

Kapitel 9

LINQ to SQL

In diesem Kapitel:

Einleitung	176
Voraussetzungen für die Beispiele dieses Kapitels	176
Wie es bisher war – Visual Studio 2005 vs. Visual Studio 2008	177
Die ersten Schritte	179
Kaskadierte Abfragen	184
Daten verändern und speichern	187
Transaktionen	191
Was, wenn LINQ einmal nicht reicht	192

Einleitung

Als ich meine ersten Gehversuche mit LINQ machte, war ich begeistert. Nur nach und nach stellte ich fest, welche enormen Möglichkeiten sich mir mit LINQ auf einmal boten – und ich hatte zu diesem Zeitpunkt doch noch gar nichts gesehen! Während meiner anfänglichen Gehversuche mit einer der frühen Betas von Visual Studio 2008 wusste ich nämlich zum damaligen Zeitpunkt, irgendwann im April 2007, noch gar nicht, wie sehr ich erst an der Spitze des Eisberges 'rumdokterte.

Doch nach und nach begriff ich, wohin »uns« diese ganze LINQ-Geschichte führen sollte, und irgendwann war es dann auch so weit, dass es die ersten Visual Studio Betas gab, die uns die ersten Versionen von *LINQ to SQL* vorführten. Wir waren alle total begeistert.

Das Prinzip ist eigentlich ganz einfach. Sie »sagen«: »Ich programmiere im Folgenden genau so, wie ich es mit LINQ to Objects gelernt habe, die Quelle meiner Daten ist jedoch keine Auflistung, sondern eine SQL Server-Datenbank. Ansonsten bleibt alles genau so«.

Und so einfach soll das sein? Keine neuen Objekte, Klassen und Verfahren, die man lernen muss? Nichts Spezielles, was man zu beachten hat? Gibt es keine Falltüren, auf die man aufpassen sollte?

Doch, na klar, ein paar gibt es, denn Sie wollen Daten ja nicht nur aus dem SQL Server »abholen«, Sie wollen sie ja schließlich nach ihrer Verarbeitung auch wieder zurück in den Server bekommen, und dazu muss die vorhandene Infrastruktur natürlich ein wenig »aufgebohrt« werden, damit sie das gestattet. Und dann muss es ja immerhin auch noch möglich sein, mit ein paar SQL Server-Schmankerln zu arbeiten, beispielsweise mit gespeicherten Prozeduren! Und zu guter Letzt soll dieses Kapitel auch nicht nur diese, sondern ein paar Seiten mehr haben, und das hat es schließlich auch...

Voraussetzungen für die Beispiele dieses Kapitels

Dieses Kapitel nennt sich LINQ to SQL.¹ Und mit SQL im Part dieses Namens ist nicht etwa irgendein SQL gemeint, sondern SQL in Form von Microsoft SQL Server. Mit der Visual Studio Version, mit der dieses Buch entstand, war es lediglich möglich, mit dem SQL Server der Versionen 2000 und 2005 von Microsoft zu arbeiten. Erst mit dem Erscheinen des SQL Server 2008 und dem Service Pack 1 für Visual Studio 2008 wird es möglich sein, auch LINQ to SQL mit dieser Serverversion zu betreiben. Für andere SQL-Server, wie Oracle oder MySQL gibt es zur Drucklegung noch keine entsprechenden Treiber.

Um die Beispiele dieses Buches nachvollziehen zu können, benötigen Sie also eine funktionsfähige SQL Server-Instanz – am besten in der Version 2005. SQL Server 2005 in der Express Edition ist dazu ausreichend, und Sie können sie, falls Sie sie nicht im Rahmen von Visual Studio 2008 ohnehin haben mitinstallieren lassen, auch ohne Probleme von den Microsoft-Seiten herunterladen.

¹ Übrigens: Tatsächlich sollte es »Link tuh Ess Kju Äll« ausgesprochen werden, aber genau wie bei Microsofts SQL Server, den (fast) jeder »*fiekwäl* Sörwä« ausspricht, nennen es die meisten »Link to *fiekwäl*«.

Darüber hinaus brauchen Sie für die Beispiele dieses Kapitels auch die Beispieldatenbank *AdventureWorks* die Sie unter <http://codeplex.com/SqlServerSamples> herunterladen und installieren können.

Wählen Sie das MSI-Installerpaket *AdventureWorksLT.msi* zum Download und zur Installation aus.²

Beta-Warnung

Dieses Buch entstand, wie bereits angedeutet, nicht mit der RTM-Version von Visual Studio 2008, sondern mit dem Release Candidate. Auch aus diesem Grund könnte es zu folgenden Abweichungen kommen:

Je nach eingesetzter Visual Studio Version werden ggf. durch den Klassendesigner unterschiedliche Klassennamen generiert. Beim Ausprobieren mit der RC-Version und der finalen Visual Basic 2008 Express Edition konnten wir folgende Unterschiede feststellen:

Version	Tabelle	Klasse, die einen Kunden abbildet	Table(of Customer) im DataContext
Visual Studio 2008 RC1	Customer	Customer	Customers
Visual Studio Express 2008	Customer	Customer	Customer

Tabelle 9.1: Unterschiedliche Namensgebung der verschiedenen Visual Studio-Versionen

Desweiteren verwendet Visual Basic 2008 Express einen anderen Datenbankauswahldialog, da Visual Basic 2008 in der Express Edition nur den Modus des Anhängens von Datenbankdateien in SQL Express unterstützt.

Wie es bisher war – Visual Studio 2005 vs. Visual Studio 2008

Auch in Visual Studio 2005 war es natürlich schon möglich, Client/Server-Anwendungen auf Basis von SQL Server 2000/2005 zu erstellen. Man bediente sich dabei regelmäßig eines interaktiven DataSet-Designers, der es ermöglichte, so genannte typisierte DataSets zu erstellen. Diese übernahmen die Infrastruktur für den Austausch von Daten zwischen Client und SQL-Server.

² Eine recht ausführliche Anleitung zur Installation von SQL Express finden Sie unter www.activedevelop.de in der SQL Server-Section.

