

Yukon is calling ...

SQL Server Query Notifications und ADO.NET 2.0 informieren über Datenänderungen

von Marcel Gnoth

Query Notifications sind ein neuer Mechanismus des SQL Server 2005. Der SQL Server überwacht Daten und informiert selbstständig Clients, wenn sich Daten, die zuvor abgefragt wurden, ändern. Dadurch wird es für Softwareentwickler leichter, in ihren Anwendungen immer aktuelle Daten anzuzeigen. Aber auch für den ASP.NET Cache findet dieser Mechanismus Anwendung.

Montagmorgen um halb neun in Deutschland. Herr Schulze startet wie jeden Morgen seine Auftragsverwaltung. Aus der Datenbank werden frische Daten geladen. Durch einen kurzen Blick auf den Bildschirm registriert er, dass keine dringende Eilbestellung eingetroffen ist. Also geht Herr Schulze erst einmal in die Büroküche, um sich mit Kaffee zu versorgen und die Kollegen zu begrüßen. Nach einer halben Stunde kehrt er zurück und beginnt seine Aufträge der Reihe nach abzuarbeiten. Doch was Herr Schulze nicht ahnen kann: In der Zwischenzeit traf die dringend erwartete Eilanfrage ein, nur sieht er davon nichts in seiner Anwendung. Die Daten auf seinem Monitor sind schlicht und ergreifend nicht mehr aktuell. Damit Ihnen in Zukunft nicht das Gleiche widerfährt, gibt es die Query Notifications.

Datenaktualisierung

An uns als Softwareentwickler wird immer wieder die Anforderung herangetragen, die Anzeige automatisch zu aktualisieren, sobald sich irgendetwas an den Daten geändert hat. Ein anderes Szenario ist

ein Daten-Cache in der Mittelschicht, der für Clients oder eine ASP-Anwendung einen aktuellen Datenstand zur Verfügung stellen muss. Auch dieser Cache muss bei Datenänderungen aktualisiert werden. Wie kann so etwas sichergestellt werden? Ein Weg ist das regelmäßige Abfragen (pollen) der Datenbank in einem bestimmten Intervall. Für den Cache ist das sicher eine Variante, aber wenn 1.000 Clients alle zehn Sekunden ihre Daten aktualisieren, ist das keine so gute Idee.

Eine andere Variante könnte über Trigger auf Tabellen realisiert werden. Solch ein Trigger könnte dann irgendeine Aktivität anstoßen: etwa eine COM-Komponente oder eine EXE starten oder eine Datei irgendwo erzeugen. Dabei sollte berücksichtigt werden, dass der Code des Trigger innerhalb der DML-Transaktion läuft und diese Transaktion verlangsamt oder im Fehlerfall sogar abbricht. Dafür können sicherlich Vorkehrungen getroffen werden.

Zum Beispiel könnte ein zentraler Dienst pollen oder von einem Trigger informiert werden. Aber damit hat dann der Entwickler wieder einen Haufen Arbeit, weil er sich um Infrastrukturcode kümmern muss, wo er doch eigentlich Geschäftslogik implementieren soll.

Query Notifications

Schön wäre es doch, wenn der Datenbankserver Bescheid sagen würde, falls Daten in der Zwischenzeit von anderen Benutzern verändert wurden. Und genau dafür gibt es *Query Notifications* in SQL Server 2005.

Wird eine *SELECT*-Abfrage zum Datenbankserver geschickt, kann dabei ein Notification Request beantragt werden. Für jeden *Request* wird eine *Subscription* in einer Tabelle der Datenbank angelegt. Der SQL Server überwacht dabei, ob Daten der Ergebnismenge von DML-Transaktionen betroffen sind. Für jede betroffene Subscription wird dann eine Nachricht generiert. Solche Subscriptions können nicht nur mit ADO.NET, sondern auch über OLE DB angelegt werden. Es ist auch möglich, HTTP und SOAP zu verwenden, aber dafür muss der SQL Server 2005 auf einem Windows Server 2003 oder Windows XP SP2 laufen.

