Kapitel 1

Einführung

In diesem Kapitel:

Was ist ein O/R-Mapper und wozu braucht man ihn? 20
Architektur des ADO.NET Entity Framework 25
Das Entitätenmodell 25
Wichtige Bestandteile des EDM 27
Praktischer Entwurf eines Entitätenmodells 53
T4-Vorlagen 55
Metadata Workspace 60
Windows Workflow Foundation 62

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Schon seit .NET Framework 3.5 Service Pack 1 bietet Microsoft einen eigenen O/R-Mapper an: das ADO.NET Entity Framework (EF), dem hier ein ganzes Buch gewidmet ist. (Was genau ein O/R-Mapper ist, erfahren Sie gleich.)

Dieser O/R-Mapper wurde mit .NET Framework 4.0 stark überarbeitet und ist nun in einer erweiterten Version mit graphischem Designer verfügbar. Das EF kann als technische Weiterentwicklung zu LINQ to SQL gesehen werden.

Während LINQ to SQL noch recht eng mit dem SQL Server als Schicht zur Datenspeicherung verwoben ist, ist dies beim ADO.NET Entity Framework nicht mehr der Fall. Die Datenbank im Hintergrund ist stark abstrahiert und es ist denkbar, Anwendungen zu schreiben, die völlig unabhängig von der verwendeten Datenbanktechnologie sind.

Das ADO.NET Entity Framework wurde bei seiner Entwicklung so konzipiert, dass Entwickler Datenzugriffsanwendungen erstellen können, indem sie für ein konzeptionelles Anwendungsmodell und nicht für ein relationales Datenbankschema programmieren. Das Ziel war es dabei auch, die Menge des Codes und den Wartungsaufwand zu verringern, die für datenorientierte Anwendungen erforderlich sind.

Dieses erste Kapitel legt nun die Grundlagen für das Verständnis, das bei der Arbeit mit dem EF benötigt wird. Es erklärt was ein O/R-Mapper eigentlich ist, wozu er eingesetzt werden kann (oder sollte) und beschreibt die einzelnen Komponenten, die dabei eine Rolle spielen.

Was ist ein O/R-Mapper und wozu braucht man ihn?

Vorneweg sei erklärt, dass O/R-Mapper für *Objektrelationaler Mapper* steht und eine Schicht zwischen Anwendung und Datenbank darstellt. Dabei kümmert er sich um das komplette Mapping zwischen Codeobjekten und Tabellen auf der Datenbankseite. Dieser Vorgang ist für den Entwickler unsichtbar und das ist auch durchaus gewollt, auch wenn ein gewisses Grundverständnis darüber, wie eine Abfrage später ausgeführt wird, sicherlich nicht schaden kann.

Prinzipiell kann man sich gerade zu Beginn die Frage stellen, warum überhaupt einen O/R-Mapper verwenden? Schließlich gibt es eine schiere Menge von Programmen auf diesem Planeten, die dies nicht tun und trotzdem gut laufen. Der Hauptgrund ist einfach: Schnellere Entwicklung mit weniger Fehlerpotential, da es so einfach möglich ist, auf Spalten einer Tabellenzeile als Eigenschaft einer Entität/Klasse zuzugreifen. Die gesamte Tabelle wird als Auflistung von Instanzen dieser Klassen dargestellt. Auf diese Weise ist es möglich, mit Unterstützung der IntelliSense-Funktion des Editors sich ein wenig bei der Quellcodeerstellung unterstützen zu lassen.

Allerdings sei auch gesagt, dass diese Erleichterung schnell zum Handicap werden kann, wenn ungewöhnliche Datenzugriffe realisiert werden müssen. In solchen Fällen gibt es immer zwei Gedanken, die es wert sind, kurz durchdacht zu werden. Erstens: Wird der Zugriff wirklich in dieser Form benötigt? Das eine einfache Realisierung mittels eines O/R-Mappers nicht machbar ist, kann ein Indiz dafür sein, dass der Grundgedanke des Zugriffs der Funktionsweise einer relationalen Datenbank widerspricht. Zweitens: Muss ich den Zugriff mittels des O/R-Mapper durchführen? Schließlich stehen Technologien wie ADO.NET trotz O/R-Mapper zur Verfügung.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

WICHTIG Vermeiden Sie nach Möglichkeit eine Mischkultur! Konsequent einen O/R-Mapper zu nutzen und nur für die absoluten Ausnahmen, in denen dies nicht möglich oder effizient ist, eine direktere Zugriffstechnologie zu nutzen, sollte das Ziel sein. Eine Anwendung, die quasi »auf beiden Hochzeiten tanzt«, ist weder einfach zu verstehen, noch leicht zu warten. Und der Vorteil eines Mappers ist damit schnell verpufft. Notfalls steht mit Entity SQL, das später in diesem Kapitel noch angesprochen wird, ein guter Kompromiss zur Verfügung, der absolute Flexibilität auch beim Einsatz des ADO.NET Entity Framework bietet.

Für Abfragen stehen somit LINQ to Entities und der EF-eigene SQL-Dialekt *Entity SQL* bereit. Beiden Themen ist in diesem Buch ein komplettes Kapitel gewidmet. Trotzdem seien an dieser Stelle zwei kleine Beispiele gestattet, damit der hier einsteigende Leser eine anschauliche Idee bekommt, welche Art der Abfrage wie aussieht.

