

SQL Server 2008: Die neuen Datentypen

Datenverwaltung optimiert

Datenbanken stellen zur Datenverwaltung spezielle Datentypen zur Verfügung. Neue Datentypen erleichtern die optimierte Verwaltung von geografischen, hierarchischen oder auch binären Daten. **Jens K. Süßmeyer**

Auf einen Blick

Der SQL Server 2008 stellt neue Datentypen bereit und ermöglicht über diese eine verbesserte und flexiblere Datenverwaltung. Die neuen Datentypen erleichtern den Umgang mit:

- Datums-Zeitwerten
- binären Dateiinhalten
- hierarchischen Datenstrukturen
- spatialen Daten

■ **Plattform**

SQL Server 2008

■ **Technik/Anwendung**

T-SQL, Datentypen

■ **Voraussetzungen**

SQL Server 2008

Eine Fülle an neuen Datentypen erweitert das Portfolio des SQL Server 2008. Die Erweiterungen sprechen sowohl Entwickler als auch Administratoren an und eröffnen neue Einsatzgebiete für den SQL Server. Die neuen Datentypen dienen der Performance-Verbesserung bei der Laufzeit von Applikationen und vereinfachen die Wartung von Datenbanken.

Nachdem der SQL Server 2005 mit *VARCHAR(MAX)*, *NVARCHAR(MAX)*, *VARBINARY(MAX)* und *XML* nur wenige neue Datentypen zur Verfügung stellte, überrascht SQL Server 2008 mit einer Fülle an neuen und erweiterten Datentypen, die sowohl Entwickler als Architekten schon seit Längerem für ihre tägliche Arbeit forderten. Unter anderem macht dies die Integration der Common Language Runtime (CLR) des .NET Frameworks möglich, die nicht mehr explizit für eine Erweiterung der Funktionalitäten aktiviert werden muss, sondern bereits vom System aus bei Bedarf geladen wird. Hierbei wird aber dennoch strikt danach unterschieden, ob selbstentwickelte CLR-Routinen im SQL Server 2008 verwendet werden dürfen oder ob interne Systemfunktionen aufgerufen werden. Anders als Systemfunktionen, die ohne weitere Konfigurationen des Administrators das .NET Framework verwenden können, muss für selbstgeschriebene Assemblies im SQL Server die CLR-Funktion manuell und explizit über einen Systembefehl aktiviert werden.

Have a nice date

Microsoft erweiterte seine Typenbibliothek mit den folgenden Datentypen:

- Der Datentyp *DATE* speichert nur den Datumswert mit einer Spanne vom 01.01.0001 bis

zum 31.12.9999 mit einer Genauigkeit von einem Tag in 3 Byte.

- Der Datentyp *TIME(n)* speichert nur die Zeit in einer variablen Genauigkeit von bis zu 100 Nanosekunden. Dem Entwickler oder Architekten ist es dabei durch Spezifikation der Nachkommastellen in Klammern vorbehalten, wie genau ein Wert abgelegt werden soll. Je nach Genauigkeit ergibt sich ein verwendeter Speicherplatz von 3 Byte (0 bis 2 Nachkommastellen), 4 Byte (3 bis 4 Nachkommastellen) oder 5 Byte (5 bis 7 Nachkommastellen). Wird keine Genauigkeit in der Angabe des Datentyps festgelegt, wird automatisch die höchste Genauigkeit von 5 Byte beziehungsweise 7 Nachkommastellen angenommen.
- Der Datentyp *DATETIMEOFFSET(N)* vereint die Genauigkeit der Datentypen *DATE* und *TIME*. Über die Angabe der Nachkommastellen in Klammern kann auch hier die Genauigkeit je nach Bedarf gespeichert werden. Zudem ermöglicht dieser Datentyp eine Speicherung des Offsets nach ISO-8601-Standard durch Angabe der Zeitzone in den Spannen -14:00 bis +14:00. Verwechslungen, die sich bei geografisch verteilten Anwendungen ergaben, gehören mit diesem Datentyp nunmehr der Vergangenheit an.
- *DATETIME2(N)* kommt allen zugute, die auf die gemeinsame Speicherung von Datum und Uhrzeit nicht verzichten möchten, zugleich aber auch die Vorteile der neuen Genauigkeit für sich in Anspruch nehmen wollen. Der Datentyp speichert konfigurierbar das Datum und die Uhrzeit mit einer Genauigkeit von 0 bis 7 Nachkommastellen.