ADO.NET 2.0 kommt ins Spiel

Mit ADO.NET 2.0 werden zwei neue Klassen eingeführt, *SQLDependency* und *SQLNotificationRequest*. Die erste bietet einen High-level-Zugriff auf *Query Notifications*, die zweite erlaubt einen flexibleren Zugriff. *SQLDependency* ver-



Quellcode auf CD

knüpft ein *SQLCommand*-Objekt mit einer *NotificationSubscription* im SQL Server und löst ein Ereignis aus, wenn der SQL Server eine Benachrichtigung generiert, weil sich die Daten geändert haben (Abbildung 1).

Dabei hat der Entwickler wenig Arbeit (Listing 1). Beim Erzeugen eines neuen *SQLDependency*-Objektes wird im Konstruktor ein *SQLCommand*-Objekt übergeben. Danach muss dem *SQLDependency*-Objekt ein *OnChange*-Ereignis nur noch eine Ereignisbehandlungsroutine zugewiesen werden, und schon ist alles fertig. Sobald das Kommando via *Execute* zum SQL Server gesendet wird, legt das *Dependency*-Objekt im Hintergrund einen *Notification Request* an. Sobald nun der SQL Server eine Änderung bemerkt, löst das *SQLDependency*-Objekt ein Event aus.

Listing 1

SQLDependency wird mit einem *SQLCommand* verknüpft

```
cnSQL.Open()
sqlcmdQuery.Connection = cnSQL
sqlcmdQuery.CommandText = txtSelCommand.Text
'set dependency and add eventhandler
m_SQLDep = New SqlDependency(sqlcmdQuery)
AddHandler m_SQLDep.OnChanged, New OnChanged
    EventHandler(AddressOf SqlDependency_OnChanged)
System.Diagnostics.Debug.WriteLine(sqlcmdQuery.
    Notification.Id)
Dim dr As SqlDataReader = sqlcmdQuery.ExecuteReader()
```

Listing 2

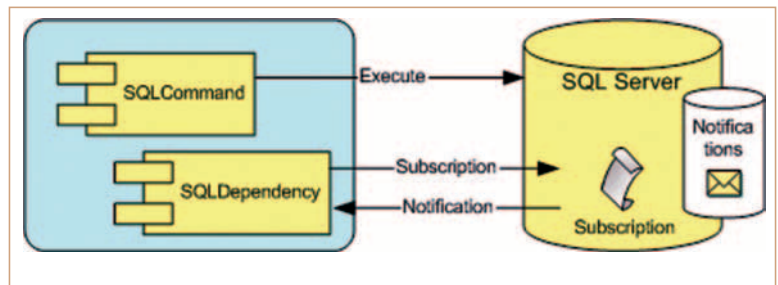
Auf das Ereignis des *SQLDependency*-Objektes reagieren

```
'Eventhandler des SQLDependency-Objektes
Private Sub SqlDependency_OnChanged(ByVal sender
    As Object, ByVal args As SqlNotificationEventArgs)
    Me.Invoke(New frmSQLDep.delgAddNotify(AddressOf
    Me.AddNotification), New Object() {args.Info.ToString,
    args.Type.ToString, args.Source.ToString})
End Sub

'This function is called via delegate for thread safety
Private Sub AddNotification(ByVal sInfo As String,
    ByVal sType As String, ByVal sSource As String)
    lstNotify.Items.Add("Info: " + sInfo + "; Type: " + sType
    + "; Source: " + sSource)

'Daten erneut abfragen
Call btnQuery_Click(Nothing, Nothing)
End Sub
```

Abb. 1: *SQLDependency* verknüpft eine Abfrage mit einem *Notification Request*.



SQLDependency meldet eine Änderung

In diesem Event kann durch die drei Eigenschaften *Info*, *Type* und *Source* der Grund der Nachricht ermittelt werden (Tabelle 1). Dabei gilt zu beachten: Der *CallBack* erfolgt auf einem separaten Thread. Deshalb kann in der Ereignisbehandlungsroutine nicht einfach die Oberfläche des Anwenders aktualisiert werden. Die beiden Threads müssen miteinander synchronisiert werden. Hierfür kann die *Invoke*-Methode einer Form eingesetzt werden. Dieser muss ein *Delegate* mit der eigentlichen Aktualisierungsroutine übergeben werden. Dieses *Delegate* wird aufgerufen, sobald die Synchronisation zwischen beiden Threads das zulässt (Listing 2).

