Kapitel 33

Der xml-Datentyp

In diesem Kapitel:

Eine der wichtigsten Neuigkeiten von SQL Server 2005 ist die Einführung eines eigenen Datentyps für XML. Damit zollt Microsoft der immer weiter reichenden Verbreitung von XML-Dokumenten in der Informationstechnologie Rechnung. Bisher hatte man nur zwei Möglichkeiten, mit XML-Dokumenten in SQL Server umzugehen: Zum einen nimmt man jedes einzelne XML-Element und versucht, es in einer relationalen Tabelle abzulegen. Für manche Informationen mag das gehen, aber je stärker die Dokumente strukturiert sind, desto schwieriger wird eine vernünftige Umsetzung. Nimmt man das Beispiel in Listing 33.1, so bereitet schon die relationale Darstellung des *Lebenslauf*-Knotens Kopfzerbrechen.

```
<Lebenslauf>
  <Name>Walter Müller</Name>
  <Geburtstag>12.6.43</Geburtstag>
  <Geburtsort>Düsseldorf</Geburtsort>
  <Eltern>
    <Vater>
      <Name>Egon Müller</Name>
      <Geburtsort>Bern</Geburtsort>
    </Vater>
    <Mutter>
      <Name>Marga Müller</Name>
      <Mädchenname>Meier</Mädchenname>
      <Geburtsort>München</Geburtsort>
    </Mutter>
  </Eltern>
</Lebenslauf>
```

Listing 33.1 Einfaches strukturiertes XML-Fragment

Name, Geburtstag und -ort lassen sich ja noch einfach in der *Employees*-Tabelle als eigene Spalten abbil-
den – aber für die Eltern wäre dies sicher nicht so sinnvoll, da diese selbst wieder einen Geburtsort und eventuell weitere Informationen besitzen. Man würde in diesem Fall eine weitere Tabelle für die Eltern anlegen. Als Alternative bietet sich hier das vollständige Speichern des XML-Dokuments in der Tabelle an. Bisher musste man dazu eine *varchar()*- oder *text*-Spalte benutzen. Das Ein- und Auslesen der Daten mit anschließendem Weiterverarbeiten der enthaltenen Informationen war dann aber doch auch sehr aufwändig, da man immer das XML-Dokument in eine Zeichenkette umwandelt, und diese dann später wieder in XML transformieren muss. Zum Glück ist es mit SQL Server 2005 nun möglich, komplette XML-Dokumente samt Struktur in einer Spalte des *xml*-Datentyps abzulegen.

Der xml-Datentyp

Der *xml*-Datentyp kann zum Speichern von kompletten XML-Dokumenten oder Fragmenten verwendet werden. Einzige Einschränkung dabei: Die maximale Größe eines Dokuments darf 2 GByte nicht überschreiten. Intern speichert SQL Server 2005 den *xml*-Datentyp nämlich als BLOB, also ein Binary Large Object, das wiederum der 2 GByte-Beschränkung unterliegt.

Die Verwendung von *xml*-Datentypen innerhalb von Tabellen orientiert sich an dem üblichen Vorgehen anderer Datentypen in SQL Server. So benutzt Listing 33.2 zum Definieren einer *xml*-Spalte in der *Employees*-Tabelle folgende Anweisungen:

```
ALTER TABLE HRDepartment.Employees
  ADD CV xml NULL
```

Listing 33.2 Tabelle mit xml-Spalte ergänzen

Einschränkungen und Standardwerte

Wie bei anderen Datentypen üblich, kann man auch bei *xml*-Spalten einen Standardwert hinterlegen, mit dem die Spalte befüllt wird, wenn beim Einfügen einer neuen Zeile für die Spalte kein Wert explizit übergeben wird. Auf diese Weise kann man ein Grundgerüst für die weitere Verarbeitung der XML-Inhalte bereitstellen. Beim Neuanlegen eines Datensatzes könnte man also schon das Grundelement für das künftige Lebenslauf-Dokument einfügen, sodass man später nur noch einzelne Elemente in das Dokument einfügen braucht. Ein möglicher Ansatz dafür ist in Listing 33.3 zu finden.

```
ALTER TABLE HRDepartment.Employees
  ADD CONSTRAINT DF_Employees_CV DEFAULT N'<Lebenslauf></Lebenslauf>'
  FOR CV
```

Listing 33.3 Standardwert in xml-Spalten

Darüber hinaus können Sie auch Einschränkungen an die *xml*-Spalte binden, um die zulässigen Werte für den Lebenslauf noch genauer festzulegen.