Es wurden aber nicht nur neue Datentypen zur Verwaltung von Datums- und Zeitwerten einge-

Tabelle 1: Übersicht über Datums- und Zeitdatentypen

Datentyp	Inhalt	Größe	Besonderheit	Beispiel
<i>Date</i>	Nur Datum	3 Byte	Spanne: 01.01.0001 bis 31.12.9999	05.09.1978
<i>Time(n)</i>	Nur Zeit	3 bis 5 Byte (je nach Genauigkeit)	Bis zu 100 Nanosekunden Genauigkeit	11:30:12.223652
<i>Datetimeoffset(n)</i>	Datum und Uhrzeit inklusive Zeitzone	8 bis 10 Byte (je nach Genauigkeit)	Bis zu 100 Nanosekunden Genauigkeit	05.09.1978 11:30:12.223652 01:00
<i>Datetime2(n)</i>	Kombination aus Date- und Time-Datentyp	6 bis 8 Byte (je nach Genauigkeit)	Spanne: 01.01.0001 bis 31.12.9999, bis zu 100 Nanosekunden Genauigkeit	05.09.1978 11:30:12.223652

führt, sondern auch bestehende Systemfunktionen überarbeitet, um mit diesen neuen Typen umzugehen. Exemplarisch sind hier die alten Bekannten wie *GETDATE*, *DATEADD*, *DATE-DIFF* oder *DATEPART* zu nennen. Aber auch neue Funktionen wie *SWITCHOFFSET* zum Ändern der Zeitzone oder *TODATETIMEOFFSET* zum Umwandeln eines Wertes in einen zeit-zonenaffinen Typ zählen dazu. Eine komplette Übersicht der Funktionen finden Sie in der Online-Hilfe von Microsoft [1] sowie in der aktuellen Version von Books Online (BOL) des SQL Server 2008 [2].

Anders als die bisherigen Funktionalitäten im SQL Server 2005 erlauben die Erweiterungen der Datums- und Zeittypen Entwicklern sowohl eine einfachere Implementierung als auch eine effizientere Speicherung von Datums- und Zeitwerten. Nunmehr können Sie die bisherige Speicherung von *DATETIME*-Datumswerten mit standardmäßig 8 Byte Speicherbedarf nach Bedarf variieren (Tabelle 1). Wer dennoch den *DATETIME*-Wert weiterhin verwenden möchte, um sich eine Migration zu ersparen, sei beruhigt. Der „alte“ Datentyp steht nicht auf der Liste der abgekündigten Features und wird sich somit aller Wahrscheinlichkeit nach auch noch in der nächsten Version von SQL Server wiederfinden.

Hierarchien werden erwachsen

Hierarchien im SQL Server 2005 und früheren Version abzulegen wurde von der Database-Engine so gut wie gar nicht unterstützt. Konstruk-

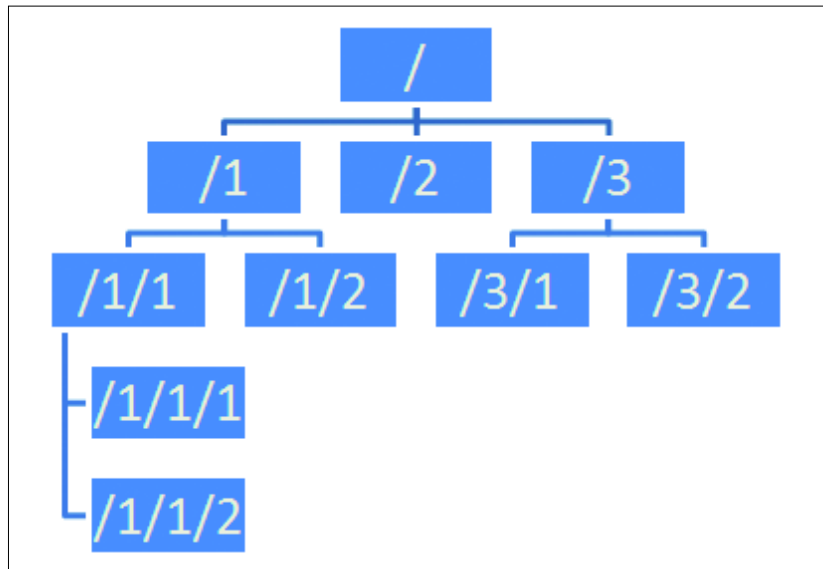


Abbildung einer einfachen Hierarchie und String-Repräsentierung (Bild 1)

te mit Parent/Child-Abbildungen in Spalten und die Speicherung von XML-Bäumen halfen den meisten Entwicklern und Administratoren als Zwischenlösung.