Nach jeder Benachrichtigung muss die *Subscription* erneuert werden. Dies geschieht am besten, indem die aktuellen Daten auf Grund der Änderung abgefragt werden und dabei wieder ein *SQLDependency*-Objekt mit der Abfrage verknüpft wird.

Benachrichtigungen

Der SQL Server sendet nicht nur Nachrichten, wenn Daten geändert wurden (Tabelle 1). Nachrichten vom Typ *Change* können auch durch *Alter Table* oder *Drop Table* oder das Neustarten des SQL Server ausgelöst werden. Weitere Gründe können das Ablaufen der *Subscription* sein oder aber die *Subscription* konnte nicht angelegt werden, weil der Server überlastet ist oder andere Bedingungen nicht erfüllt sind.

Wenn Sie eine Abfrage mit *Notification Request* abschicken und sofort eine Antwort erhalten, obwohl keine Daten geändert wurden, dann gab es irgendwelche Probleme mit dem *Notification Request*. Diese Nachrichten sind vom Typ *Subscription*. Der häufigste Grund liegt darin, dass das *SQL-Statement* nicht den Bedingungen entsprach. Einige Hinweise zu Problemen mit *Query Notifications* finden Sie in

Kasten 1. Bei großer Last auf dem Server können auch einmal Nachrichten versandt werden, obwohl eine Abfrage gar nicht von einer Änderung betroffen war. Diese Abfrage basierte aber auf einer Tabelle, die geändert wurde. Unter großer Last wird dann nicht jede *Where*-Bedingung einzeln geprüft, sondern es werden nach dem Motto „lieber einer mehr als zu wenig“ gleich alle benachrichtigt.

Bedingungen

Da nun die *Query Notifications* auf dem *ChangeDetection*-Mechanismus für indizierte Views aufbauen, gelten auch die gleichen Bedingungen und Voraussetzungen. Vor allem muss das *SELECT* bestimmte Bedingungen erfüllen. Eine ausführliche Liste finden Sie in der *SQL-Server-Onlinehilfe*. Die wichtigsten Bedingungen sind zweiseitige Tabellennamen und kein *Select **. Also nicht:

```
SELECT * FROM Product WHERE ListPrice < 300
```

sondern:

```
SELECT ProductNumber, Name FROM Production.
    Product WHERE ListPrice < 300
```

Darüber hinaus sind auch *Distinct*, *Compute*, *Into* und die *BLOB*-Datentypen *text*, *ntext*, *image* nicht erlaubt.

Notifications vom Typ Change

Source = Date	Truncate, Update, Insert, Delete
Source = Timeout	Timeout
Source = Object	Drop, Alter
Source = System	Restart, Error, Ressource

Notifications vom Typ Subscriptions

Source = Statement	Query, Invalid
aSource = Environment	Options, Isolations

Tabelle 1: *Notifications*, die vom SQL Server versendet werden

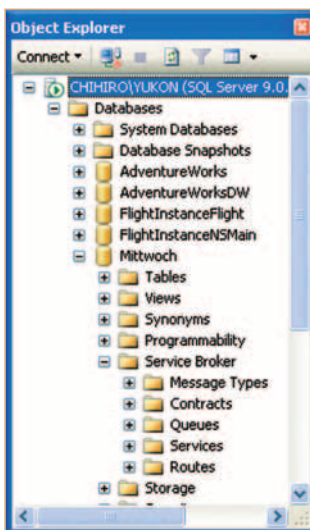


Abb. 2: Zu einer Datenbank gehören nicht nur Tabellen und Ansichten, sondern auch Service-Broker-Objekte

Server- und Netzwerklast erzeugt. Mühsen zu viele Anwender informiert werden, dann sollte über eine Mittelschicht nachgedacht werden, die die Informationen zum Beispiel via MSMQ an viele Anwender verteilt [1].

Es ist auch wichtig, welche Arten von *Select*-Anweisungen überwacht werden. Gleiche Abfragen, die sich nur in den Werten der Parameter unterscheiden, sind vorteilhaft. Dann kann der SQL Server optimieren. Werden hingegen viele sehr unterschiedliche Anweisungen verwendet, wird die Belastung für den Server größer.