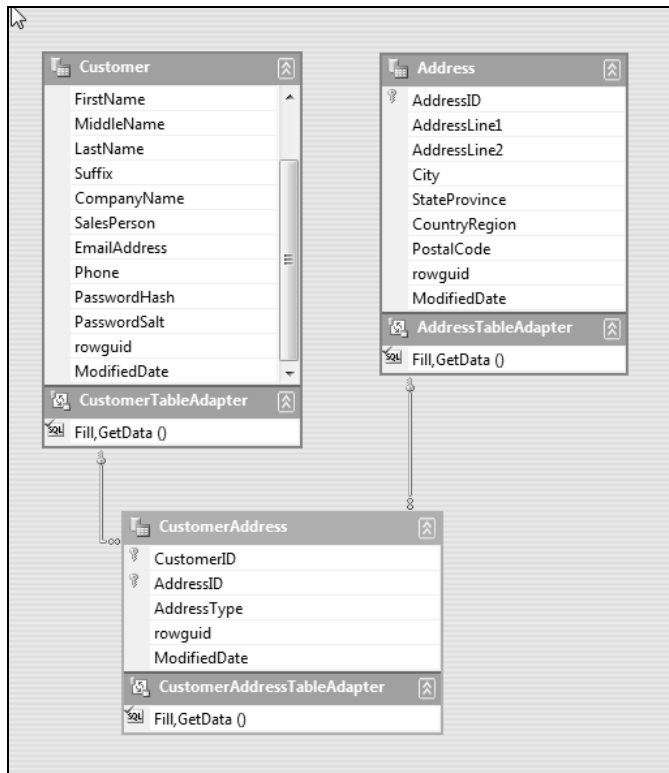


Abbildung 9.1 Der DataSet-Designer von VS 2005

Pro Tabelle wurde eine DataTable-Klasse erstellt und zudem noch jeweils eine Klasse mit dem TableAdapter. Letzterer enthielt die Methoden, um den Austausch der Daten (Selektieren, Aktualisieren, Einfügen und Löschen) mit dem SQL Server zu gewährleisten.

HINWEIS Visual Studio 2008 bietet auch die Möglichkeit zu *LINQ to DataSets*. Dabei wird die Kommunikation zwischen SQL Server und der Client-Anwendung wie in bisherigen Versionen über DataSets abgewickelt, und auch die Aktualisierungslogik läuft ganz normal, wie gewohnt, über die DataSets ab. Die Abfrage, Selektierung und Sortierung der Daten innerhalb der DataSets kann dann aber über LINQ-Abfragen erfolgen. Durch die sehr große Ähnlichkeit zu *LINQ to Objects* wollen wir dieses Thema im Rahmen dieses Buches nicht näher vertiefen.

Auch für LINQ to SQL gibt es einen Designer, der dem seines Vorgängers vergleichsweise ähnlich sieht:

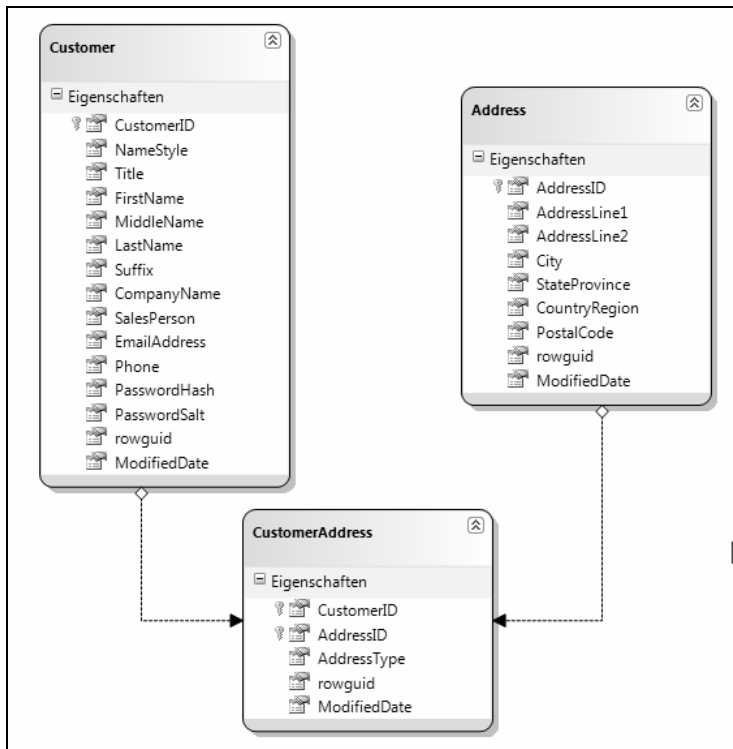


Abbildung 9.2 Der O/R-Designer von VS 2008

Wie Sie sehen, fehlen die TableAdapter, und Verhaltensweisen, die man zuvor über Tabellenattribute einstellen konnte, werden nun über Eigenschaften gesteuert.

Und damit enden die Gemeinsamkeiten auch schon – lassen Sie sich deswegen nicht über die oberflächlichen Ähnlichkeiten hinwegtäuschen: LINQ to SQL verfolgt einen gänzlich anderen Ansatz als die typisierten DataSets (mit denen Sie, falls Sie es wünschten, natürlich auch in dieser Version von Visual Basic noch weiterarbeiten könnten).

LINQ to Objects haben Sie bereits kennen gelernt, und falls nicht, sollten Sie sich auf alle Fälle das vorherige Kapitel zu Gemüte führen. Die gleiche Syntax und ein sehr, sehr ähnlicher Ansatz wird nämlich auch von LINQ to SQL verwendet.

Die ersten Schritte

Learning by Doing ist immer noch der beste Weg, neue Technologien kennen zu lernen, deswegen lassen Sie uns im Folgenden ein kleines LINQ to SQL-Beispiel erstellen.

BEGLEITDATEIEN Unter `.\Samples\Chapter09 - LinqToSql\LinqToSqlDemo` finden Sie die Beispieldateien für dieses Projekt, falls Sie die Beispiele nicht »händisch« nachvollziehen wollen. Denken Sie daran, dass Sie für das Nachvollziehen der Beispiele Zugriff auf eine Instanz von SQL Server 2005 haben müssen und dort die *AdventureWorks*-Beispiele eingerichtet sind.

- Erstellen Sie ein neues Visual Basic-Projekt (als Konsolenanwendung).
- Fügen Sie mithilfe des Projektmappen-Explorers ein neues Element in die Projektmappe ein – den entsprechenden Dialog erreichen Sie über das Kontext-Menü des Projektnamens –, und wählen Sie für das neue Element, wie in der Abbildung zu sehen, die Vorlage *LINQ to SQL-Klasse* aus.
- Nennen Sie sie AdventureWorks.