Die Abfrage dieser Konstrukte wurde in SQL Server 2005 durch die Einführung der Common Table Expressions und durch die Abfragesprache XQuery für XML-Datentypen ermöglicht. Eine durchgängige Implementierung folgt aber erst jetzt mit der Einführung des SQL Server 2008 und des nativen CLR-Datentyps *HierarchyId*, der genau für diese Zwecke vorgesehen und optimiert ist. Der Datentyp speichert die binäre Information über die Position des Datensatzes in der Hierarchie (Bild 1). ▶

Tabelle 2: Übersicht über mitgelieferte Systemfunktionen für Hierarchien

Funktion	Beschreibung	Beispiel
<i>GetRoot()</i>	Liefert den obersten Knoten einer Hierarchie ("")	SELECT OrgNode.ToString() AS Text_OrgNode FROM HumanResources.EmployeeDemo WHERE OrgNode = hierarchyid::GetRoot() ? liefert ""
<i>GetLevel()</i>	Liefert einen Integer des aktuell abgefragten Levels	SELECT OrgNode.ToString() AS Text_OrgNode FROM HumanResources.EmployeeDemo WHERE OrgNode = hierarchyid::GetRoot() ? liefert 0
<i>IsDescendant()</i>	Liefert einen booleschen Wert, ob die übergebene <i>HierarchyId</i> ein Kind des abgefragten Objekts ist	SELECT OrgNode.ToString() AS Text_OrgNode FROM HumanResources.EmployeeDemo WHERE OrgNode = (Select OrgNode WHERE OrgNode.ToString() = '/1')
<i>ToString()</i>	Liefert die String-Variante der <i>hierarchy</i> -Position	SELECT hierarchyid::GetRoot().ToString() FROM HumanResources.EmployeeDemo Liefert "/"
<i>Parse()</i>	Wandelt einen String in eine <i>HierarchyId</i> -Instanz um	SELECT hierarchyid::Parse('/') ? liefert das Top-Element
<i>Read() / Write()</i>	Wandelt von beziehungsweise in einen Binärstream	-
<i>GetDescendant()</i>	Gibt einen Kind-Datensatz wieder, der tiefer in der Hierarchie liegt	SELECT OrgNode. GetDescendant (NULL,NULL) FROM HumanResources.EmployeeDemo WHERE OrgNode = (Select OrgNode WHERE OrgNode.ToString() = '/1')
<i>GetAncestor(n)</i>	Gibt den n-ten Elternsatz in der Hierarchie wieder	Select OrgNode. GetAncestor(1) WHERE OrgNode.ToString() = '/1/'
<i>GetReparented Value()</i>	Bietet die Möglichkeit, eine Instanz einer <i>HierarchyId</i> in der Hierarchie an einen neuen Platz zu verschieben	Select OrgNode. GetReparentedValue(CAST('/1/1/1' AS HIERARCHYID), CAST('/4/' AS HIERARCHYID)) FROM HumanResources.EmployeeDemo

Neue Zeiten brechen an

Wer im SQL Server 2005 und früheren Versionen mit den Datentypen *DATETIME* und *SMALLDATETIME* gearbeitet hat, der weiß, welche Schwierigkeiten sich bei der Verwendung ergeben können. Dieser Datentyp umfasst stets das Datum und die Zeit und lässt sich nicht für die Aufnahme nur eines dieser Inhalte konfigurieren. Dies bringt den einen oder anderen Entwickler bei der Datenbankimplementierung in Bedrängnis. Optimierte Statements fallen durch eine Konvertierung oder Anwendung von Zeichenkettenoperationen auf den Datumswert zum Opfer.

Letztendlich wirkt sich dies nicht nur auf die Übersichtlichkeit des Codes, sondern auch auf die Performance der Abfragen aus. Des Weiteren ergaben sich Ungenauigkeiten bei der Verarbeitung der Datumswerte, da SQL Server für den Zeit-Teil nur die Ticks seit Mitternacht speichert. Dies ist eine Ungenauigkeit, die nur eine Präzision von 3.33 ms zulässt. Eine Speicherung der Uhrzeit 00:00:00 aus der Anwendung führt zu einer Speicherung von 00:00:00,003 und Abfragen auf die eingefügte Zeit laufen ins Leere (siehe [3]).