Service Broker

Der Change-Detection-Mechanismus ist dafür zuständig, die Änderungen zu entdecken. Für das Zustellen der Nachrichten ist der Service Broker verantwortlich. Das ist ein neuer Dienst, der mit dem SQL Server 2005 eingeführt wird. Der Service Broker ermöglicht das asynchrone Versenden von Nachrichten in Warteschlangen, ähnlich Microsoft Message Queue. Nur dass hier die Funktionalität innerhalb der Datenbank verwendet wird. Die Warteschlangen sind Tabellen der Datenbank. Darüber hinaus können noch Nachrichten und Dialoge definiert werden und Routen zwischen SQL-Servern festgelegt werden (Abbildung 2).

SQLNotificationRequest

Der große Bruder des *SQLDependency*-Objektes ist das *SQLNotificationRequest*-Objekt. Es bietet dem Programmierer nicht

so viel Komfort wie sein „kleinerer Bruder“, dafür aber mehr Eingriffsmöglichkeiten in den Vorgang. Der Hauptunterschied liegt darin, dass die Service-Broker-

Wann ist der Einsatz sinnvoll?

Durch den Einsatz von Query Notifications bekommt der SQL Server natürlich mehr zu tun, jedes *Update* wird „teurer“. Deshalb lohnt der Einsatz vor allem dann, wenn viel gelesen, aber nur wenig geschrieben wird. Dadurch kann das regelmäßige pollen der Datenbank entfallen, was den Server wieder entlastet. Wenn die Anzahl der Anwender, die stets aktuelle Daten auf ihrem Bildschirm wollen, überschaubar bleibt, kann der Einsatz lohnen, auch wenn viel geschrieben wird. Viele schreiben, nur wenige brauchen Notifications. Mit Hilfe von Query Notifications kann eine Benachrichtigung der Anwender bei Änderungen leicht implementiert werden. Dabei sollte immer beachtet werden, dass jede Abfrage und jede Benachrichtigung

Troubleshooting

Viele Entwickler hatten Probleme, die Beispiele aus dem MSDN zum Laufen zu bringen. Hier eine kleine Checkliste für die Datenbank „Mittwoch“:

- Der Service-Broker muss für eine Datenbank aktiviert sein:

```
--Checks if Service Broker is enabled at the DB
select databasepropertyex('Mittwoch', 'IsBrokerEnabled')
--Set Service Broker enabled for the Database
ALTER DATABASE Mittwoch SET ENABLE_BROKER
```

- Wenn nicht mit dem Administrator-Account gearbeitet wird, dann werden einige Rechte benötigt:

```
GRANT send on service::MittwochNotifications to guest
GRANT Subscribe Query Notifications To guest
```

- Die *Select*-Anweisung muss zweiteilige Tabellenamen verwenden: *dbo.Kunde* und muss jede Spalte explizit angeben, also kein * verwenden.
- Mit *SQLNotificationRequest* kann erst einmal geprüft werden, ob eine Subscription angelegt werden konnte.
- Eine Firewall kann ein Problem sein. Auf einem lokalen Windows XP mit SP2 Firewall liefern die Beispiele mit Yukon und Whidbey.
- Die Log-Dateien des SQL Server sind ein guter Platz, um Hinweise auf Probleme zu finden.
- Bei Problemen ist Hilfe in der ADO.NET Beta Newsgroup von Microsoft zu finden: communities.microsoft.com/newsgroups/default.asp?icp=whidbey

Listing 2

Service-Broker-Objekte in der Datenbank „Mittwoch“ anlegen

```
USE Mittwoch ;

CREATE QUEUE MittwochMessages ;

CREATE SERVICE MittwochNotifications
ON QUEUE MittwochMessages
([http://schemas.microsoft.com/SQL/Notifications/PostQueryNotification]);

CREATE ROUTE
MittwochMessagesRoute
WITH SERVICE_NAME = 'MittwochNotifications',
ADDRESS = 'LOCAL' ;
```

Listing 3

Mit *SQLNotificationRequest* eine Subscription anlegen

```
cnMittwoch.Open()

' Define the service to receive the notifications.
Dim service As String = "MittwochNotifications"
Dim query As String = "Select ID, Name From dbo.
                        Person Where ID<1000"
Dim command As SqlCommand = New SqlCommand
                        (query, cnMittwoch)

' NotificationRequest Object
Dim nrRequest As SqlNotificationRequest
nrRequest = New SqlNotificationRequest
                        (Guid.NewGuid().ToString(), service, 1200)

' Assign the Request to the Command
command.Notification = nrRequest

' Exec Command, this generate
Dim reader As SqlDataReader = command.ExecuteReader()
```