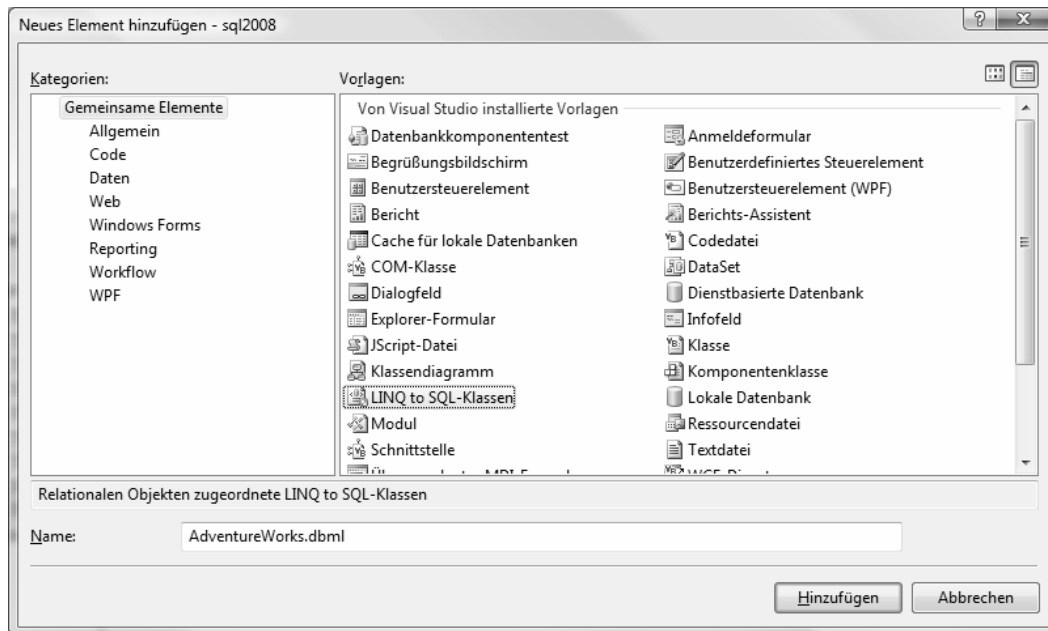


Abbildung 9.3 Hinzufügen einer LINQ to SQL-Klasse

- Schließen Sie den Dialog mit Mausklick auf *Hinzufügen* ab.
- Im nächsten Schritt sehen Sie den so genannten Object Relational Designer (O/R Designer), der es Ihnen gestattet, neue Business-Objektklassen auf Basis von Datenbankobjekten zu entwerfen. Öffnen Sie den Server-Explorer (falls Sie ihn nicht sehen, lassen Sie ihn über das *Ansicht*-Menü anzeigen), öffnen Sie das Kontextmenü von *Datenverbindungen*, und fügen Sie eine neue Verbindung hinzu.

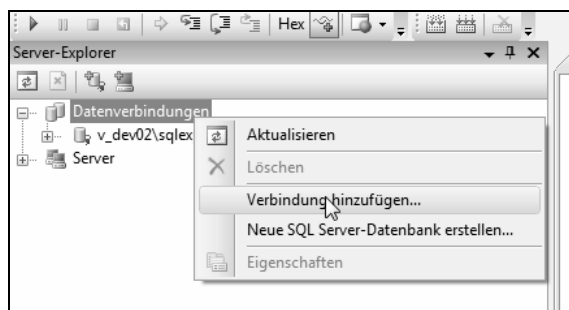


Abbildung 9.4 Erstellen einer neuen Datenverbindung

- Wählen Sie in dem nun folgenden Dialog die SQL Server-Instanz sowie die Datenbank AdventureWorks aus, die wir im folgenden Beispiel verwenden wollen.

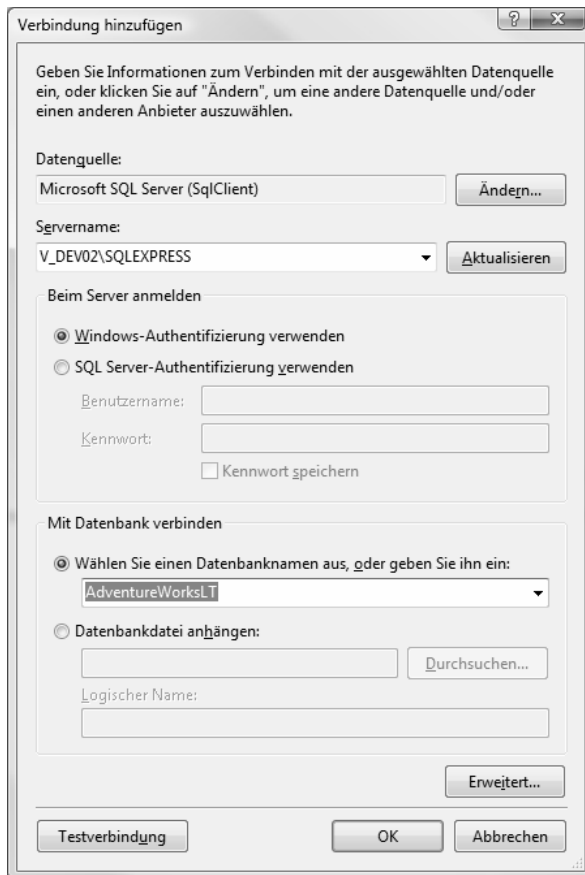


Abbildung 9.5 Auswählen von SQL Server und Datenbank

- Sobald Sie die Verbindung getestet und den Dialog mit *OK* beendet haben, können Sie sich im *Server Explorer* durch die Datenbankstrukturen bewegen. Ziehen Sie die Tabellen *Customer*, *CustomerAddress* und *Address* per Drag & Drop in den O/R-Designer. Sie erhalten eine ähnliche Ansicht wie in *Abbildung 9.2*.
- Dabei ist es zunächst übrigens völlig gleich, ob Sie Tabellen der Datenbank oder Sichten in den O/R-Designer ziehen – das hat bestenfalls Einfluss auf die entsprechenden generierten Klassen. Auch gespeicherte Prozeduren lassen sich auf diese Weise mit dem O/R-Designer verarbeiten, jedoch landen diese, anders als Sichten oder Tabellen, nicht im Hauptbereich sondern im rechten Bereich des O/R-Designers, wie in *Abbildung 9.6* gekennzeichnet.

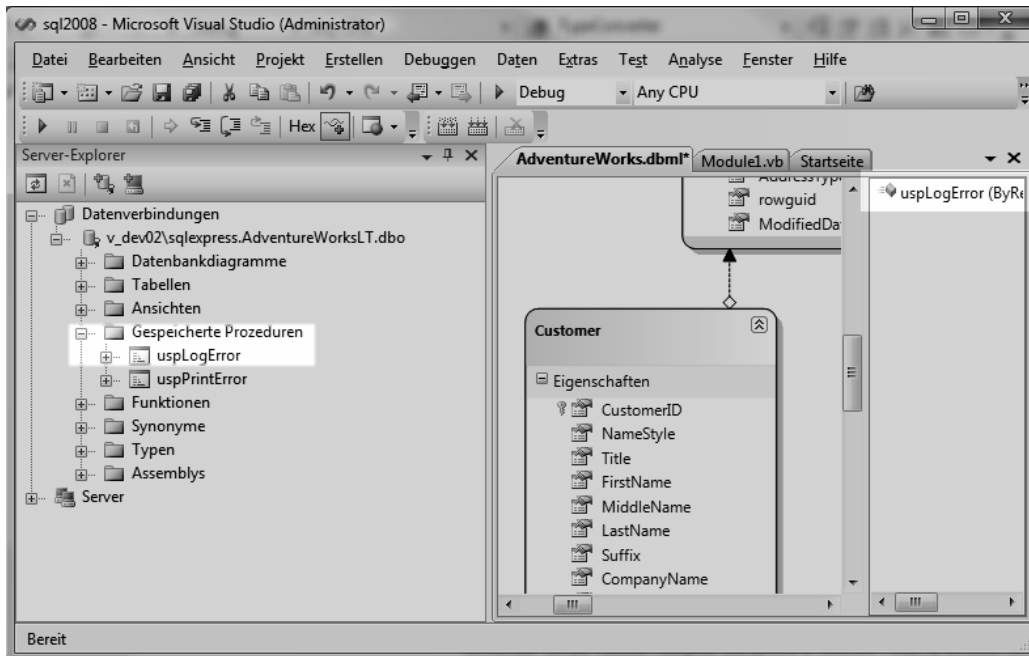


Abbildung 9.6 Hinzufügen der gespeicherten Prozedur uspLogError

- Speichern Sie Ihre Design-Änderungen ab. Wenn Sie mit dem entsprechenden Symbol des Projektmappen-Explorers *Alle Dateien* im Projektmappen-Explorer eingblendet haben, finden Sie dort eine Datei namens *AdventureWorks.designer.vb*.