Zur vereinfachten Handhabung im TSQL- beziehungsweise Clientcode ist der neue Datentyp bereits von Haus aus mit vielen Systemfunktionen ausgestattet. Eine Übersicht dieser Systemfunktionen stellt **Tabelle 2** zusammen. Nicht nur das Abfragen der *HierarchyId*-Datensätze ist somit wesentlich einfacher geworden und bedarf keiner genauen Kenntnisse der aufgebauten Logik der Hierarchie, sondern auch das Einfügen (**Listing 1**) oder Verschieben eines Datensatzes ist durch die zur Verfügung gestellte Methode *GetReparentedValue()* leicht zu implementieren. Auch die Optimierung von Abfragen auf den Typ *HierarchyId* wird durch die Implementierung eines Indexes ermöglicht. Der Designer muss sich hierbei nur entscheiden, ob er einen „Breath-first“ oder einen „Depth-first“ verwenden möchte. Beim „Breath-First“ wird die Breite des Indexes besonders berücksichtigt, um etwa in einem Verzeichnisbaum alle beinhalteten Verzeichnisse eines bestimmten Verzeichnisses zu erhalten. Beim „Depth-first“ erhält die Tiefe des Indexes Vorrang, beispielsweise die Suche nach allen Dateien in einem bestimmten Ordner und dessen Unterordnern. Obwohl der neue Datentyp bereits eine vielfältige Manipulation und auch Abfragen ermöglicht, besteht durch die Implementierung als CLR-Datentyp auch die Möglichkeit, durch Ableitung die neue Datenstruktur und deren Funktionalitäten auch an eigene spezielle Bedürfnisse anzupassen.

FILESTREAM

FILESTREAM stellt formalerweise keinen eigenen Datentyp dar, da es nur ein Zusatz für eine

Spalte des Typs *VARBINARY(MAX)* ist. Dennoch lohnt es sich gerade in Hinsicht auf die immer wachsenden Anforderungen zur Speicherung von unstrukturierten Daten (zum Beispiel Dateien) diese neue Funktionalität in Betracht zu ziehen. Bisher existierten immer zwei Lager von Datenbank-Designern, die entweder Dateien in der Datenbank ablegten, oder andere, die nur einen Verweis auf eine im Dateisystem gespeicherte Datei verwendeten. Nun besteht die Möglichkeit, die Vorteile von beiden Methoden miteinander zu vereinen.

Bei der Anlage einer *FILESTREAM*-Spalte muss der Ersteller zuerst für die Datenbank einen *FileStream-Storage*-Pfad vorsehen. Dieser Pfad repräsentiert ein Verzeichnis auf einem lokalen Laufwerk beziehungsweise ein Verzeichnis in einer geclusterten Umgebung mit einem gemeinsamen Datenspeicher (Shared Storage), das selbst ein NTFS-Dateisystem verwendet. Dieses erstellte Verzeichnis wird bei erstmaliger Anlage vom SQL Server mit den entsprechenden Berechtigungen belegt, die das manuelle Ändern von Dateien innerhalb dieses Verzeichnisses verhindern beziehungsweise erschweren sollen.

Wird eine Datei über einen DML-Befehl (Data Manipulation Language) an den SQL Server gesendet, erstellt der SQL Server innerhalb eines transaktionalen Kontextes die Datei und sorgt dafür, dass bei einem eventuellen Rollback die Dateien auch wieder gelöscht werden. Die speziell gesicherten Dateien werden unter dem Sicherheitskontext des aktuellen Benutzers abgefragt, was bedeutet, dass wenn ein Benutzer keinen Zugriff auf eine vom FileStream verwendete Tabelle oder einen Datensatz hat, er automatisch auch keinen Zugriff auf die zugehörige Datei erhält. Durch die zur Verfügung gestellte Win32-Streaming-APIs kann die Datei innerhalb des vom SQL Server verwendeten TDS-Protokolls (tabellenförmiger Datenstrom) auch durch den Benutzer direkt über einen Stream empfangen und gesendet werden. Somit ist ein granulares Abfragen und optimiertes Streaming der Daten entgegen dem kompletten Abfragen einer Datei (etwa einer 2-GB-Datei) möglich.

Die Einarbeitung des *FILESTREAM*-Objekts als „First Class Citizen“, also als volle Funktionalität im SQL Server, erlaubt es, die transaktionalen Eigenschaften und auch Funktionen wie beispielsweise Backup-Mechanismen zu verwenden, um selbst die Daten der Datei in einem konsistenten Backup vorzuhalten. Damit ent-

