnamn>Sivert</Förnamn>
21:     <Efternamn>Persson</Efternamn>
22:     <Ålder>35</Ålder>
23:   </Person>
24: </Avdelning>
25: </Anställda>'
26:
27: EXEC sp_xml_preparedocument @resultDoc OUTPUT, @xmlDoc
28:
29: SELECT *
30: FROM OpenXML(@resultDoc, N'Anställda/Avdelning/Person', 3)
31: WITH (
32:   PersonId INTEGER '@Id',
33:   Förnamn VARCHAR(15),
34:   Efternamn VARCHAR(15),
35:   Ålder INTEGER,
36:   AvdelningsId INTEGER './@Id'
37: )
38:
39: EXEC sp_xml_removedocument @resultDoc
```

Utskrift

	PersonId	Förnamn	Efternamn	Ålder	AvdelningsId
1	1	Anna	Andersson	26	3
2	2	Bertil	Johansson	48	3
3	3	Sivert	Persson	35	5

Som vi ser så är XML-koden inte riktigt densamma som innan. Vi har nu infört några attribut, och vi har också ett helt nytt element – Avdelning. Det går att likna det här vid två tabeller i en vanlig databas. En tabell som heter Avdelning, och en som heter Person. I Person finns det sedan en Främmande Nyckel från Avdelning, som talar om på vilken avdelning personen arbetar på. I XML ser detta ut som det ovan skrivet .

På rad 30, är det lite ändrat. Vi har såklart ändrat ”sökvägen”, då vi fortfarande vill utgå från Person. Vi har också skrivit en 3:a istället för en 2:a, detta för att vi är intresserade av både element och attribut.

Det är på rad 32 till 37 som det intressanta kommer. Vi börjar med rad 32. Precis som tidigare har vi angivit datatypen integer, men nu har vi skrivit ett annat namn (PersonId) för kolumnen. Eftersom vi gjorde detta, och detta namn inte finns i XML-dokumentet, behöver vi en tredje del som talar om vad vi vill ha i XML-dokumentet, nämligen attributet Id. @-tecknet talar om att det är ett attribut vi vill ha, hade vi utelämnat detta tecken hade den letat efter ett element som heter Id istället. Eftersom vi inte har detta hade det resulterat i att vi fått NULL istället för de Id-nummer vi fick. Tilläggs ska också att detta är Case-sensitive. Vi har ett attribut som heter Id, men vi har inget som heter ID, och kan därför inte skriva:

```
PersonId INTEGER '@ID'
```

Detta har att göra med att XML är känslig för stora och små bokstäver, och </anställda> kan inte avsluta start-taggen <Anställda>.

På rad 36 kommer något nytt igen. Här anger vi att den ska gå upp en nivå i hierarkin, och hämta attributet Id. Så uppenbarligen är vi inte knutna till en nivå i vårt XML-dokument, utan kan befinna oss var som helst i det och hämta data.

Lägga in i en tabell

Äntligen kommer vi till det mest intressanta, att lägga in data i en tabell. Vi kommer att använda oss av föregående exempel, och skapar därför en tabell som heter Person. Därefter lägger vi in data från XML-filen, till denna tabell. Detta blir SQL-frågorna:

```
1: CREATE TABLE Person
2: (
3:     PersonId INTEGER IDENTITY (1,1) NOT NULL,
4:     Fornamn VARCHAR(15),
5:     Efternamn VARCHAR(15),
6:     Alder INTEGER,
7:     AvdelningsID INTEGER
8: )
9: GO
10: DECLARE @resultDoc int, @xmlDoc varchar(4000)
11:
12: SET @xmlDoc = N'
13: <?xml version="1.0" encoding="ISO-8859-1"?>
14: <Anställda>
15:     <Avdelning Id="3">
16:         <Person Id="1">
17:             <Förnamn>Anna</Förnamn>
18:             <Efternamn>Andersson</Efternamn>
19:             <Ålder>26</Ålder>
20:         </Person>
21:         <Person Id="2">
22:             <Förnamn>Bertil</Förnamn>
23:             <Efternamn>Johansson</Efternamn>
24:             <Ålder>48</Ålder>
25:         </Person>
26:     </Avdelning>
27:     <Avdelning Id="5">
28:         <Person Id="3">
29:             <Förnamn>Sivert</Förnamn>
30:             <Efternamn>Persson</Efternamn>
31:             <Ålder>35</Ålder>
32:         </Person>
33:     </Avdelning>
34: </Anställda>'
35:
36: EXEC sp_xml_preparedocument @resultDoc OUTPUT, @xmlDoc
37:
38: SET IDENTITY_INSERT Person ON
39: INSERT INTO Person (PersonId, Efternamn, Fornamn, Alder, AvdelningsID)
40: SELECT PersonId, Efternamn, Fornamn, Alder, AvdelningsID
41: FROM OpenXML(@resultDoc, N'Anställda/Avdelning/Person', 3)
42: WITH (
43:     PersonId INTEGER '@Id',
44:     Fornamn VARCHAR(15),
45:     Efternamn VARCHAR(15),
46:     Alder INTEGER,
```

```
47:   AvdelningsID INTEGER './@ld'  
48: )  
49: SET IDENTITY_INSERT Person OFF  
50:  
51: EXEC sp_xml_removedocument @resultDoc
```

På rad 1 till 8 skapar vi en ny tabell som heter Person. Därefter på rad 10 till 36 är det exakt likadant som tidigare. Det är inte förrän på rad 38 som det blir något nytt. Eftersom vi har en Primärnyckel som ökas med ett (1) automatiskt, så kan vi inte sätta in vilket värde som helst i den kolumnen när vi gör en INSERT. Utan för att vi ska få sätta in de värden vi har fått från XML-dokumentet, måste vi sätta IDENTITY_INSERT till ON, vilket vi gör här.

På rad 39 börjar vår INSERT-fråga. Den är egentligen inte så konstig, om vi tänker på en vanlig INSERT-fråga med en SELECT-del. För det är precis det vi har här, en INSERT-fråga, som hämtar värdena från en annan tabell genom att använda SELECT.

På rad 40 väljer vi ut vad vi vill lägga in i tabellen Person. Precis som vid en vanlig INSERT så spelare det ingen roll vad kolumnerna heter, utan det är ordningen på dem.

Rad 41 till 49 är lika som förut, och på rad 51 gör vi motsatsen till vad vi gjorde på rad 36.

Om vi nu gör denna enkla fråga:

```
SELECT * FROM Person
```

Får vi detta resultat:

	PersonId	Fornamn	Efternamn	Alder	AvdelningsID
1	1	Anna	Andersson	26	3
2	2	Bertil	Johansson	48	3
3	3	Sivert	Persson	35	5

Vilket är exakt samma tabell som när vi gjorde en SELECT-fråga direkt mot XML-filen. Kort sagt, vi har lyckats få in vår data i tabellen Person!