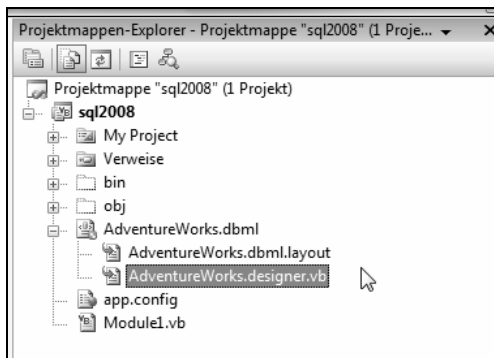


Abbildung 9.7 Der O/R-Designer erstellt eine VB-Codatei

Diese Datei beinhaltet unter anderem den Quellcode für einen so genannten *DataContext*. Der DataContext ist die Verwaltungsinstanz bei LINQ to SQL, etwas, was wir bislang bei LINQ to Objects nicht kennen gelernt haben, weil es nicht benötigt wurde: Der DataContext verwaltet beispielsweise die Verbindungszeichenfolge für den Verbindungsaufbau zur Datenbank und übernimmt die Transaktionssteuerung. Sämtliche Tabellenstrukturen werden im DataContext definiert. Der DataContext übernimmt zur Laufzeit in etwa die Rolle, die der TableAdapter bei typisierten DataSets spielt – er kümmert sich ebenfalls um die Aktualisierungslogik.

- Wechseln Sie nun zu *Module1.vb*, und ändern Sie die von Sub `Main` definierte Methode folgendermaßen ab:

```
Sub Main()  
    Dim datacontext As New AdventureWorksDataContext  
    Dim customeraddress = From adr In datacontext.Addresses _  
        Join custadrassoc In datacontext.CustomerAddresses _  
        On adr.AddressID Equals custadrassoc.AddressID _  
        Join cust In datacontext.Customers _  
        On cust.CustomerID Equals custadrassoc.CustomerID _  
        Where cust.LastName = "Geist" _  
        Select New With {.Title = cust.Title, _  
            .Firstname = cust.Firstname, _  
            .LastName = cust.LastName, _  
            .City = adr.City, _  
            .CountryRegion = adr.CountryRegion, _  
            .Phone = cust.Phone _  
        }  
    For Each ds In customeraddress  
        Console.WriteLine("{0} {1} {2} wohnt in {3}/{4} " & _  
            "und ist unter der Telefonnummer {5} " & _  
            "erreichbar.", ds.Title, _  
            ds.Firstname, ds.LastName, _  
            ds.City, ds.CountryRegion, _  
            ds.Phone)  
    Next  
    Console.WriteLine("Bitte mit Return bestätigen")  
    Console.ReadLine()  
End Sub
```

Die Syntax von LINQ haben Sie bereits in Kapitel 8 »LINQ to Objects« kennen gelernt – und jetzt lernen Sie die Stärken von LINQ kennen: Kennen Sie eine, kennen Sie alle! (Einmal von den Feinheiten abgesehen, die Sie in Ergänzung kennen müssen.)

In dieser Abfrage werden die drei Tabellen *Address*, *CustomerAddress* und *Customer* durch eine `Join`-Klausel miteinander verknüpft. Zudem wird die Abfrage auf den Nachnamen *Geist* beschränkt. Für alle gefundenen Datensätze (hier nur einer) werden anschließend Informationen zu dem Kunden und der Adresse ausgegeben – für die Ergebnismenge wird dazu mit `Select` eine Auflistung mit Instanzen einer entsprechenden anonymen Klasse erstellt.

Die Ausgabe dieses Programms liefert:

```
Mr. Jim Geist wohnt in Puyallup/United States und ist unter der Telefonnummer 724-555-0161 erreichbar.  
Bitte mit Return bestätigen
```

Und jetzt kommt das Entscheidende: Im Gegensatz zu LINQ to Objects wird das Ergebnis dieser Abfrage nicht auf dem Rechner ermittelt, auf dem das Programm läuft, sondern aus der LINQ-Abfrage wird ein SQL-Kommando erstellt, zur Datenbank versendet und das Ergebnis wird anschließend von der Datenbank quasi »abgeholt«.

Sie können das einfach überprüfen, indem Sie vor der For/Each-Schleife der Log-Eigenschaft des Datenkontexts eine TextWriter-Instanz hinzufügen. Wir nutzen hierfür Console.Out:

```
datacontext.Log = Console.Out
```

Die Ausgabe des Programms liefert nun etwas mehr Informationen, und verrät das Geheimnis, wie oder vielmehr mit welchen SQL-Statements die Kommunikation zwischen dem Client und dem SQL Server über den Datenkontext vonstatten geht:

```
SELECT [t2].[Title], [t2].[FirstName] AS [Firstname], [t2].[LastName],
       [t0].[City], [t0].[CountryRegion], [t2].[Phone]
FROM [SalesLT].[Address] AS [t0]
INNER JOIN [SalesLT].[CustomerAddress] AS [t1]
  ON [t0].[AddressID] = [t1].[AddressID]
INNER JOIN [SalesLT].[Customer] AS [t2]
  ON [t1].[CustomerID] = [t2].[CustomerID]
WHERE [t2].[LastName] = @p0
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8

Mr. Jim Geist wohnt in Puyallup/United States und ist unter der Telefonnummer 72
4-555-0161 erreichbar.
Bitte mit Return bestätigen
```

Wie Sie sehen, werden nur die Daten selektiert, die wir auch in dem Select-Ausdruck festgelegt haben. Zudem wird unsere Where-Klausel ebenso an den SQL Server übermittelt. (Der Parameter *@p0* wird, wie in der nächsten Zeile aufgeführt, durch *Geist* ersetzt.)

Diese Umwandlung von LINQ to SQL in eine Select-Anweisung wird durch einen datenbankspezifischen LINQ-Provider durchgeführt, und ich kann Ihnen sehr empfehlen, ein wenig mit verschiedenen Abfragen gegen die AdventureWorks-Datenbank zu experimentieren, um einerseits ein Gefühl für LINQ to SQL an sich, andererseits aber auch eines für die Umsetzung von lokalem LINQ auf »echtes« T-SQL zu bekommen.