Sie können den *xml*-Datentyp auch in Sichten verwenden. Einzige Einschränkung hierbei: Es werden keine partitionierten Sichten auf *xml*-Spalten unterstützt!

Verwendung von Triggern und berechneten Spalten

Mithilfe von Triggern ist es ein Leichtes, auch automatisch relationale Spalten aus einem XML-Dokument zu extrahieren. Nehmen wir einmal an, im Lebenslauf-Dokument gibt es ein Element »Geburtstag«, das Sie auch in anderen Anwendungen verwenden möchten. Wenn Sie nun eine eigene *Birthday*-Spalte in die *Employee*-Tabelle einfügen, können Sie den dazugehörigen Wert gleich aus dem *Lebenslauf*-Dokument auslesen. Definieren Sie dazu einfach einen Trigger, der den Inhalt des Geburtstags-Elementes aus dem XML-Dokument ermittelt und füllen Sie mit diesem Wert die *Birthday*-Spalte in Ihrer Tabelle. Damit wird automatisch der Wert in dieser Spalte aktualisiert, wenn sich das XML-Dokument ändert. In Listing 33.4 finden Sie ein Beispiel für so einen Trigger.

```
ALTER TRIGGER HRDepartment.InsertBirthday
   ON  HRDepartment.Employees
   AFTER INSERT, UPDATE
AS
```

```
  SET NOCOUNT ON;
  DECLARE @dt smalldatetime, @ID int
  -- Nur Aufrufen wenn die CV-Spalte verändert wurde
  IF UPDATE(CV)
  BEGIN
    -- Der folgende XQuery-Ausdruck wird in Kapitel 32 genauer erläutert
    SELECT @dt = CAST(CV.query('//Geburtstag/text()') AS varchar(10))
      , @ID = ID
    FROM inserted
    -- Nun die Birthday-Spalte aktualisieren
    UPDATE HRDepartment.Employees
    SET HiredDate = @dt
    WHERE ID = @ID
END
GO
```

Listing 33.4 Trigger zum Aktualisieren redundanter Daten aus einer xml-Spalte

Sollten Sie statt redundanter Spalten lieber berechnete Spalten verwenden wollen, können Sie das mithilfe des in Listing 33.5 gezeigten Beispieles tun.

```
CREATE FUNCTION BirthdayFromCV(@var xml) returns smalldatetime
AS BEGIN
RETURN @var.value('(//Geburtstag/text())[1]' , 'varchar(10)')
END
Go
-- Employee-Tabelle mit berechneter Spalte erweitern
ALTER TABLE HRDepartment.Employees
ADD BirthdayCV as dbo.BirthdayFromCV(CV)
Go
```

Listing 33.5 Berechnete Spalte aus xml-Daten

Typisierte xml-Spalten

Um die Gültigkeit eines XML-Dokumentes sicherzustellen, können Schemata in so genannten Schemaauflistungen in der Datenbank hinterlegt werden. Jedes XML-Dokument, das in eine *xml*-Spalte mit hinterlegtem Schema eingefügt wird, wird automatisch gegen das Schema geprüft und ein fehlerhaftes Dokument wird abgewiesen. Diese Spalten nennt man typisierte *xml*-Spalten. Im Listing 33.6 wird das Schema des Lebenslauf-Dokumentes in einer Schemaauflistung in der Datenbank hinterlegt. Anschließend kann dann für jede beliebige weitere *xml*-Spalte das entsprechende Schema in dieser Schemaauflistung hinterlegt werden. Im Beispiel wird dazu eine typisierte CV-Spalte namens *TypCV* der *Employees*-Tabelle hinzugefügt.

```
CREATE XML SCHEMA COLLECTION CVSchemaCollection
AS
N'<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Lebenslauf">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element name="PersönlicheDaten">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Geburtstag">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="Ort" type="xs:string" use="required" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="Geburtsort" type="xs:string" />
              <xs:element name="Eltern">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Vater" type="xs:string" />
                    <xs:element name="Mutter" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Schulbildung">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="Schule">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Name" type="xs:string" />
                    <xs:element name="Ort" type="xs:string" />
                    <xs:element name="Eintrittsjahr" type="xs:unsignedShort" />
                    <xs:element name="Austrittsjahr" type="xs:unsignedShort" />
                    <xs:element minOccurs="0" name="Abschluss" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Ausbildung">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="Ausbildungsstätte">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Name" type="xs:string" />
                    <xs:element name="Ort" type="xs:string" />
                    <xs:element name="Bereich" type="xs:string" />
                    <xs:element name="Eintrittsjahr" type="xs:unsignedShort" />
                    <xs:element name="Austrittsjahr" type="xs:unsignedShort" />
                    <xs:element name="Abschluss" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
```

```
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="BeruflicherWerdegang">
        <xs:complexType>
          <xs:sequence>
            <xs:element maxOccurs="unbounded" name="Arbeitsstätte">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="Name" type="xs:string" />
                  <xs:element name="Ort" type="xs:string" />
                  <xs:element name="Position" type="xs:string" />
                  <xs:element name="Eintrittsjahr" type="xs:unsignedShort" />
                  <xs:element name="Austrittsjahr" type="xs:unsignedShort" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="Id" type="xs:unsignedInt" use="required" />
  </xs:complexType>
 </xs:element>
</xs:schema>'

ALTER TABLE HRDepartment.Employees
 ADD TypCV xml(CVSchemaCollection)
```