Zunächst eine LINQ to Entities-Abfrage:

```
// Objektkontext für LINQ-Abfrage erstellen. Durch den hier verwendeten parameterlosen Konstruktor
// wird die benötigte Verbindungszeichenfolge aus der Konfiguration der Anwendung verwendet
TonisTortenTraum.TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities();

' Die eigentliche LINQ-Abfrage, die allerdings an dieser Stelle noch nicht(!) ausgeführt wird
var KundenMitK = from k in TonisTortenTraumContext.Kunden where k.Name.StartsWith("K") orderby k.PLZ select k;
```

Listing 1.1 LINQ to Entities mit dem ADO.NET Entity Framework im Einsatz (C#)

Und als VB.NET-Quellcode:

```
Option Infer On

' Objektkontext für LINQ-Abfrage erstellen. Durch den hier verwendeten parameterlosen Konstruktor
' wird die benötigte Verbindungszeichenfolge aus der Konfiguration der Anwendung verwendet
Dim TonisTortenTraumContext As New TonisTortenTraumEntities()

' LINQ-Abfrage, die allerdings an dieser Stelle noch nicht(!) ausgeführt wird
Dim KundenMitK = From k In TonisTortenTraumContext.Kunden Where k.Name.StartsWith("K") Order By k.PLZ
```

Listing 1.2 LINQ to Entities mit dem ADO.NET Entity Framework im Einsatz (VB.NET)

Abfragen mit Entity SQL verwenden die Entity Client-API und weisen an sich eine gewisse Ähnlichkeit zu SQL-Dialekten aus, auch wenn es einige empfindliche Unterschiede gibt.

```
// Verbindungsobjekt erstellen und öffnen
// Der Name bezieht sich auf einen Schlüssel in der Konfiguration
using (EntityConnection con = new EntityConnection(csb.ToString()))
{
    // Das spätere Schließen übernimmt der using-Block
    con.Open();

    using (EntityCommand cmd = con.CreateCommand())
    {
        // Art und Inhalt der Abfrage festlegen
        cmd.CommandType = CommandType.Text;
```

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

```
        cmd.CommandText = "SELECT K.Name FROM TonisTortenTraumEntities.Kunden AS K WHERE K.Name LIKE 'K*';";

        // Datareader erzeugen. Dabei muss
        // CommandBehavior.SequentialAccess als Wert für CommandBehavior festgelegt werden
        using (EntityDataReader dr =
            cmd.ExecuteReader(CommandBehavior.SequentialAccess))
        {
            // Alle Zeilen durchlaufen
            while (dr.Read())
            {
                // Und ausgeben
                Debug.Print("Name='{0}'", dr["Name"]);
            }
        }
    }
}
```

Listing 1.3 Entity SQL mit dem ADO.NET Entity Framework im Einsatz (C#)

Und als VB.NET-Quellcode:

```
' Verbindungsobjekt erstellen und öffnen
' Der Name bezieht sich auf einen Schlüssel in der Konfiguration
Using con As New EntityConnection("Name=TonisTortenTraumEntities;")
    ' Das spätere Schließen übernimmt der using-Block
    con.Open()

    Using cmd As EntityCommand = con.CreateCommand()
        ' Art und Inhalt der Abfrage festlegen
        cmd.CommandType = CommandType.Text
        cmd.CommandText = "SELECT K.Name FROM TonisTortenTraumEntities.Kunden AS K WHERE K.Name LIKE 'K*';"

        ' Datareader erzeugen. Dabei muss
        ' CommandBehavior.SequentialAccess als Wert für CommandBehavior festgelegt werden
        Using dr As EntityDataReader = cmd.ExecuteReader(CommandBehavior.SequentialAccess)
            ' Alle Zeilen durchlaufen
            While dr.Read
                ' Und testweise ausgeben
                Debug.Print("Name='{0}'", dr("Name"))
            End While
        End Using
    End Using
End Using
```

Listing 1.4 Entity SQL mit dem ADO.NET Entity Framework im Einsatz (VB.NET)

Der Einsatz der API gestaltet sich damit ähnlich wie ADO.NET mit Verbindungs-Objekt, Command-Objekt, DataReader-Objekt, etc[1]. Für weitere Beispiele und Details siehe Kapitel 6 »Die EntityClient-API«.

[1] Was auch nicht verwunderlich ist, da die gleichen Basisklassen aus dem `System.Data.Common`–Namensraum verwendet werden.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Um doch Änderungen an Daten mithilfe von Entity SQL durchzuführen, können komplette Entitäten abgefragt werden, an denen Änderungen vorgenommen und gespeichert werden können. Auch dies soll als Beispiel an dieser Stelle vorweggenommen werden. Im Kapitel 5 »Entity SQL« finden Sie hier mehr Details und Beispiele.

```
// Objektkontext erstellen
TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities();

const string ESql = "SELECT VALUE K FROM TonisTortenTraumEntities.Kunden AS K WHERE K.Name LIKE 'K*'";

// Abfrage anlegen, jedoch noch nicht ausführen!
ObjectQuery<Kunden> KundenMitK =
    TonisTortenTraumContext.CreateQuery<Kunden>(ESql);

// Abfrage ausführen, Daten ausgeben und modifizieren
foreach (Kunden k in KundenMitK)
{
    Debug.Print("{0} in {1}", k.Name, k.Ort);
    k.Name = "*" + k.Name;
}

// Änderungen speichern
TonisTortenTraumContext.SaveChanges(SaveOptions.AcceptAllChangesAfterSave);
```

Listing 1.5 Entity SQL und die generische CreateQuery()-Methode im Einsatz (C#)

Und als VB.NET-Quellcode:

```
' Objektkontext erstellen
Dim TonisTortenTraumContext As New TonisTortenTraumEntities()

Const ESql As String = "SELECT VALUE K FROM TonisTortenTraumEntities.Kunden AS K WHERE K.Name LIKE 'K*'"

' Abfrage anlegen, jedoch noch nicht ausführen!
Dim KundenMitK As ObjectQuery(Of Kunden) =
    TonisTortenTraumContext.CreateQuery(Of Kunden)(ESql)

' Abfrage ausführen, Daten ausgeben und modifizieren
For Each k As Kunden In KundenMitK
    Debug.Print("{0} in {1}", k.Name, k.Ort)
    k.Name = "*" + k.Name
Next

' Änderungen speichern
TonisTortenTraumContext.SaveChanges(SaveOptions.AcceptAllChangesAfterSave)
```

Listing 1.6 Entity SQL und die generische `CreateQuery()`-Methode im Einsatz (VB.NET)