Kaskadierte Abfragen

Aufgrund der verzögerten Ausführung von LINQ-Abfragen, die für LINQ to SQL genau so gilt wie für LINQ to Objects (Kapitel 8 hält mehr darüber bereit), können sehr einfache und übersichtliche Sub-Select-Abfragen mit LINQ to SQL durchgeführt werden.

Im folgenden Beispiel wiederholen wir die Adressabfrage für unseren Herrn Geist. Diesmal interessieren uns jedoch nicht die Kundendaten, sondern lediglich seine Adresse:

```
Dim datacontext As New AdventureWorksDataContext
Dim cust = From row In datacontext.Customers _
           Where row.LastName = "Geist"
Dim custadrassoc = From row In datacontext.CustomerAddresses _
                   Where cust.Any(Function(c) c.CustomerID = row.CustomerID)
Dim adr = From row In datacontext.Addresses _
           Where custadrassoc.Any(Function(a) a.AddressID = row.AddressID)
```

```

datacontext.Log = Console.Out
For Each ds In adr
    Console.WriteLine("Herr Geist wohnt in {0}/{1}", ds.City, ds.CountryRegion)
Next
Console.WriteLine("Bitte mit Return bestätigen")
Console.ReadLine()
End Sub

```

Als erstes selektieren wir die Kundendaten des Herrn Geist. Diese Abfrage wird in `cust` gespeichert. Als nächstes selektieren wir aus der Tabelle `CustomerAddresses` den Datensatz, der zur Kundennummer von Herrn Geist passt. Diese Abfrage wird in `custadrassoc` gespeichert.

Nun können wir in der Tabelle `Addresses` suchen, und wir benutzen dazu die `Any`-Klausel, um den entsprechenden Datensatz zu finden, der den Kriterien entspricht, da die Ausgangsliste natürlich ebenfalls komplett durchsucht werden muss – und das ist genau das, was `Any` leistet. Diese Abfrage wird in `adr` gespeichert.

HINWEIS Noch einmal zur Erinnerung: Das Prinzip der verzögerten Ausführung bestimmt, dass bislang noch keine Abfrage ausgeführt wurde. Es wurden lediglich die Abfragen definiert. Erst durch die `For/Each`-Schleife werden die drei Abfragen zu einer zusammengesetzt und ausgeführt, weil – wie wir es bei LINQ to Objects bereits kennen gelernt haben – das erste Mal der Versuch gestartet wird, ein Element der Ergebnismenge abzurufen.

Als Ergebnis erhalten wir die Anschrift von Herrn Geist (und, da das Logging für die SQL-Abfragegenerierung noch aktiv ist, auch die eigentliche SQL-Abfrage, die an den SQL-Server gesendet wird).

```

SELECT [t0].[AddressID], [t0].[AddressLine1], [t0].[AddressLine2], [t0].[City],
       [t0].[StateProvince], [t0].[CountryRegion], [t0].[PostalCode], [t0].[rowguid],
       [t0].[ModifiedDate]
FROM [SalesLT].[Address] AS [t0]
WHERE EXISTS(
    SELECT NULL AS [EMPTY]
    FROM [SalesLT].[CustomerAddress] AS [t1]
    WHERE ([t1].[AddressID] = [t0].[AddressID]) AND (EXISTS(
        SELECT NULL AS [EMPTY]
        FROM [SalesLT].[Customer] AS [t2]
        WHERE ([t2].[CustomerID] = [t1].[CustomerID]) AND ([t2].[LastName] = @p0)
    ))
)
)
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8

Herr Geist wohnt in Puyallup/United States.
Bitte mit Return bestätigen

```

Übrigens: Wenn Sie schon ein wenig mit den verschiedenen Abfragemöglichkeiten von LINQ to Objects bzw. LINQ to SQL herumexperimentiert haben, haben Sie sich möglicherweise gefragt, warum wir hier im Beispiel die `Any`-Methode von den Abfragen `cust` und `custadrassoc` verwendet haben, obwohl `All` in diesem Zusammenhang sinnvoller erscheinen könnte. Um Ihnen diesen Unterschied vor Augen zu führen, ändern wir den Code ab und verwenden nun die `All`-Methode für das weitere Selektieren der Daten:

```

Dim datacontext As New AdventureWorksDataContext
Dim cust = From row In datacontext.Customers _
            Where row.LastName = "Geist"
Dim custadrassoc = From row In datacontext.CustomerAddresses _
                   Where cust.All(Function(c) c.CustomerID = row.CustomerID)
Dim adr = From row In datacontext.Addresses _
           Where custadrassoc.All(Function(a) a.AddressID = row.AddressID)

datacontext.Log = Console.Out
For Each ds In adr
    Console.WriteLine("Herr Geist wohnt in {0}/{1}", ds.City, ds.CountryRegion)
Next
Console.WriteLine("Bitte mit Return bestätigen")
Console.ReadLine()

```

Wenn das Programm nun gestartet wird erhalten wir folgende Ausgabe:

```

SELECT [t0].[AddressID], [t0].[AddressLine1], [t0].[AddressLine2], [t0].[City],
       [t0].[StateProvince], [t0].[CountryRegion], [t0].[PostalCode], [t0].[rowguid],
       [t0].[ModifiedDate]
FROM [SalesLT].[Address] AS [t0]
WHERE NOT (EXISTS(
    SELECT NULL AS [EMPTY]
    FROM [SalesLT].[CustomerAddress] AS [t1]
    WHERE ((
        (CASE
            WHEN [t1].[AddressID] = [t0].[AddressID] THEN 1
            ELSE 0
        END)) = 0) AND (NOT (EXISTS(
    SELECT NULL AS [EMPTY]
    FROM [SalesLT].[Customer] AS [t2]
    WHERE ((
        (CASE
            WHEN [t2].[CustomerID] = [t1].[CustomerID] THEN 1
            ELSE 0
        END)) = 0) AND ([t2].[LastName] = @p0)
    )))
))
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8

Herr Geist wohnt in Puyallup/United States.
Bitte mit Return bestätigen

```

Diese Abfrage funktioniert ebenfalls, aber wie Sie sehen, ist das erstellte Select deutlich komplexer.

Die All-Methode betrachtet jeden Datensatz, während Any nur die Datensätze betrachtet, die den Kriterien entsprechen.

Daten verändern und speichern

Wenn Sie mit den aus den Abfragen erhaltenen Daten weiterarbeiten möchten, ohne sie jedes Mal neu zu selektieren, wandeln Sie das Ergebnis der Abfrage in eine Liste oder ein Array um. Dieses erfolgt beispielsweise mit den Methoden `ToList` oder `ToArray` – wir haben das im LINQ to Objects-Kapitel bereits kennengelernt.

