

Michael Rys über Native XML Support im SQL Server und XQuery und .NET 2.0

Michael Rys ist als Program Manager bei Microsoft für die XML-Technologien im SQL Server 2005 verantwortlich. Als Mitglied einer Arbeitsgruppe am W3C ist er außerdem an der Schaffung der XQuery-Spezifikation beteiligt.

dm: Der SQL Server 2005 wartet mit Native XML Support auf. Was genau versteht man darunter?

MR: Native XML Support bedeutet, dass man XML abspeichert und eine gewisse Originaltreue garantiert, also dass man die Ordnung und den *Mixed Content* erhält. Dies ist nicht der Fall, wenn ich die Daten nur in relationale Tabellen zerlege. Diese Originaltreue kann man natürlich auch erreichen, indem man die Daten als Textblock in der Datenbank speichert. Aber das ist nicht genug. Man muss auch die Funktionalität anbieten, um die einzelnen Informationseinheiten aus dem XML-Baum herauszubekommen, zum Beispiel mittels XPath oder XQuery. Diese Funktionalität wird vom SQL Server 2005 angeboten.

dm: In welchem Bereich sehen Sie einen Bedarf, XML in relationalen Datenbanken zu verwalten?

MR: Vom Datenbankbenutzer aus gesehen, ist das Hauptszenario, dass man relationale Daten hat, die man als XML in irgendeinen XML-Datenaustausch hineinbringen will, oder relationale Daten, die als XML reinkommen, wieder relational verwalten will. Dies wird im Jargon „XML-Publishing“ und „XML-Shredding“ genannt. Dies haben wir schon im SQL Server 2000 unterstützt. Und ich bin immer noch der Meinung, dass dies zumindest für die vorhersehbare Zeit der Hauptanwendungsbereich für XML und Datenbanken sein wird. Es hat sich jedoch gezeigt, dass immer mehr Leute auch *Mixed-Content*-Dokumente wie XHTML- oder Office-Dokumente sowie semistrukturierte Daten verwalten wollen. Deshalb haben wir den XML-Datentypen in SQL Server 2005 eingeführt.

dm: Native XML-Datenbanken bieten diese Funktionalität von Haus aus. Worin liegt der Vorteil für den Entwickler stattdessen den SQL Server 2005 einzusetzen?

MR: Native XML-Datenbanken sind meiner Meinung nach ein Nischenprodukt. Bei den nativen XML-Datenbanken gibt es erstmal verschiedene Kategorien. Einige native XML-Datenbanken sehen sich als eine Art Kompendium zu den relationalen Datenbanken. Für solche Anwendungen habe ich kein gutes Gefühl – weil genauso wenig, wie ich nur relationale Daten verwalte, es selten ist, dass ich nur XML-Daten verwalte. Ich verwalte viele Arten von Daten und ich möchte diese zusammen verwalten und nur ein Produkt kaufen und nicht zwanzig Produkte. Ich möchte unter Umständen diese Daten gemeinsam sichern und wiederherstellen. Ich möchte mich nicht darum kümmern, dass gewisse Daten im Dateisystem und andere in der Datenbank liegen. Es gibt auch native XML-Datenbanken, die eine Abstraktionsschicht über existierende XML-Dateien legen. Das kann durchaus seinen Reiz haben – vor allem, wenn man schon viele Dateien hat. Eine andere Anwendung wäre Mid-Tier-Cache – für diese Bereiche sehe ich durchaus Sinn, solche spezialisierten Datenbanken zu haben.

dm: XML wird also nur eines von vielen Formaten sein?

MR: Ja, das ist durchaus so. XML gleicht zum Beispiel von der Struktur her einem Baum. Es gibt aber verschiedene Datenmodelle, die sich eher als Graph strukturieren lassen, zum Beispiel Schüler und Lehrer. Jeder Lehrer hat mehrere Schüler und jeder Schüler hat mehre-

re Lehrer. Das kann man nicht so einfach in einem Baum darstellen. Und die XML-Funktionalität, um Referenzen zu modellieren, ist nicht so besonders geeignet. Für Nachrichten- und Dokumentenformate ist XML sicherlich ein wichtiges Datenformat.

dm: Welche Vorteile bringt XQuery gegenüber XPath und XSLT?