Wie leicht zu erkennen ist, führt der Einsatz von Entity SQL (selbst mit der `CreateQuery()`-Methode) schnell zu einer nicht ganz unerheblichen Menge Quellcode. Auch können mit der aktuellen Version lediglich Daten gelesen, aber in keinerlei Weise modifiziert werden.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Dies führt zu Recht zu der Frage: Warum Entity SQL einsetzen, wenn LINQ to Entities doch so einfach und elegant erscheint!? Die Antwort liegt im Detail: Nicht alle Abfragen lassen sich einfach mit LINQ abbilden. Ist z.B. erst zur Laufzeit bekannt, welche Entitäten und welche Eigenschaften bei der Abfrage berücksichtigt werden sollen, ist LINQ mit vertretbarem Aufwand schnell am Ende. Entity SQL-Abfragen sind nur Zeichenketten – mit allen Vor- und Nachteilen. Sie können zur Laufzeit erstellt werden, was sicherlich in manchen Situationen ein Vorteil ist. Leider hat somit der Compiler keinerlei Chance, syntaktisch ungültige Abfragen im Vorfeld zu erkennen.

HINWEIS Microsoft hat zudem angekündigt, bei zukünftigen Versionen der SQL Server Reporting Services das ADO.NET Entity Framework zu verwenden und Abfragen so in Entity SQL zu verarbeiten. Daher sollten Sie den SQL-Dialekt zumindest nicht komplett vergessen, auch wenn er in aktuellen Projekten nicht zum Einsatz kommt.

Um dem geneigten Leser einen Eindruck von der Entity SQL-Abfragesprache zu bieten, folgen zwei Beispiele.

```
(SELECT '<Leer>', 1 FROM {0})
UNION
(SELECT VALUE K.Name FROM TonisTortenTraumEntities.Kunden AS K
    WHERE K.PLZ = '50441')
UNION
(SELECT VALUE K.Name FROM TonisTortenTraumEntities.Kunden AS K
    WHERE K.PLZ = '60331');
```

Listing 1.7 Entity SQL-Beispiel

```
SELECT VALUE K.Name FROM TonisTortenTraumEntities.Kunden AS K WHERE
(
    (SELECT B FROM TonisTortenTraumEntities.Bestellungspositionen AS B
     WHERE B.Bestellungen.KundenID = K.ID AND B.Stueckzahl > 10)
    OVERLAPS
    (SELECT B FROM TonisTortenTraumEntities.Bestellungspositionen AS B
     WHERE B.Bestellungen.KundenID = K.ID AND B.Stueckzahl < 20)
);
```

Listing 1.8 Entity SQL-Beispiel

Kapitel 5 beschäftigt sich ausgiebig mit der Entity SQL-Sprache, sodass von hier aus darauf verwiesen sei.

Alle Abfragen, egal ob LINQ to Entities oder Entity SQL, werden von dem Datenanbieter in die entsprechende Abfragesprache der verwendeten Datenbank umgesetzt. Ist dies nicht möglich, was durchaus passieren kann, so kommt es zur Laufzeit zu einem Fehler.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Architektur des ADO.NET Entity Framework

Da ein Bild bekanntlich mehr sagt als tausend Worte, an dieser Stelle eine Abbildung der Architektur des ADO.NET Entity Framework.

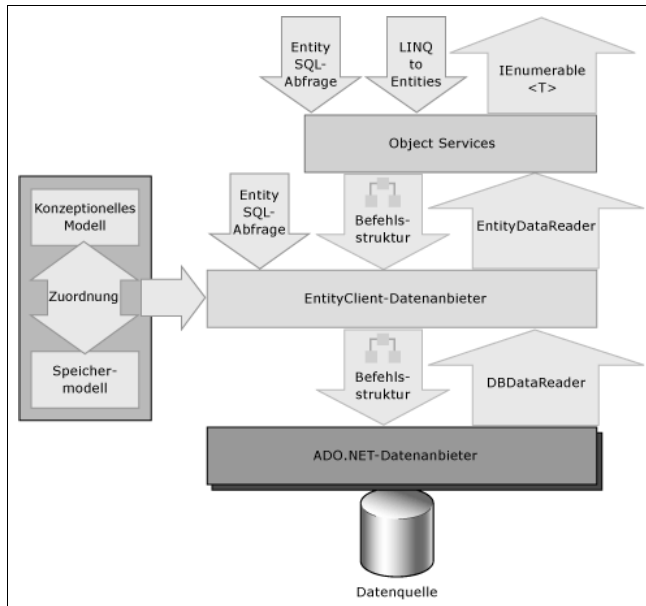

Abbildung 1.1 Die Architektur des ADO.NET Entity Framework 4.0 (Quelle: MSDN)

Am oberen Rand sind gut die Zugriffsmöglichkeiten für Anwendungen zu erkennen: Entity SQL, LINQ to Entities oder einfache Auflistungen via `IEnumerable<T>`/ `IEnumerable(Of T)`. Dazwischen der EntityClient-Datenanbieter, der über die Definitionen im konzeptionellen und physikalischen (Speicher-) Modell den Zugriff über ADO.NET-Datenanbieter auf die Datenquellen durchführt.

Das Entitätenmodell

Der Dreh- und Angelpunkt des ADO.NET Entity Framework ist das Entitätenmodell. Es hält alle beteiligten Bestandteile zusammen und dient damit sozusagen als Container. Es ist sowohl der Rahmen für alle Entitäten inklusive ihrer skalaren und komplexen Werte als auch der Beziehungen und Ableitungen der Entitäten untereinander.