HINWEIS Das gilt für LINQ to SQL umso mehr, als dass Sie es nur in den wenigsten Fällen wirklich wünschen, dass die eigentliche SQL-Abfrage immer und immer wieder zur Ausführung an den SQL-Server übermittelt wird. In den meisten Fällen wird es reichen, die Abfrage ein Mal auszuführen, und die Ergebnisliste schließlich mit `ToArray` bzw. `ToList` von der eigentlichen Abfrage zu trennen.

```
Dim datacontext As New AdventureWorksDataContext
Dim cust = From row In datacontext.Customers _
           Where row.LastName = "Geist"
datacontext.Log = Console.Out
Dim custlist = cust.ToList
```

Sie können jetzt mit der Liste arbeiten, ohne dass die Daten bei jedem Zugriff neu geladen werden.

Konstruieren wir unser Beispiel weiter, und nehmen wir an, dass unser Herr Geist die Telefongesellschaft gewechselt und nun eine neue Telefonnummer bekommen hat:

```
Dim herrGeist = custlist(0)
herrGeist.Phone = "555/1234567"
```

LINQ to SQL überwacht die geladenen Daten und kann so auch Veränderungen an diesen erkennen. Anhand dieser Veränderungen können Insert-, Update- und Delete-SQL-Kommandos für die Datenbank erstellt und die Änderungen automatisiert zurückgeschrieben werden.

Dieses erfolgt durch Aufruf der Methode `SubmitChanges` am verwendeten `DataContext`.

```
datacontext.SubmitChanges()
```

Wir verwenden folgendes Beispiel, um uns die Ausgaben des `DataContext` anzusehen:

```
Dim datacontext As New AdventureWorksDataContext
Dim cust = From row In datacontext.Customers _
           Where row.LastName = "Geist"

datacontext.Log = Console.Out
Dim custlist = cust.ToList
Dim herrGeist = custlist(0)

Console.WriteLine("Ändern der Telefonnummer")
herrGeist.Phone = "555/1234567"

Console.WriteLine("Änderungen Speichern")
datacontext.SubmitChanges()
```

Es werden folgende Ausgaben erzeugt:

```
SELECT [t0].[CustomerID], [t0].[NameStyle], [t0].[Title], [t0].[FirstName], [t0]
.....(Ausgabe gekürzt) .....
FROM [SalesLT].[Customer] AS [t0]
WHERE [t0].[LastName] = @p0
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8
Ändern der Telefonnummer
Änderungen speichern
UPDATE [SalesLT].[Customer] SET [Phone] = @p12
WHERE ([CustomerID] = @p0) AND (NOT ([NameStyle] = 1)) AND ([Title] = @p1) AND (
[FirstName] = @p2) AND ([MiddleName] IS NULL) AND ([LastName] = @p3) AND ([Suffi
x] IS NULL) AND ([CompanyName] = @p4) AND ([SalesPerson] = @p5) AND ([EmailAddre
ss] = @p6) AND ([Phone] = @p7) AND ([PasswordHash] = @p8) AND ([PasswordSalt] =
@p9) AND ([rowguid] = @p10) AND ([ModifiedDate] = @p11)
-- @p0: Input Int (Size = 0; Prec = 0; Scale = 0) [37]
.....(Ausgabe gekürzt) .....
-- @p10: Input UniqueIdentifier (Size = 0; Prec = 0; Scale = 0) [c6ebb29a-cc67-4
59c-90e3-339e0f912906]
-- @p11: Input DateTime (Size = 0; Prec = 0; Scale = 0) [13.10.2004 11:15:07]
-- @p12: Input NVarChar (Size = 11; Prec = 0; Scale = 0) [555/1234567]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8
```

Der Parameter @p12 beinhaltet nun unsere geänderte Telefonnummer. Die Änderungen wurden also zurück in die Datenbank gespeichert.

Einfügen und löschen von Datensätzen

SubmitChanges repliziert Änderungen an der Auflistung im vorherigen Beispiel in die Datenbank. Beim Einfügen *neuer* Datensätze gibt es die Möglichkeit, direkt mit der InsertOnSubmit-Methode zu arbeiten, die als Argument eine Instanz der Klasse übernimmt, die einen Datensatz abbildet. Das Löschen eines Datensatzes funktioniert wiederum mit DeleteOnSubmit. Auch hier werden Änderungen nicht sofort, sondern erst *nach* einem abschließenden SubmitChanges an die Datenbank übermittelt. Das folgende Beispiel verdeutlicht auch diese Funktionsweise.

```
Dim datacontext As New AdventureWorksDataContext, customerObjektInstanz As Customer
'Im Objektinitialisierer müssen die "Muss"-Felder (die DBNull nicht enthalten dürfen) angegeben werden
datacontext.Customers.InsertOnSubmit(New Customer() with { .FirstName="Andreas", .LastName="Belke", ...})
'So würden Sie eine vorhandene (aber instanziierte!) customer-Instanz löschen
datacontext.Customers.DeleteOnSubmit(customerObjektInstanz)
'Erst hier werden eingefügte/gelöschte Datensätze in die entsprechende SQL Server-Tabelle übernommen
datacontext.SubmitChanges()
```

Concurrency-Check (Schreibkonfliktprüfung)

Was aber, wenn die Daten bereits von einem anderen Anwender geändert wurden? In diesem Fall wird eine ChangeConflictException ausgelöst. LINQ überprüft beim Aktualisieren, ob der zu ändernde Datensatz noch in der Version vorliegt, in der er ursprünglich geladen wurde. Ist das nicht der Fall, wird die Ausnahme ausgelöst. Standardmäßig werden alle Felder einer Tabelle überprüft. Dieses kann jedoch im Designer für jedes Feld einzeln eingestellt werden.

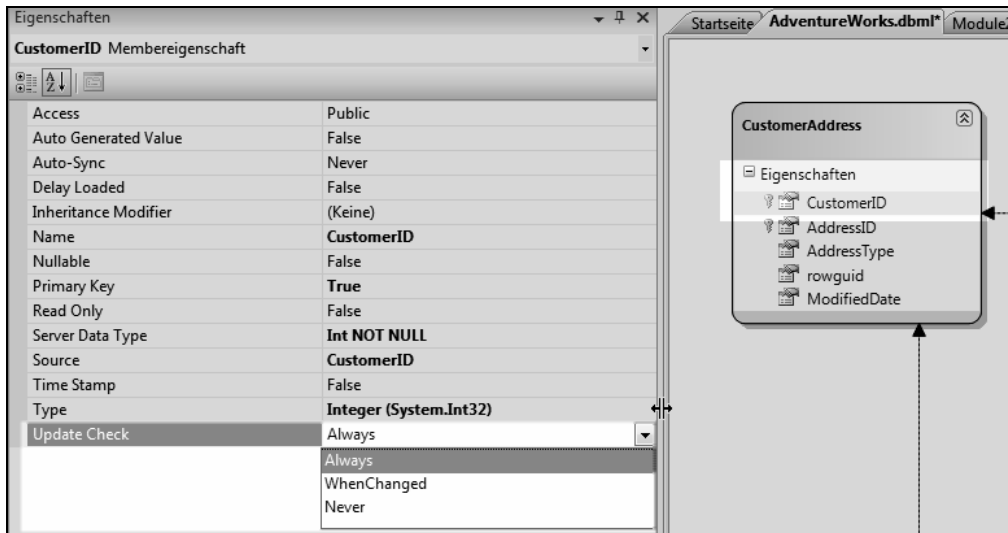


Abbildung 9.8 Ändern der Kollisionsüberprüfung eines Attributes

Wenn eine Tabelle einen Änderungszeitstempel besitzt oder eine Versionsnummer des Datensatzes gepflegt wird, können die Daten auch anhand dieser Angaben auf Eindeutigkeit geprüft werden. Hierzu muss die Time Stamp Eigenschaft bei dem jeweiligen Attribut auf true gesetzt werden.