MR: XPath 2.0 ist zu 95 Prozent eine Untermenge von XQuery 1.0. Fast jeder XPath-2.0-Ausdruck ist auch ein XQuery-1.0-Ausdruck. XPath 2.0 wird auch im Kontext von XSLT 2.0 benutzt. Wo liegt also der Unterschied? XPath ist eine Navigations- und Knotenidentifizierungssprache. Mit XPath kann man in einem XML-Baum Knoten und Knotenmengen identifizieren und Funktionen darauf anwenden. Man kann damit jedoch keine neuen Bäume generieren und auch nicht Werte aggregieren, zum Beispiel die Summe bilden, oder nur Teile zurückgeben. Um eine Selektion oder eine Transformierung durchführen zu können, braucht man entweder XQuery oder XSLT. In dem Bereich haben XQuery und XSLT die gleiche Mächtigkeit – sie können beide ein XML-Dokument in ein anderes XML-Dokument umwandeln. Der Hauptunterschied ist: XSLT ist im Allgemeinen für datengesteuerte Transformationen gedacht, während XQuery eher für deklarative Abfragen gedacht ist. XSLT basiert auf Vorlagen [so genannte „Templates“]. Das ermöglicht eine modulare Entwicklungsweise und eine datengesteuerte Ausführung. Das Problem ist allerdings, dass es relativ schwierig ist, solche datengesteuerte XSLT Stylesheets zu optimieren: XSLT-Templates werden in komplexe *If-then-else*-Bäume umgesetzt. Die sind schwierig zu optimieren. XQuery ist wegen seiner Art einfacher zu optimieren. In XSLT kann man jedoch gewisse Sachen machen, die man mit XQuery wiederum nicht machen kann: Man kann es zum Beispiel benutzen, um Nicht-XML-Resultate zu generieren. Das ist mit XQuery nicht möglich, denn XQuery ist geschlossen über dem XML-Datenmodell.

dm: Wenn XQuery also durchaus eine gute Ergänzung zu XSLT ist, warum hat Microsoft dann XQuery aus dem .NET Framework 2.0 herausgenommen?

MR: Es hat ein paar Gründe gegeben – ich glaube, einer der wichtigsten ist, dass der Standard noch nicht fertig war. Wir haben damals im Jahre 1999 mit der MSXML-Bibliothek einen XSLT-Prozessor herausgebracht, der auf einem frühen Working Draft basierte. Dann gab es relativ große Breaking Changes zwischen dem Working Draft und der Final Recommendation. Das hat dazu geführt, dass wir die Leute unterstützen mussten, die die alte Version hatten, und wir mussten die Leute unterstützen, die die neue Version hatten und viele unserer Kunden wussten nicht, welche Version welche ist und welche sie denn benutzen müssen. Es war einfach ein bisschen zu viel Aufwand in diesem Zusammenhang. Die Begründung damals [für eine Vorabveröffentlichung] war, dass es noch nichts gab, was diese Funktionalität zur Verfügung stellte. Und ich glaube, dass es der Adoption von XSLT geholfen hat, dass der Internet Explorer zumindest diese Frühversion von XSLT unterstützt hat.

Mit XQuery haben wir das gleiche Problem und noch ein zusätzliches Problem: Erstens, XQuery hat keinen Namensraum. Ein XSLT-Dokument ist ein XML-Dokument und hat deshalb einen Namensraum für jedes Element. Und gibt es für jeden Working Draft einen unterschiedlichen Namensraum. Auf diese Weise kann man Versionierung betreiben und mehrere Versionen unterstützen. Jetzt hat aber XQuery keine einfache Möglichkeit, eine solche Versioning zu betreiben. Und auch das .NET Framework bietet keine Möglichkeit, verschiedene Versionen der gleichen Bibliothek zur Verfügung zu stellen.

Michael Rys über Native XML Support im SQL Server und XQuery und .NET 2.0

Der zweite Grund ist, dass für viele der Anwendungen auf dem Mid-Tier XSLT und XPath eigentlich reichen – so die Erfahrung unserer Kunden. Die Frage ist, wenn wir XQuery auf dem Mid-Tier implementieren, was tun die Benutzer damit? Was sind die Benutzervorteile zwischen XSLT 1.0 mit ein paar Erweiterungsbibliotheken und XQuery 1.0? Leute, die XQuery auf dem Mid-Tier benutzen möchten, können uns per E-Mail mitteilen, was ihre Beweggründe sind. Dann können wir das bei der Planung berücksichtigen.

dm: Waren die Diskussionen innerhalb der Working Group der Grund dafür, dass die Arbeit an XQuery solange gedauert hat?

MR: Das ist eine gute Frage. Zum Teil ja, allerdings dauert es eben im Allgemeinen länger einen Konsens zu finden, wenn viele Leute daran beteiligt sind, als wenn nur zwei oder drei Leute an einem Tisch sitzen. Es gibt auch andere Gründe, warum das so lange gedauert hat: Wenn man etwas auf der grünen Wiese plant, ist man norma-

lerweise schneller, als wenn man etwas in ein bestehendes Framework einarbeiten muss. Und wir haben ein bestehendes Framework: Wir haben XML Schema, XML InfoSet und XPath. Meine Erfahrung mit den Leuten in der Working Group war, dass alle immer sehr daran interessiert waren, vorwärts zu kommen. Aber es dauert halt – die Idee ist eben auch, dass die Sprache ein bisschen länger überlebt als die fünf Jahre, die es gedauert hat, sie zu entwickeln.

dm: Also eher "Was lange währt, wird endlich gut" und nicht etwa "Viele Köche verderben den Brei"?

MR: Die Nachwelt wird entscheiden, welches von Beiden eintrifft. [LACHT]

dm: Dann sprechen wir uns nächstes Jahr wieder auf der TechEd! Vielen Dank für das Gespräch.

Das Interview führte Martin Szugat