Listing 33.6 Erzeugen einer SchemaCollection und einer typisierten Spalte

Falls Ihre XML-Dokumente über mehrere Namensräume verfügen, so können Sie in der Schemaauflistung für jeden Namensraum auch ein Schema hinterlegen. Wenn Sie nun versuchen, ein XML-Dokument einzufügen, das nicht dem hinterlegten Schema entspricht, das zum Beispiel ein »Vatter«- anstatt eines »Vater«-Elementes besitzt, so wird der Vorgang mit einer Fehlermeldung wie in Listing 33.7 abgebrochen.

```
Meldung 6965, Ebene 16, Status 1, Zeile 1
XML-Überprüfung: Ungültiger Inhalt. Erwartete Elemente: :Vater, angegeben wurde das Element 'Vatter'.
Ort: /*:Lebenslauf[1]/*:PersönlicheDaten[1]/*:Eltern[1]/*:Vatter[1].
```

Listing 33.7 Fehlermeldung bei ungültigem XML-Dokument

Umwandeln bestehender varchar()- oder text-Spalten in xml-Spalten

Vielleicht verfügen Ihre Datenbanken ja bereits über Tabellen, die XML-Daten in *text*- oder *varchar*-Feldern speichern. Mithilfe der *ALTER TABLE*-Anweisung, wie in Listing 33.8 zu sehen, können Sie diese Spalten nachträglich in eine *xml*-Spalte mit all den neuen nützlichen Eigenschaften umwandeln. Dabei wird gleich geprüft, ob es sich bei den bereits gespeicherten Inhalten um wohlgeformte XML-

Dokumente handelt. Falls Sie ein Schema mit der Tabelle verbunden haben, können Sie auch gleichzeitig die in den *xml*-Spalten abgelegten XML-Dokumente gegen dieses Schema validieren lassen.

```
ALTER TABLE Employees
ALTER COLUMN OldXmlDocuments xml
```

Listing 33.8 Ändern des Datentyps einer varchar()-Spalte in eine xml-Spalte

Abfragen aus xml-Datentypen

Der *xml*-Datentyp ist nicht nur zum Speichern von XML-Daten bzw. -Dokumenten ausgelegt, er bietet darüber hinaus auch verschiedene Methoden an, die für die Selektierung von Daten in XML-Inhalten nützlich sind. Gerade diese funktionalen Möglichkeiten des Datentyps sind eine große Bereicherung für das Arbeiten mit XML in SQL Server 2005. Anstatt aufwändig die Daten zunächst aus der Tabelle auszulesen, zu verarbeiten und wieder in die Tabelle zurückzuschreiben, werden die Methoden direkt in der Spalte verarbeitet. In Tabelle 33.1 sind die möglichen Methoden aufgeführt, die mit einer *xml*-Spalte verbunden sind.

Methode	Beschreibung
query()	Einfache Abfrage.
value()	Wert abrufen.
exist()	Prüfen auf NULL-Werte.
modify()	Updates mithilfe von XML-DML.
nodes()	Knoten auslesen.

Tabelle 33.1 Methoden des xml-Datentyps

Es ist also möglich, ohne weitere Hilfsmittel direkt Abfragen auf die XML-Dokumente in einer *xml*-Spalte auszuführen, bestimmte Werte oder XML-Fragmente zu extrahieren und auf die Existenz bestimmter Elemente oder Attribute zu prüfen. Darüber hinaus ist es sogar möglich, das XML-Dokument direkt in der Spalte zu verändern, ohne dass die Daten komplett ausgelesen und wieder zurückgeschrieben werden müssten.