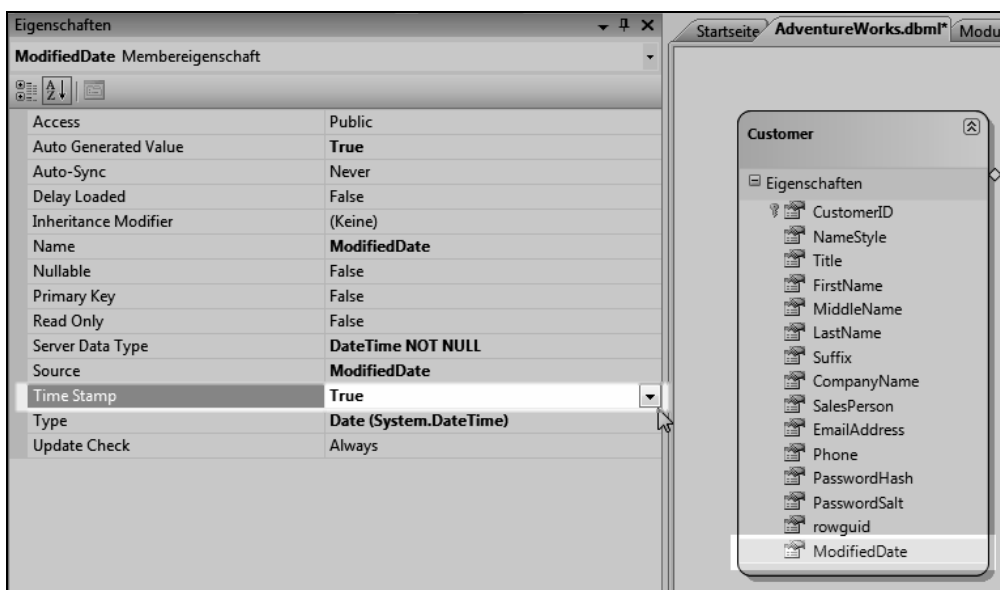


Abbildung 9.9 Verwenden eines Zeitstempels zur Kollisionsüberprüfung

HINWEIS Häufig wird anstelle von Time Stamp auch von IsVersion gesprochen. Tatsächlich wird die Einstellung von Time Stamp in dem ColumnAttribut unter IsVersion gespeichert. Es handelt sich also letzten Endes um die gleiche Eigenschaft, die im Designer nur »über Umwege« angesprochen wird.

WICHTIG Wird bei einem Feld eine Tabelle Time Stamp auf true gesetzt, werden für alle anderen Felder die Update Check-Angaben ignoriert.

Ändern Sie für unser Beispiel im O/R Designer das ModifiedDate der Tabelle Customer. Setzen Sie die Eigenschaft Time Stamp auf true. Ändern Sie den Code anschließend folgendermaßen ab:

```
Console.WriteLine("Ändern der Telefonnummer")
herrGeist.Phone = "555/123456"
```

WICHTIG Ändern Sie auch die Telefonnummer, die Sie herrGeist.Phone zuweisen. Da das Programm bereits einmal gelaufen ist, wurde die Telefonnummer bereits in der Datenbank geändert. Demzufolge wird diese Telefonnummer auch wieder geladen und das Zuweisen derselben Telefonnummer wäre keine Änderung und es würden auch keine Update-Kommandos erstellt.

```
SELECT [t0].[CustomerID], [t0].[NameStyle], [t0].[Title], [t0].[FirstName],
[t0].[MiddleName], [t0].[LastName], [t0].[Suffix], [t0].[CompanyName],
[t0].[SalesPerson], [t0].[EmailAddress], [t0].[Phone], [t0].[PasswordHash],
[t0].[PasswordSalt], [t0].[rowguid], [t0].[ModifiedDate]
FROM [SalesLT].[Customer] AS [t0]
WHERE [t0].[LastName] = @p0
-- @p0: Input NVarChar (Size = 5; Prec = 0; Scale = 0) [Geist]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8
```

```
Ändern der Telefonnummer
Änderungen speichern
UPDATE [SalesLT].[Customer]
SET [Phone] = @p2
WHERE ([CustomerID] = @p0) AND ([ModifiedDate] = @p1)
```

```
SELECT [t1].[ModifiedDate]
FROM [SalesLT].[Customer] AS [t1]
WHERE ((@ROWCOUNT) > 0) AND ([t1].[CustomerID] = @p3)
-- @p0: Input Int (Size = 0; Prec = 0; Scale = 0) [37]
-- @p1: Input DateTime (Size = 0; Prec = 0; Scale = 0) [13.10.2004 11:15:07]
-- @p2: Input NVarChar (Size = 11; Prec = 0; Scale = 0) [555/1234567]
-- @p3: Input Int (Size = 0; Prec = 0; Scale = 0) [37]
-- Context: SqlProvider(Sql2005) Model: AttributedMetaModel Build: 3.5.21022.8
```

Bitte mit Return bestätigen

HINWEIS Falls im obigen Bildschirmauszug der Teil mit dem Update-Kommando fehlen sollte, müssen Sie die Telefonnummer in der Datenbank ändern! Wie Sie sehen, wurde bei dem Update in der where-Klausel nur auf das ModifiedDate geprüft.

Eine weitere Möglichkeit, die Kollisionsüberprüfung zu steuern, liegt in der Methode `SubmitChanges` selbst. Es existiert eine überladene Version, bei der eine `ConflictMode`-Einstellung angegeben werden kann.

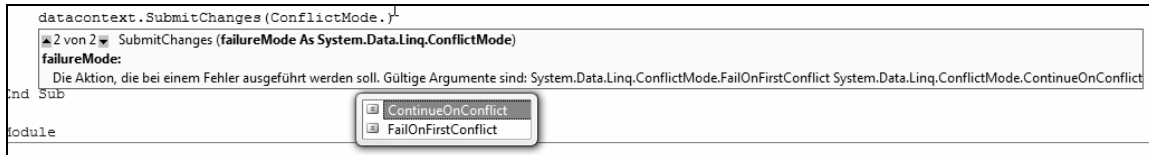


Abbildung 9.10 Mögliche ConflictModes-Einstellungen

Mögliche Werte sind:

- `ContinueOnConflict`: Es bedeutet *nicht*, dass Fehler durch parallele Zugriffe ignoriert werden, sondern es ist vielmehr so, dass auftretende Kollisionen quasi »gesammelt« und erst zum Schluss durch eine entsprechende Ausnahme »bekannt gegeben« werden.
- `FailOnFirstConflict`: Schon beim Auftreten des *ersten* Konfliktes wird eine Ausnahme ausgelöst.