In den meisten Fällen wird dieses Modell wohl durch den grafischen Designer, der mit Visual Studio mitgeliefert wird, bearbeitet werden, doch es gibt auch einen Ansatz, dieses auf reiner Codebasis zu tun.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

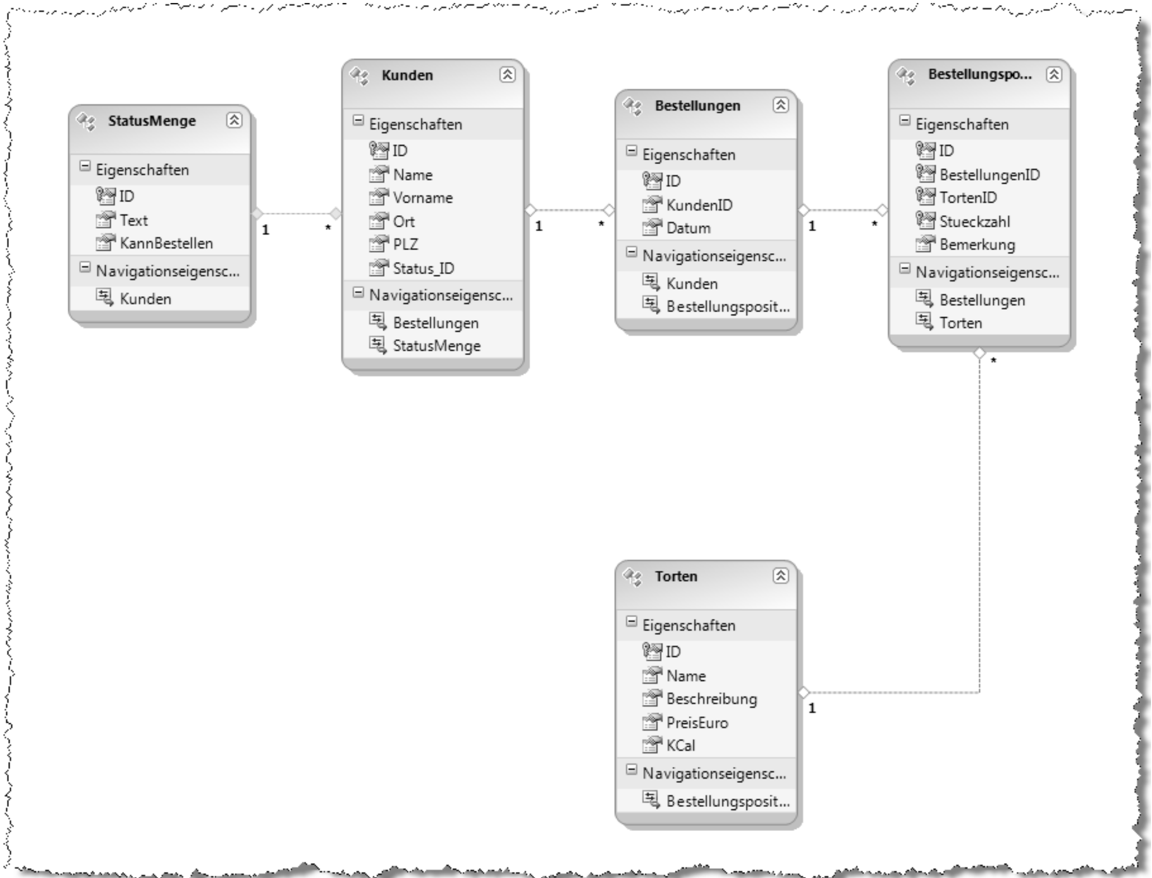

Abbildung 1.2 Ein Entitätenmodell im grafischen Designer von Visual Studio 2010

Mittels einer T4-Vorlage und per Generator wird dieses Modell in entsprechende Klassen mit entsprechenden Eigenschaften umgesetzt. Wie üblich geschieht dies in der Code-Behind-Datei, die zwar geöffnet, jedoch nicht verändert werden sollte!

T4-Vorlagen, die auch schon in Visual Studio 2008 zur Verfügung standen, stellen dabei Code dar, der wiederum anderen Code (erwähnte Code-Behind-Datei) erstellt. Der Gedanke dabei ist ähnlich dem klassischen ASP (nicht ASP.NET!), das Code mit HTML-Fragmenten mischt und schlussendlich eine vollständige HTML-Seite mit dynamischen Bestandteilen erzeugt.

Modellbrowser

Der Modellbrowser, der standardmäßig an der rechten Seite angezeigt wird, stellt das Entitätenmodell, sowohl das konzeptionelle als auch das physikalische Modell dar.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

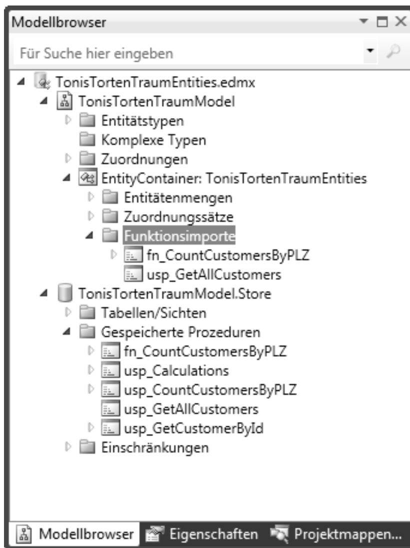

Abbildung 1.3 Der Modellbrowser zeigt sowohl das konzeptionelle als auch das physikalische Modell an

Über das Kontextmenü des Entitätenmodells kann der Modelbrowser, wenn nicht geöffnet oder sichtbar, zum Vorschein gebracht werden.

Der Modellbrowser stellt als Baumstruktur übersichtlich das konzeptionelle und das physikalische Modell dar, aus dem sich das Entitätenmodell zusammensetzt. Was beide (Teil-)Modelle speichern und wozu sie gut sind, wurde bereits weiter vorn in diesem Kapitel erläutert.

Wichtige Bestandteile des EDM

Und damit ist es soweit, die wichtigsten Elemente eines Entitätenmodells (EDM = Entity Data Model) zu beleuchten. Das EDM wird als Datei gespeichert, welche intern einen XML-Aufbau besitzt und die Dateierweiterung *.EDMX* besitzt. Auch hier gilt: Ein direktes Öffnen ist möglich, ein direktes Ändern am grafischen Designer vorbei sollte tunlichst vermieden und notfalls nur in Anwesenheit eines aktuellen Dateibackups durchgeführt werden.