Listing 1: Hinzufügen eines neuen Datensatzes am Ende der Hierarchie

```
INSERT HumanResources.EmployeeDemo (OrgNode, EmployeeID, LoginID, Title, HireDate)
SELECT Max(OrgNode), '0012254', 'MyDomain\TheNewEmployee', 'Praktikant', '01.10.2008' From HumanResources.EmployeeDemo
WHERE OrgNode = (Select OrgNode From HumanResources.EmployeeDemo WHERE LOGINId = 'MyDomain\TheManagerLoginName')
```

stehen keine fragwürdigen Lücken und Dateien mehr, die sich bei alten Methoden, wie beispielsweise dem separaten Speichern von Datei und UNC-Pfad (Universal Naming Convention) oftmals zeigten. Beschränkende Größe für *FILESTREAM*-Daten ist im SQL Server 2008 einzig und allein die Größe des verwendeten Speichermediums, obwohl der unter der Schale liegende und intern verwendete Datentyp *VARBINARY(MAX)* auf maximale Speichergößen von 2 GByte angewendet werden kann. Für *Filestream*-Daten gilt diese Beschränkung nicht.

Bisherige Anwendungen, die Daten bisher in BLOB-Spalten (Binary Large Objects) gespeichert und die Funktionalität des Volltextindizierungsdienstes mit *iFiltern* verwendet haben, können nach einer Migration auf den neuen Datentyp auch diese Funktionalität weiterhin nutzen. Auch der von Grund auf neu erstellte und integrierte Volltextindizierer (*iFTS* – Integrated Full Text Search), ermöglicht es, die *FILESTREAM*-Daten mit dem entsprechenden *iFilter* zu durchsuchen.

Räumliche Datentypen

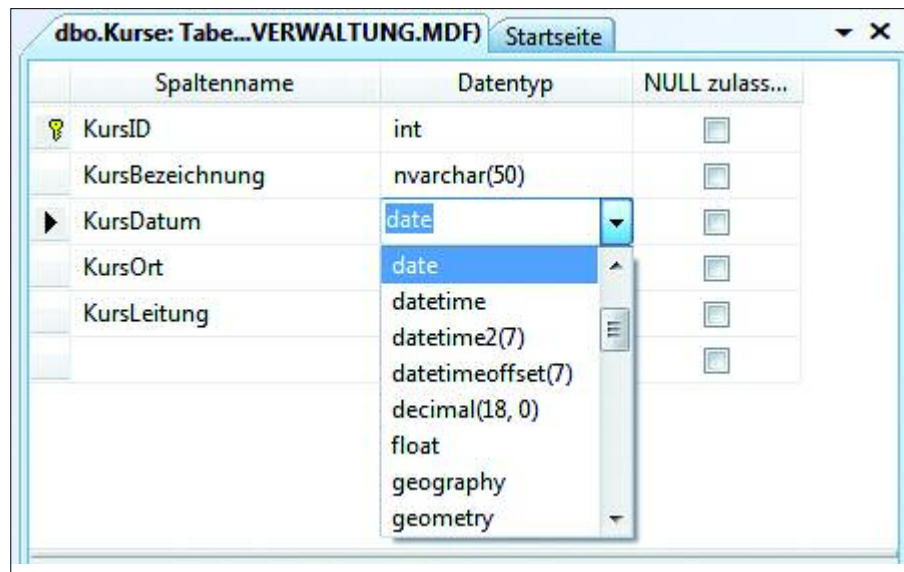
Geospatale Daten sind nur etwas für Spezialanwendungen? Weit gefehlt! Viele Endgeräte fangen mittlerweile an, geografische Daten zu sammeln. Ein gutes Beispiel ist die Seite von einem meiner Kollegen, der auf seiner Internetseite anhand eines kleinen GPS-Empfängers zyklisch Daten auf einem Webserver aktualisiert und diese mittels des Karten- und Informationsdienstes Virtual Earth grafisch darstellt (siehe [4]).

Ob Handys mit GPS Empfänger, Kameras oder auch viele andere Endgeräte, es wächst der Bedarf Informationen nicht nur zeitlich und inhaltlich festzuhalten, sondern diese auch mit geografischen Daten zu verknüpfen. Bisher existierten für diesen Fall hauptsächlich Spezialsysteme, die solche Informationen ablegen und auch verarbeiten konnten. Mit SQL Server 2008 werden zwei neue Datentypen eingeführt, die speziell für die Aufnahme dieser geografischen Informationen entwickelt wurden.