Transaktionen

Standardmäßig werden Änderungen an die Datenbank durch den `DataContext` automatisch als Transaktion übermittelt, falls keine explizite Transaktion im Gültigkeitsbereich vorgefunden wird. Um eine übergreifende Transaktion zu nutzen, stehen zwei Möglichkeiten zur Verfügung.

TransactionScope (Transaktionsgültigkeitsbereich)

Um Transaktionen mit `TransactionScope` nutzen zu können, muss die `System.Transactions.dll` Assembly in das Projekt aufgenommen werden.

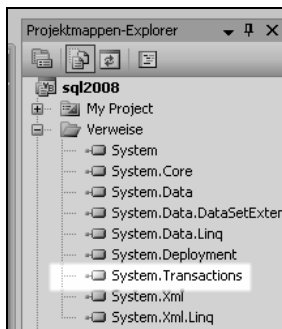


Abbildung 9.11 Einbinden der System.Transaction.dll Assembly

```
Dim ts As New Transactions.TransactionScope
Using ts
    DataContext.ExecuteCommand("exec protokolliere_aenderungen")
End Using
```

```
datacontext.SubmitChanges(ConflictMode.ContinueOnConflict)
ts.Complete()
End Using
```

Exemplarisch für weitere Änderungen an dem Datenhaushalt wurde hier die `ExecuteCommand`-Methode des `DataContext` verwendet, um eine datenverändernde gespeicherte Prozedur aufzurufen.

`TransactionScope` sorgt automatisch für das Durchführen eines Rollbacks, falls Fehler in der Transaktion auftreten. Sie müssen lediglich dafür sorgen, dass zum richtigen Zeitpunkt die `Complete`-Methode des `TransactionScope`-Objekts aufgerufen wird, um die geänderten Daten in der Datenbank mit einem *Commit* festzuschreiben.

Verwenden der Transaktionssteuerung des DataContext

Gerade für Anwendungen die noch mehr ADO.Net-orientiert sind, existiert eine weitere Möglichkeit mit Transaktionen zu arbeiten.

`DataContext` besitzt eine `Transaction`-Eigenschaft. Es ist ebenso möglich, über diese Eigenschaft Transaktionen zu steuern. Jedoch ist hier ein deutlicher Mehraufwand notwendig – der folgende Codeauszug soll das exemplarisch demonstrieren:

```
datacontext.Transaction = datacontext.Connection.BeginTransaction()
Try
    datacontext.ExecuteCommand("exec protokolliere_aenderungen")
    datacontext.SubmitChanges()
    datacontext.Transaction.Commit()
Catch ex As Exception
    datacontext.Transaction.Rollback()
    Throw ex
Finally
    datacontext.Transaction = Nothing
End Try
```

Als erstes muss eine neue Transaktion eingeleitet werden. Dazu wird auf dem `Connection`-Objekt des `DataContexts` die `BeginTransaction`-Methode aufgerufen. Sie erstellt ein neues Transaktionsobjekt. Änderungen am Datenbankhaushalt müssen nun in einem `Try-Catch-Block` durchgeführt werden. Im Fehlerfall wird eine Ausnahme ausgelöst. Sie muss abgefangen werden, um das Rollback durchzuführen.

Zudem sollte im `Finally-Block` das Transaktionsobjekt wieder auf `Nothing` gesetzt werden. So startet der `DataContext` automatisch eine neue Transaktion bei einem `SubmitChanges`, sofern nicht explizit über `BeginTransaction` eine neue Transaktion begonnen wird.

Was, wenn LINQ einmal nicht reicht

Unter Umständen kommen Sie einmal in eine Situation, in der Sie eine `Select`-Abfrage mit LINQ nicht abbilden können. In diesem Fall können Sie selbst das T-SQL-Kommando für SQL Server erstellen, um die Datenbank abzufragen. Das Schöne daran ist, dass Sie, nachdem die Daten geladen wurden, nicht auf die LINQ-Features zu verzichten brauchen.

Der DataContext stellt eine Methode namens `ExecuteQuery` bereit, mit dem beliebige T-SQL-Abfragen direkt an den SQL Server geschickt werden können:

```
Dim allrows = From ds In datacontext.Customers
Dim cmd As Common.DbCommand = datacontext.GetCommand(allrows)
Dim sel = cmd.CommandText & " where lastname = {0}"
Dim res = datacontext.ExecuteQuery(Of Customer)(sel, New Object() {"Geist"})
For Each ds In res
    Console.WriteLine("Herr Geist hat die KundenNr {0}", ds.CustomerID)
Next
```

Wir definieren hier im Beispielcode eine Abfrage auf der Tabelle `Customers` und speichern sie in `allrows`. Der DataContext stellt die Methode `GetCommand` zur Verfügung, mit der man das generierte Select-Kommando erfragen kann.

Dieses wird als `cmd` gespeichert. Der `CommandText` enthält das eigentliche Select-Kommando im Klartext.

Wir erweitern es um

```
where lastname = {0}
```

Bei `{0}` handelt es sich um einen Parameter, und zwar genauer gesagt, um den ersten Parameter, welcher der Abfrage übergeben wurde. Den zweiten erreichen Sie mit `{1}`, usw. Beim Aufrufen der Methode `ExecuteQuery` muss die Klasse angegeben werden, die einen Datensatz der Tabelle aufnehmen kann. In diesem Fall die Klasse `Customer` (demgegenüber steht `Customers`, welches eine *Tabelle* von `Customer`-Instanzen ist).³ Als Parameter wird das Select-Kommando übergeben, sowie ein Objekt-Array, das die im *Select* festgelegten Parameter in der entsprechenden Reihenfolge enthält (die Parameter können ebenso kommasepariert – also ohne Verwendung eines Arrays – übergeben werden). In diesem Fall wird nur ein Parameter vom Typ `String` übergeben, nämlich »Geist«. Als Ergebnis erhalten wir:

```
Herr Geist hat die KundenNr 37
```

Die Methode `ExecuteQuery` ist für Datenbankabfragen gedacht. Für Datenbankänderungen existiert die Methode `ExecuteCommand` des DataContext.

Als Beispiel:

```
datacontext.ExecuteCommand("update saleslt.customer set phone={0} where lastname={1}", "555-987654321", "Geist")
```

³ Die Namen werden durch den O/R-Designer von LINQ to SQL vergeben – leider haben wir hier Abweichungen in der deutschen Visual Basic 2008 Express Edition und der Visual Studio 2008 Professional Edition (deutsche Vorabversion auf Basis des RC) bei der automatischen Namensvergabe für die jeweiligen Objektamen feststellen müssen. Es ist davon auszugehen, dass diese Inkonsistenzen in der finalen Version behoben sein werden.