HINWEIS Die *.EDMX*-Datei enthält standardmäßig die gesamten Metadaten, welche im gleichnamigen Abschnitt in diesem Kapitel vorgestellt werden.

Objektkontext

Das wichtigste Objekt im Entity Framework ist, neben den Entitäten selbst, der Objektkontext. Er hat einige wichtige Aufgaben inne, die im Folgenden erläutert werden:

- Er verwaltet die Verbindung zur Persistenzschicht und sorgt somit dafür, dass über die entsprechende Verbindungszeichenfolge auf die Datenbank zugegriffen werden kann
- Durch den Objektkontext werden alle Informationen des EDM (konzeptionelles und physikalisches Modell) verwaltet und intern zum richtigen Zeitpunkt bereitgestellt

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Der Objektkontext überwacht all Änderungen, die an den Daten der Entitäten vorgenommen werden und stellt diese beim Speichern in der Persistenzschicht zur Verfügung. Für Selftracking Entities (siehe Kapitel 8) gilt dies nicht.

Auf Grund der zentralen Bedeutung des Objektkontextes ist seine Instanziierung zwingend notwendig, bevor eine Abfrage, sei es per LINQ oder Entity SQL, durchgeführt oder Änderungen in die Datenbank geschrieben werden können. Die Instanziierung kann wahlweise per Konstruktor ohne Parameter oder per Konstruktor, der eine entsprechende Verbindungszeichenfolge akzeptiert, geschehen. Eine dritte Variante ist die Übergabe einer gültigen Verbindung in Form einer `System.Data.EntityClient.EntityConnection`-Instanz (siehe Kapitel 6).

Eine Verbindungszeichenfolge für das Entity Framework in der Konfiguration (*App.Config*) sieht beispielsweise so aus:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="TonisTortenTraumEntities"
connectionString="metadata=res://*/TonisTortenTraumEntities.csdl|res://*/TonisTortenTraumEntities.ssdl|r
es://*/TonisTortenTraumEntities.msl;provider=System.Data.SqlClient;provider connection string=&quot;Data
Source=.\sqlserver2008r2;Initial Catalog=TonisTortenTraum;Integrated
Security=True;MultipleActiveResultSets=True&quot;" providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

Listing 1.9 Verbindungszeichenfolge für das Entity Framework

Der Name der Verbindungszeichenfolge hier lautet *TonisTortenTraumEntities*. Damit wird die Instanziierung zu einem eleganten Einzeiler.

```
TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities();
```

bzw.

```
Dim TonisTortenTraumContext As New TonisTortenTraumEntities()
```

Da eine der Aufgaben des Objektkontexts darin besteht, Änderungen an den Entitäten zu überwachen, muss er in vielen Anwendungsfällen erzeugt und über einen längeren Zeitraum verwendet werden – der Objektkontext hat also, gemessen an der Gesamtlebensdauer der Anwendung, eine recht hohe Lebenserwartung. Es ist somit zu erwarten, dass er oftmals beim Start erzeugt und erst bei Beendigung der Anwendung zerstört wird. Nichtsdestotrotz implementiert der Objektkontext durch seine Basisklasse die `System.IDisposable`-Schnittstelle, sodass bei kurzfristigen Anwendungsszenarien der Einsatz von `using`-Blöcken möglich und auch ratsam ist. Auf diese Weise werden Ressourcen wie Datenbankverbindungen so schnell wie möglich freigegeben.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Entitäten

Entitäten sind das zentrale und namensgebende Element in einem Entitätenmodell. Der Leser, der sich immer noch ein wenig unklar ist, was eine Entität eigentlich ist, kann sich eine Entität als Zeile in einer entsprechenden Tabelle vorstellen. Eine Tabelle hat dabei typischerweise Spalten und eine Entität entsprechend Eigenschaften. Welche Eigenschaft dabei welcher Spalte in welcher Tabelle zugeordnet ist, steht nicht fest und kann durch das sogenannte Mapping (siehe den Abschnitt »Mappings« in diesem Kapitel) festgestellt werden.

Des Weiteren entspricht den Zeilen einer Tabelle das Objektset, das über eine entsprechende Eigenschaft des Objektkontexts (`Object Set Name`-Eigenschaft) verfügbar ist.

Bevor jedoch auf die einzelnen Eigenschaften näher eingegangen wird, sei im anschließenden Abschnitt auf eine einfache Möglichkeit hingewiesen, die automatisch generierten Klassen durch eigenen Code zu erweitern.

Partielle Entitätenklassen

Alle Entitätenklassen (und im Übrigen auch die Klassen des Objektkontextes) sind als partiell (`partial`/`Partial`) gekennzeichnet, damit der eigenen Erweiterung in separaten Codedateien ohne größeren Aufwand nichts im Wege steht. Aus naheliegenden Gründen sollten Sie selbstverständlich keine direkten Änderungen an der automatisch generierten Codedatei vornehmen, da diese nach dem nächsten automatischen Generierungsprozess eh verloren wären.

Das folgende Beispiel zeigt die Definitionen der Klasse für die Entität *Kunden*

```
[EdmEntityTypeAttribute(NamespaceName="TonisTortenTraumModel", Name="Kunden")]
[Serializable()]
[DataContractAttribute(IsReference=true)]
public partial class Kunden : EntityObject
{
    ...
}
```

Listing 1.10 Alle generierten Entitätenklassen sind als partielle Klassen angelegt (C#)

Und als VB.NET-Quellcode:

```
<EdmEntityTypeAttribute(NamespaceName:="TonisTortenTraumModel", Name:="Kunden")>
<Serializable()>
<DataContractAttribute(IsReference:=True)>
Public Partial Class Kunden
    Inherits Personen
    ...
End Class
```

Listing 1.11 Alle generierten Entitätenklassen sind als partielle Klassen angelegt (VB.NET)

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Alle partiell gekennzeichneten Klassen lassen sich wie gesagt in eigenen Codedateien erweitern, so als würde direkt der Code der bestehenden Klasse erweitert werden. Dabei kann die Klasse nur durch weitere Mitglieder erweitert werden – ein Entfernen oder Überlagern in andere Codedateien der definierten Mitglieder ist nicht möglich.