Für die folgenden Beispiele werden Grundkenntnisse in XML vorausgesetzt, damit der Schwerpunkt des Kapitels auf die erweiterten Möglichkeiten von SQLXML 4.0 gelegt werden kann. Wenn Sie also vermehrt mit XML-Dokumenten zu tun haben und einen dafür strukturierten Speicherort suchen, der schon die wichtigsten XML-Parser-Eigenschaften mitbringt, lohnt sich der Blick auf die Fähigkeiten des neuen SQL Server 2005.

Grundsätzliches zur Verwendung von Namensräumen

Wie in Kapitel 30 beschrieben, ist es bei typisierten *xml*-Spalten, also Spalten, die ein hinterlegtes Schema zur Validierung der XML-Dokumente verwenden, durchaus üblich, Elemente aus unterschiedlichen Bereichen mit verschiedenen Namensräumen zu unterscheiden. Beim Zugriff auf diese Dokumente ist es deshalb unbedingt nötig, diese Namensräume »mitzuführen«. Das mag auf den ersten Blick etwas umständlich und aufwändig erscheinen, und es bleibt auch auf dem zweiten Blick so. Aber es hat durchaus handfeste Vorteile, sich mit diesem Thema etwas ausführlicher auseinanderzusetzen. Sehen Sie sich als Beispiel doch einmal das *Lebenslauf*-Dokument mit mehreren Namensräumen in Listing 33.9 an.

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<Lebenslauf xmlns:pe="http://www.NetShop.com/person"
  xmlns:sc="http://www.NetShop.com/school"
```

```
  xmlns:ed="http://www.NetShop.com/education"
  xmlns:wo="http://www.NetShop.com/work"
  Id="12345678">
  <pe:PersönlicheDaten>
    <pe:Geburtstag Ort="Hannover">12.4.1966</pe:Geburtstag>
    <pe:Geburtsort>Hannover</pe:Geburtsort>
    <pe:Eltern>
      <pe:Vater>Karl Muster</pe:Vater>
      <pe:Mutter>Luise Muster</pe:Mutter>
    </pe:Eltern>
  </pe:PersönlicheDaten>
  <sc:Schulbildung>
    <sc:Schule>
      <sc:Name>Paul-Ernst-Grundschule</sc:Name>
      <sc:Ort>Berlin</sc:Ort>
      <sc:Eintrittsjahr>1972</sc:Eintrittsjahr>
      <sc:Austrittsjahr>1978</sc:Austrittsjahr>
    </sc:Schule>
  </sc:Schulbildung>
  <ed:Ausbildung>
    <ed:Ausbildungsstätte>
      <ed:Name>TU Berlin</ed:Name>
      <ed:Ort>Berlin</ed:Ort>
      <ed:Bereich>Informatik</ed:Bereich>
      <ed:Eintrittsjahr>1985</ed:Eintrittsjahr>
      <ed:Austrittsjahr>1997</ed:Austrittsjahr>
      <ed:Abschluss>Dipl.-Inf.</ed:Abschluss>
    </ed:Ausbildungsstätte>
  </ed:Ausbildung>
  <wo:BeruflicherWerdegang>
    <wo:Arbeitsstätte>
      <wo:Name>Nordwind GmbH</wo:Name>
      <wo:Ort>Berlin</wo:Ort>
      <wo:Position>Systemadministrator</wo:Position>
      <wo:Eintrittsjahr>1997</wo:Eintrittsjahr>
      <wo:Austrittsjahr>2001</wo:Austrittsjahr>
    </wo:Arbeitsstätte>
  </wo:BeruflicherWerdegang>
</Lebenslauf>
```

Listing 33.9 »Lebenslauf«-Dokument mit Namensräumen

Sie fragen sich sicher, wozu dieser Aufwand betrieben wird. Wenn wir jedoch auf die Namensräume verzichten, so tritt in unserem Dokument das *Name*-Element in mehrfacher Bedeutung auf. Einmal als Name der Schule, als Name der Ausbildungsstelle oder als Name der Arbeitsstätte. Beim Austausch von Informationen ist dies aber ein klares K.o.-Kriterium! Wir brauchen doch gerade hier eine eindeutige Identifikation der Elemente, um sie später richtig zuordnen zu können. Natürlich kann man aufgrund der Position des Elementes Rückschlüsse auf den Inhalt ziehen, aber das ist auf Dauer relativ schwierig umzusetzen. Besser ist hier die Verwendung der Namensräume. So wird für jeden »logischen« Bereich des Dokuments ein möglichst klar bezeichneter Namensraum definiert, also für die persönlichen Daten, die schulischen, usw. Das Ergebnis sehen Sie in Listing 33.9. Wenn wir nun für dieses Dokument ein Schema erstellen, so sieht dieses recht umfangreich aus. Für jeden Namensraum wird dabei