Geometry – 2D ist nicht mehr passé

Zurückversetzt in alte Schulzeiten kann sich wahrscheinlich noch jeder an die Grundelemente und Formen der Geometrie erinnern. Linien, Punkte und Polygone sind zusammengefasst die Welt des neuen *GEOMETRY*-Datentyps. Dieser dient zur Aufnahme von zweidimensionalen Elementen und hilft dabei, mit diversen bereits in der Datenbank-Engine implementierten Funktionen mit diesen Grundformen umzuge-

hen. Dabei stützt sich Microsoft auf die offiziellen Standards und veröffentlichten Methoden des OGC (Open Geospatial Consortium) (siehe [5]). Objekte können vereinfacht durch Aufrufe wie *STGeomFromText* generiert oder natürlich direkt aus der Datenbank geladen werden. Der nachfolgende Befehl zeigt die Generierung einer Linie von Punkt 1,1 zu Punkt 2,2 und die gleichzeitige Rückgabe der Länge des erzeugten LineStrings – in diesem Fall als einer Geraden. Zurückerrinnernd an den Satz des Pythagoras sollte das Ergebnis – hier wird der Wert als *Float*-Daten-



Neue Datentypen in der IDE von Visual Studio 2008 (Bild 2)

typ zurückgegeben – eine Länge von 1,41421 ergeben, also die Quadratwurzel von 2. Richtige Zahl, aber welche Einheit? SQL Server ist die Einheit in diesem Falle nicht wichtig, sie spielt nur eine untergeordnete Rolle und dementsprechend erfolgt die Berechnung einheitenlos.

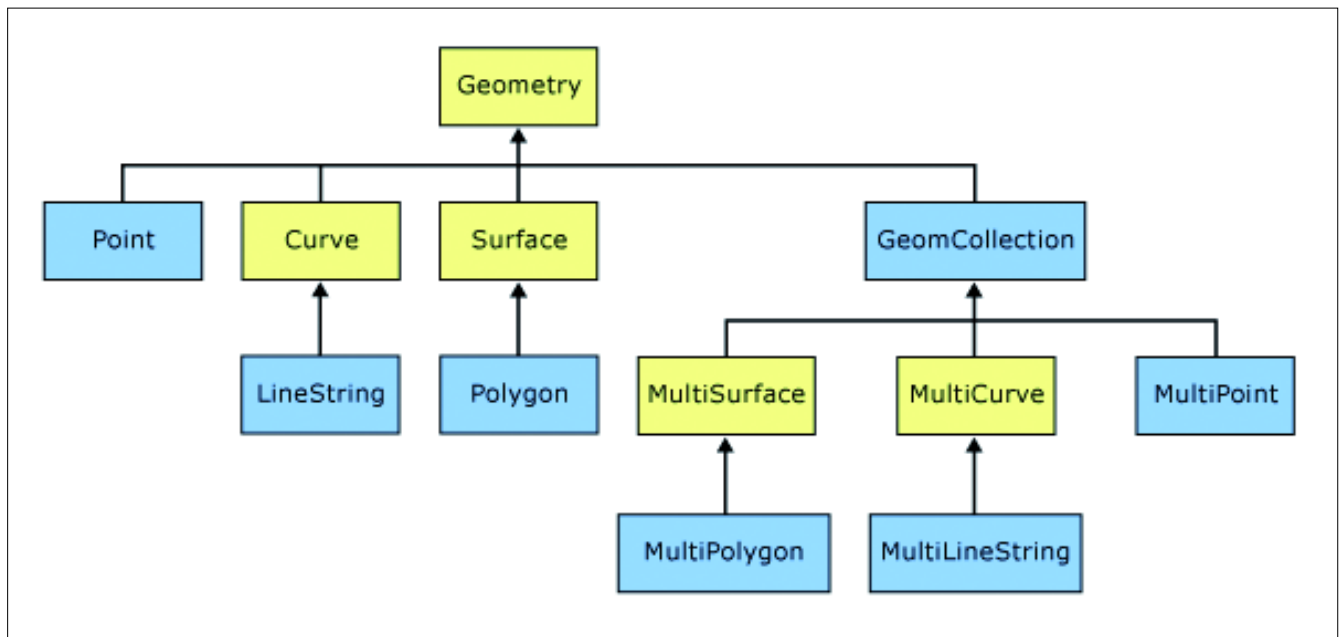
```
PRINT geometry::STGeomFromText(
'LINESTRING(1 1, 2 2)',0).STLength()
```

Geometry unterstützt aber nicht nur die genannten einfachen Formen, sondern erlaubt auch die Konstruktion komplexerer Formen im zweidimensionalen Raum. **Bild 2** zeigt die Datentyp-Anwahl in Visual Studio 2008. **Bild 3** zeigt die Hierarchie der Elemente, die sich durch *Geometry*- und *Geography*-Operationen erzeugen lassen.

Somit sind Flächenoperationen und Entfernungsmessungen sehr leicht möglich. **Listing 2** zeigt die Operationen von zwei Flächen gegeneinander, um deren Schnittflächen zu erhalten.