Dabei kann die Klasse in jeder ihrer Codedateien um beliebige Methoden, Eigenschaften, Attribute, etc. erweitert werden. Lediglich der Zugriffslevel (`Private`, `Public`, `Protected`) und Zusätze wie `abstract/MustInherit` oder `sealed/NotInheritable` dürfen sich nicht widersprechen, wenn sie angegeben werden.

HINWEIS Aus genau diesem Grund ist es ratsam, weder das eine noch das andere anzugeben, gemäß dem Motto »Wer nichts macht, macht nichts falsch«. Der automatisch generierte Teil der Klasse verfügt über den richtigen Zugriffslevel und alle notwendigen Auszeichnungen durch Attribute.

Die folgenden Punkte sind bei der Verwendung von partiellen Klassen zu berücksichtigen:

- Die Aufteilung der partiellen Klasse auf mehrere Dateien ist lediglich eine physikalische Trennung. Das absolut gleiche Ergebnis ließe sich erzielen, wenn der gesamte Code sich in einer einzelnen Datei befände.
- Durch die Trennung wird lediglich sichergestellt, dass das automatische Erzeugen von Code die eigene Programmierung unangetastet lässt
- Da der Code zur gleichen Klasse gehört, ist ein uneingeschränkter Zugriff auf private Mitglieder möglich
- Alle Teile der Klasse müssen im gleichen Namensraum definiert werden. Ist das nicht der Fall, handelt es sich um zwei unterschiedliche Klassen. Diese tragen dann zwar den gleichen Kurznamen, der vollständige Name ist dann jedoch völlig unterschiedlich (z.B. `NameSpaceA.MeineKlasse` und `NameSpaceB.MeineKlasse`). Ändert sich der Namensraum eines Teiles der Klasse, so gibt der Compiler nicht zwangläufig einen Fehler oder eine Warnung aus. Nur wenn dadurch Mitglieder plötzlich nicht mehr vorhanden sind, kommt es zu einer entsprechenden Ausgabe!

Mit diesem Wissen ausgestattet ist es ein Leichtes, die Klasse in einer eigenen, getrennten Datei zu erweitern.

Das folgende Beispiel implementiert eine Methode, welche die Klasse serialisiert als SOAP auf die Festplatte speichert. Beachten Sie, dass der eigene Teil der partiellen Klasse nicht mit dem `Serializable`-Attribut ausgezeichnet werden darf und dies auch nicht geschehen muss, da dieses Attribut bereits von der automatischen Codegenerierung angebracht wurde.

```
// Referenz auf System.Runtime.Serialization.Formatters.Soap notwendig!
using System.Runtime.Serialization.Formatters.Soap;

namespace TonisTortenTraum
{
    partial class Kunden
    {
        public void WriteToDisk(string filename)
        {
            // Parameter prüfen
            if (string.IsNullOrWhiteSpace(filename))
                throw new ArgumentException("filename");

            //  SOAP-Formatter erzeugen
            SoapFormatter sf = new SoapFormatter();
```

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

```
            //  Ausgabe in Datei
            using (System.IO.StreamWriter sw = new System.IO.StreamWriter(filename))
                sf.Serialize(sw.BaseStream, this);
        }
    }
}
```

Listing 1.12 Die generierte Klasse durch `WriteToDisk` (Datei: *Kunden_WriteToDisk.cs*) erweitert (C#)

Und als VB.NET-Quellcode:

```
Option Strict On

' Referenz auf System.Runtime.Serialization.Formatters.Soap notwendig!
Imports System.Runtime.Serialization.Formatters.Soap

Partial Class Kunden

    Public Sub WriteToDisk(ByVal filename As String)
        ' Parameter prüfen
        If String.IsNullOrWhiteSpace(filename) Then
            Throw New ArgumentException("filename")
        End If

        ' SOAP-Formatter erzeugen
        Dim sf As New SoapFormatter

        ' Ausgabe in Datei
        Using sw As New IO.StreamWriter(filename)
            sf.Serialize(sw.BaseStream, Me)
        End Using
    End Sub
End Class
```

Listing 1.13 Die generierte Klasse durch `WriteToDisk` (Datei: *Kunden_WriteToDisk.vb*) erweitert (VB.NET)

Der Aufruf gestaltet sich denkbar einfach durch die Notation der eigenen Methode.

```
// Objektkontext erstellen
TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities();

// Testkunden auswählen
Kunden Kunden = TonisTortenTraumContext.Kunden.FirstOrDefault();

// Und als SOAP auf die Festplatte schreiben
Kunden.WriteToDisk(@"d:\temp\Kunde1.xml");
```

Listing 1.14 Der Aufruf der eigenen Methode (C#)

Und als VB.NET-Quellcode:

```
' Objektkontext erstellen
Dim TonisTortenTraumContext As New TonisTortenTraumEntities()

' Testkunden auswählen
Dim kunden As Kunden = TonisTortenTraumContext.Kunden.FirstOrDefault()

' Und als SOAP auf die Festplatte schreiben
kunden.WriteToDisk("d:\temp\Kunde1.xml")
```

Listing 1.15 Der Aufruf der eigenen Methode (VB.NET)

Eigenschaften einer Entität

Jede Entität besitzt eine ganze Reihe von Eigenschaften. Und damit sind an dieser Stelle nicht die Eigenschaften gemeint, welche Daten speichern (also quasi die Spalten), sondern die, die bei der Generierung der Klassen berücksichtigt werden. Mit ihnen lässt sich so z.B. der Name, der Basistyp oder die Information festlegen, ob die Klasse eine reine Basisklasse darstellt (also `abstract/MustInherit`).