Listing 4

Lesen der Nachrichten

```
--Receive the next Message from the Queue
WAITFOR(RECEIVE * FROM MittwochMessages)

--Select Content of the Queue and Convert the
                        MessageBody in Text
select cast(message_body as nvarchar(MAX)) As
                        TheMessage, * from MittwochMessages
```

Objekte selbst angelegt werden müssen (Listing 2) und auch das Abfragen von Notifications muss der Programmierer übernehmen. Das *SQLNotificationRequest*-Objekt übernimmt nur das Anlegen der Subscription. Der große Vorteil ist, dass das Clientprogramm zwischendurch geschlossen werden kann und die Benachrichtigungen nach einem Neustart abgeholt werden können.

Nach dem Anlegen der Service-Broker-Objekte in der Datenbank kann eine Subscription angefordert werden. Hierfür wird wieder ein *SQLCommand*-Objekt mit einem *SQLNotificationRequest*-Objekt verknüpft (Listing 3). Beim Ausführen des Kommandos mit *Execute* wird in der Datenbank die Subscription angelegt. Mit der Anweisung

```
Select * From sys.dm_qn_subscriptions
```

können alle angelegten Subscriptions in der Datenbank aufgelistet werden. Werden jetzt die Daten verändert, wird eine Notification generiert und die Subscription wird gelöscht. Die Notification bleibt so lange in der Datenbank-Queue, bis sie abgeholt wird.

Lesen der Nachrichten

Listing 4 zeigt, wie die Nachrichten gelesen werden können. Der Body der Nachricht ist vom Typ *Binary* und muss mit *Cast* in lesbare XML umgewandelt werden. *Select* listet alle Nachrichten der Queue auf, *Receive* entnimmt eine Nachricht. *WaitFor* blockiert so lange, bis eine Nachricht entnommen werden kann und gibt diese Nachricht dann aus. Das ist eine interessante Variante in Verbindung mit den neuen asynchronen Datenbankabfragen von ADO.NET 2.0. Denn auf einem separaten Thread könnte mit *WaitFor* ein *SQLCom-*

mand-Objekt asynchron ausgeführt werden, das dann über *EndExecuteReader* informiert, wenn eine Nachricht eingetroffen ist. So wird der Komfort des *SQLDependency*-Objektes erreicht und die Anwendung kann trotzdem offline gehen.

Schlussbemerkung

Das *SQLNotificationRequest*-Objekt bietet dem Entwickler mehr Eingriffsmöglichkeiten, um den Query-Notification-Mechanismus zu steuern, verlangt aber auch mehr Aufwand. Wenn *SQLDependency* ausreicht, so ist dies sicher die einfachere Variante; der Entwickler muss sich nicht mit der Service-Broker-Programmierung auseinandersetzen. Query Notifications

geben uns Entwicklern eine komfortable Infrastruktur, um auf Änderungen in der Datenbank sofort zu reagieren.

SQL Server 2005 befindet sich im Beta-2-Stadium und soll im Sommer 2005 offiziell ausgeliefert werden. Die Praxis wird zeigen müssen, wie eine Anwendung damit umzugehen hat, wenn der SQL Server das Anlegen von Subscriptions verweigert, etwa weil seine Last zu hoch ist. ●

Marcel Gnoth ist Entwicklungsleiter der Berliner NTeam GmbH. Außerdem führt er Trainings durch, schreibt für Fachmagazine und ist regelmäßig als Sprecher auf der BASTA! anzutreffen. Sie erreichen ihn unter www.gnoth.net oder marcel@gnoth.net.

Anzeige

Beispielanwendung

Auf der Begleit-CD befinden sich zwei Ordner. Einer enthält zwei VS2005-Projekte. In beiden Projekten muss jeweils im Formular der Connection String angepasst werden. In einem zweiten Ordner befindet sich eine SQL Server 2005 Solution, die zwei Skriptdateien enthält. Eine zum Anlegen der Datenbank (bitte hier den Dateipfad eventuell anpassen) und die andere enthält einige interessante SQL-Anweisungen, die auch im Artikel vorkommen.