Geography – planar ist manchmal nicht genug

Wer schon einmal einen transatlantischen Flug nach Amerika unternommen hat und sich immer gewundert hat, warum der Pilot einen ►



Überblick über spatiale
Objekte in SQL Server
2008 (Bild 3)

halbkreisförmigen Umweg fliegt, wird die Erklärung im zweiten Geo-Datentyp von SQL Server 2008 wiederfinden, dem neuen *GEOGRAPHY*-Datentyp. Entgegen dem planaren System des *Geometry*-Datentyps honoriert dieser Typ auch die Erdkrümmung und somit das Globus-system mit seinen unterschiedlichen Referenzsystemen. Auch bei diesem Typ ist es möglich, mit standardisierten Methoden beziehungsweise Funktionen Berechnungen durchzuführen. Warum aber nutzt man zwei unterschiedliche Typen zur Berechnung?

Zugegebenermaßen bringt es wenig, beim Abstand eines Punktes zu einem Objekt auf einem Blatt Papier oder auch der Abmessung eines Raumes diese erst in ein Referenzsystem einzubinden und dann die Erdkrümmung einzubeziehen. Der tiefere Sinn eines Datentyps *Geography* liegt verallgemeinert in der größeren Erfassung von Objekten auf der Erde, der des Typs *Geometry* in der lokalen Abmessung beziehungsweise Darstellung von Objekten. Doch gibt es nicht nur ein Referenzsystem, auf das man sich mit einem geografischen System beziehen kann. In den Bei-

spielen des *Geometry*-Typs fiel in den Formeln der Parameter 0 auf, der kommentarlos übergangen wurde. Dieser Parameter dient zur Spezifikation der spatialen Referenzsysteme, in denen die Berechnungen durchgeführt werden sollen.

Die Referenzsysteme werden von der „European Petroleum Survey Group (EPSG) stan“ gepflegt. SQL Server 2008 unterstützt die im View *sys.spatial_reference_systems* verfügbaren Referenzsysteme. Obwohl eine Spalte des Datentyps *Geography* Objekte unterschiedlicher Referenzsysteme beinhalten kann, können Operationen nur zwischen Objekten gleicher Referenzsysteme ausgeführt werden.

Die Standardeinstellung, die der SQL Server 2008 verwendet, wenn kein anderes spezielles Referenzsystem spezifiziert wird, ist WGS84 (World Geodetic System) aus dem Jahre 1984, welches unter anderem auch von GPS-Systemen verwendet wird (siehe [6]).

Anwendungen in diesem Bereich können vielfältig sein. So sind Suchsysteme denkbar, die in einem bestimmten Umkreis Objekte bestimmen, Planungen einer Reisedstrecke für Außendienstmitarbeiter unterstützen oder genaue Auswertungen eines Verkaufsgebiets leisten. Geografische Informationen zu erhalten ist heute auch kein Problem mehr. Dienste wie Virtual Earth bieten die Möglichkeit, über diverse Schnittstellen Daten abzufragen und diese in eigenen Systemen zu verwenden.

Table Type

Im Grunde kein eigener Typ, aber dennoch wichtig für eine typensichere Implementierung ist der Table Type. Definiert als statischer Tabellentyp mit einer festen Struktur und vordefinierten Datentypen steht eine Instanz dieses Typs zur

Listing 2: Operationen zweier *Geometry*-Instanzen zueinander

```

--Die Fläche eines Quadrats (1)
DECLARE @Geom1 GEOMETRY
SET @Geom1 = geometry::STGeomFromText('POLYGON((0 0,0 1,1 1,0 0))',0)
PRINT @Geom1.STArea()

--Die Fläche eines Dreiecks (0.5)
DECLARE @Geom2 GEOMETRY
SET @Geom2 = geometry::STGeomFromText('POLYGON((0 0,1 1,0 0))',0)
PRINT @Geom2.STArea()

--Die überlappende Fläche des Dreiecks mit dem Quadrat
PRINT @Geom1.STIntersection(@Geom2).STArea()
  
```

Übergabe von Werten an eine Prozedur oder Funktion schreibgeschützt zur Verfügung. Was früher mit Tabellenvariablen nicht und mit temporären Tabellen nicht einfach möglich war, gelingt nun mit den Table Valued Parameters (Tabellenwertparametern) ohne Weiteres. So entfällt die oft leidvolle Übergabe von untypisierten kommaseparieren Werten oder XML-Strukturen, um strukturierte Daten weiterzugeben.