Abstract

Diese Eigenschaft bestimmt, ob es sich bei einer Klasse dieser Entität um eine reine Basisklasse, also eine Klasse, welche mit `abstract/MustInherit` ausgezeichnet ist, handelt. Von solchen Klassen kann keine Instanz erzeugt werden, lediglich können andere Klassen von diesen Klassen wiederum ableiten.

Dabei ist es unerheblich, ob es sich bei einer ableitenden Klasse um eine Klasse für eine andere Entität oder um eine »gewöhnliche« Klasse handelt. Obwohl der erste Fall sicherlich der häufigste sein dürfte.

Der folgende kleine Code-Ausschnitt zeigt den Aufbau solcher abstrakter Klassen für Entitäten.

```
[EdmEntityTypeAttribute(NamespaceName="TonisTortenTraumModel", Name="Person")]
[Serializable()]
[DataContractAttribute(IsReference=true)]
public abstract partial class Person : EntityObject
{
    ...
}
```

Listing 1.16 Ausschnitt einer als abstract generierten Klasse (C#)

Und als VB.NET-Quellcode:

```
<EdmEntityTypeAttribute(NamespaceName:="TonisTortenTraumModel", Name:="Personen")>
<Serializable()>
<DataContractAttribute(IsReference:=True)>
Public MustInherit Partial Class Personen
    Inherits EntityObject
    ...
End Class
```

Listing 1.17 Ausschnitt einer als `MustInherit` generierten Klasse (C#)

Der Standardwert ist `false/False`, sodass von jeder Klasse im Code Instanzen verwendet werden können.

Zugriff (Access)

Hiermit lässt sich der Zugriffslevel der Klasse bestimmen. Damit wird also festgelegt, von *wo* aus die Klasse sichtbar ist. Mögliche Werte sind hierfür:

- `Internal/Friend`
- `Private`
- `Protected`
- `Public`

Der Wert `Public` ist der Standard, womit der größte mögliche lesende Zugriff für die Entitäten-Eigenschaft geleistet wird. Über die `Setter`-Eigenschaft kann Gleiches für den schreibenden Zugriff gesteuert werden.

Basistyp (Base Type)

Gibt den Basistyp der Entitäten-Klasse an. Dabei muss es sich um eine Entitäten-Klasse auf dem gleichen Entitätenmodell handeln. Andere Klassen sind hierfür nicht zulässig. Der Standardwert ist lediglich `(None)`, sodass die generierte Klasse direkt von der `System.Data.Objects.DataClasses.EntityObject`-Klasse ableitet.

Im grafischen Designer werden Klassen und deren Basisklassen (also Vererbungen) auf zweierlei Weise dargestellt: In beiden Fällen zeigt die Spitze eines Pfeil auf den Namen der Basisklasse – entweder direkt unter dem Namen der Entität oder als Vererbungspfeil auf die Entität.

Abbildung 1.4 Basisklassen und Vererbung im grafischen Designer

Weitere Details finden Sie im Abschnitt »Beziehungen« in diesem Kapitel.

Dokumentation (Documentation)

Diese Eigenschaft gestattet die Angabe einer erklärenden Beschreibung in Langform (`Long Description`) und eine kurze Zusammenfassung (`Summary`).

Entitätsname (Entity Set Name)

Durch die `Entity Set Name`-Eigenschaft wird bestimmt, wie die Eigenschaft des Objektkontextes heißt, über welche diese Entität aufgelistet, einzelne Entitäten gelöscht oder neue hinzugefügt werden können. Diese Eigenschaft verfügt nur über einen Getter, bzw. ist als `ReadOnly` gekennzeichnet, sodass zwar die genannten Änderungen erlaubt sind, jedoch kein anderes Objektset zugewiesen werden kann (oder `null/Nothing`).

Der Standardname, der für diese Eigenschaft verwendet wird, entspricht dem Namen der Entität plus dem Zusatz `Set`.

HINWEIS Wird das Entitätenmodell anhand einer Datenbank per Assistent erzeugt, so verwendet dieser für die `Entity Set Name`-Eigenschaft den gleichen Namen, den auch die Entität selbst erhält. Sicherlich zwar nicht dramatisch, aber dennoch bei genauerer Überlegung ein wenig befremdlich.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Im Folgenden werden die erzeugten Eigenschaften des Objektkontexts gezeigt, welche das Objektset zugreifbar machen.

```
public ObjectSet<Personen> PersonenSet
{
    get
    {
        if ((_PersonenSet == null))
        {
            _PersonenSet = base.CreateObjectSet<Personen>("PersonenSet");
        }
        return _PersonenSet;
    }
}

private ObjectSet<Personen> _PersonenSet;
```

Listing 1.18 Die Eigenschaft des Datenkontextes, welcher das Objektset für *Personen* verfügbar macht (C#)

Und als VB.NET-Quellecode:

```
Public ReadOnly Property PersonenSet() As ObjectSet(Of Personen)
    Get
        If (_PersonenSet Is Nothing) Then
            _PersonenSet = MyBase.CreateObjectSet(Of Personen)("PersonenSet")
        End If
        Return _PersonenSet
    End Get
End Property

Private _PersonenSet As ObjectSet(Of Personen)
```

Listing 1.19 Die Eigenschaft des Datenkontextes, welcher das Objektset für *Personen* verfügbar macht (VB.NET)

Der Einsatz in einer LINQ to Entities-Abfrage sieht dann dementsprechend so aus – die Eigenschaften können als Quelle verwendet werden.

```
var Personen = from p in TonisTortenTraumContext.PersonenSet orderby p.Name select p;
```

Listing 1.20 Das Objektset `PersonenSet` in einer LINQ to Entities-Abfrage (C#)

Und als VB.NET-Quelltext:

```
Dim Personen = From p In TonisTortenTraumContext.PersonenSet Order By p.Name
```

Listing 1.21 Das Objektset `PersonenSet` in einer LINQ to Entities-Abfrage (VB.NET)

Name

Mit dieser Eigenschaft wird der Name der Entität und damit die der entsprechenden Klasse festgelegt. Dieser Name muss den Regeln eines Bezeichners für C#/VB.NET genügen. Außerdem darf sich der Name nicht nur durch Groß-/Kleinschreibung von dem einer anderen Eigenschaft der Entität unterscheiden.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Im grafischen Designer wird der Name im oberen Bereich der Entität dargestellt, sodass eine schnelle Identifikation gewährleistet ist.