Die Daten werden unter Verwendung von Table Valued Parameters einfach optimiert durch eine Referenz auf den Speicherbereich durch nachfolgende Logiken abgefragt, anstatt die gesamten Speicherstrukturen durch den Speicher zu jagen. Die neuen Table Valued Parameters beziehungsweise Table Types vereinen zudem noch die Vorteile von sowohl temporären Tabellen mit ihrer Möglichkeit, Indizes und Constraints zu erstellen, als auch die schnelle und wenn gewollte flüchtige Initialisierung.

Nach der Definition eines solchen Typs werden die Strukturinformationen in `sys.table_types` für den Server als Referenz abgelegt. Initialisiert werden diese Typen durch eine einfache Deklaration des Typs und nachfolgende Befüllung durch herkömmliche `INSERT`-Anweisungen.

Ein perfektes Zusammenspiel ergibt sich zudem mit Client-Code, welcher es durch neue .NET-Datentypen ermöglicht, ganze Datenobjekte wie `DataTable`, `DbDataReader` oder Objekte, die `System.Collections.Generic.IList<SqlDataReader>` implementieren, zu übergeben. Somit ent-

fallen unnötige Roundtrips zum Server, da Programmlogik auf dem Server sofort die Objekte als Ganzes anstatt nur die einzelnen Zeilen akzeptieren kann (siehe den unten stehenden Kasten „Perfekt mit MERGE“).

Fazit

Dank der neuen Datentypen eröffnen sich nicht nur neue Geschäftsbereiche für Anwendungen, vielmehr lassen sich auch alte Vorgehensweisen und Implementierungen dadurch verbessern. Somit können Probleme und Einschränkungen, die Entwickler oftmals mit Speziallösungen umgingen, jetzt einfach durch die nativ implementierten Funktionen umgesetzt werden. Unterstützend hierbei sind nicht nur die bereits implementierten und mitgelieferten Systemfunktionen, sondern auch die Erweiterung der Client-Stacks wie des .NET Frameworks. [am]

[1] Online-Dokumentation zur Verwaltung von Zeit- und Datumswerten; <http://msdn.microsoft.com/de-de/library/ms180878.aspx>

[2] SQL Server 2008 Books Online; <http://msdn.microsoft.com/en-us/library/ms130214.aspx>

[3] Beispiele zur Verwaltung von Zeit- und Datumswerten; [http://msdn.microsoft.com/en-us/library/aa175784\(SQL.80\).aspx](http://msdn.microsoft.com/en-us/library/aa175784(SQL.80).aspx)

[4] Beispiel zur GPS-Personenlokalisierung; www.woistdaniel.de

[5] Homepage zum Open Geospatial Consortium (OGC); www.opengeospatial.org

[6] Informationen zum NGA: DoD World Geodetic System 1984; http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html

Perfekt mit Merge

`MERGE`, der neue DML-Befehl (Data Manipulation Language), der alle drei DML-Befehle, also `Insert`, `Update` und `Delete` in sich vereint, bietet mit den TVPs (Table Value Parameters) ein besonders gutes Zusammenspiel. Da TVPs auf dem Server wie normale Tabellen verwendet werden können und somit auch relationale Verknüpfungen (zum Beispiel in Joins) verwendet werden können, kann somit die Logik für einen Abgleich von Daten, die als TVP übergeben wurden, in nur einem Statement durchgeführt werden. Das erneute Prüfen auf bereits bestehende, fehlende oder geänderte Datensätze kann somit direkt in einer impliziten Transaktion durchgeführt werden. Bisher mussten Entwickler benutzerinitiierte, explizite Transaktionen für eine komplette Verarbeitung derselben Logik verwenden, um die Datenbank bei einem Fehler in einem konsistenten Zustand zu hinterlassen.

```
ALTER PROCEDURE [dbo].[SpSampleMerge]
(
    @TVPSampleMerger TVPSampleMerger READONLY
)
AS
BEGIN
MERGE dbo.MergeTest MergeTable
```

```
USING @TVPSampleMerger MergeSourceTable
ON MergeTable.SomeInt =
    MergeSourceTable.SomeInt
WHEN MATCHED AND
(
    MergeSourceTable.Deleted = 0x1
)
THEN DELETE
WHEN MATCHED AND
(
    MergeTable.SomeValue !=
    MergeSourceTable.SomeValue
)
THEN UPDATE SET
    MergeTable.SomeValue =
    MergeSourceTable.SomeValue
WHEN NOT MATCHED THEN INSERT
(
    SomeInt, SomeValue
)
VALUES
(
    SomeInt,
    SomeValue
);
END;
```