Abbildung 1.5 Der Name der Entität wird im oberen Bereich angezeigt

HINWEIS Sie können den Namen auch direkt im grafischen Designer ändern, indem Sie auf den Namen klicken, sodass eine kleine Textbox an der Stelle eingeblendet wird, die Ihre Eingaben entgegen nimmt.

Eine Eigenschaft kann entweder eine skalare Eigenschaft, eine komplexe Eigenschaft oder eine Navigationseigenschaft sein. Im Folgenden werden alle drei Gruppen kurz beschrieben.

Skalare Eigenschaft

Skalare Eigenschaften stellen einzelne Werte einer Entität dar und bilden in der Analogie zu Datenbanken Tabellenspalten. Jede Entität hat mindestens eine Eigenschaft, meist jedoch erheblich mehr. Vergleicht man die Entität mit einer Tabelle, so sind die Eigenschaften deren Spalten. Und so wie es sich für eine Eigenschaft/Spalte gehört, können für diese Name, Datentyp, Genauigkeit, maximale Größe, etc. festgelegt werden.

Die folgende Abbildung zeigt eine solche Entität, die über fünf solcher Eigenschaften (ID bis PLZ) verfügt.

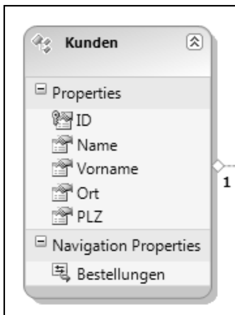

Abbildung 1.6 Eine Entität mit fünf skalaren Eigenschaften

HINWEIS Beachten Sie bitte, dass einige Eigenschaften auch nur dann Sinn machen und verfügbar sind, wenn der entsprechende Datentyp (Type) ausgewählt wurde.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Parallelitätsmodus

Beim Speichern von Änderungen kann es immer zu Situationen kommen, in denen Änderungskonflikte – auch Parallelitätsverletzungen (Concurrency conflicts) genannt – auftreten. Dies ist immer dann der Fall, wenn vor dem Speichern die Daten im Speicher des Clients in der Datenbank verändert wurden, sodass das Speichern diese Änderungen überschreiben würde.

Der Parallelitätsmodus beschreibt, wie (oder besser ob) ein solcher Konflikt erkannt werden soll. Dabei kann diese Einstellung für jede Eigenschaft einer jeden Entität festgelegt werden.

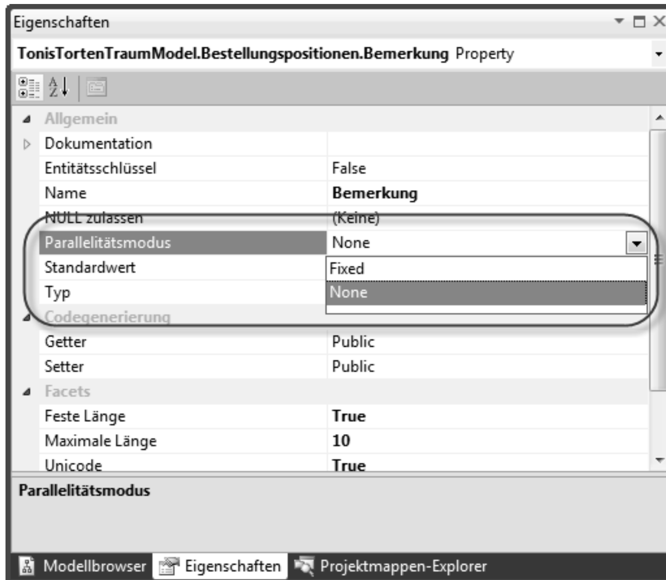

Abbildung 1.7 Der Parallelitätsmodus wird für jede einzelne Eigenschaft einer jeden Entität festgelegt

Möglich sind dafür die folgenden Werte, die auch als Aufzählung mit dem Namen `System.Data.Metadata.EdmConcurrencyMode` zur Verfügung stehen.

Mode/ Wert	Beschreibung
None	Diese Eigenschaft wird nie während des Schreibvorgangs überprüft. Dies ist der Standardparallelitätsmodus.
`Fixed`	Diese Eigenschaft wird immer während des Schreibvorgangs überprüft

Tabelle 1.1 Mögliche Werte für den Parallelitätsmodus (`EdmConcurrenyMode`)

Tritt beim Aufruf der `SaveChanges()`-Methode des Objektkontextes, welche die Änderungen dauerhaft speichert, also ein solcher Konflikt auf, so wird dies durch eine `OptimisticConcurrencyException`-Ausnahme signalisiert. Diese kann abgefangen werden und es kann entschieden werden, ob der Client (also die eigenen Änderungen) oder die Datenbank (also die Änderungen, die von anderer Seite gemacht wurden) dabei gewinnt.

Zu diesem Zweck wird die `Refresh()`-Methode aufgerufen, deren erster Parameter einen Wert aus der `System.Data.Objects.RefreshMode`-Aufzählung darstellt, die dafür genau zwei Werte zulässt.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Wert	Beschreibung
`StoreWins`	Die Änderungen aus der Persistenzschicht werden übernommen und im lokalen Speicher überschrieben – die Änderungen von anderer Seite gewinnen also
`ClientWins`	Die eigenen Änderungen werden gespeichert und gewinnen somit. Für andere Clients kann dies wiederum zu eigenen Konflikten führen.

Tabelle 1.2 Die `RefreshMode`-Aufzählung legt fest, wessen Änderungen erhalten bleiben

Das folgende Codebeispiel zeigt, wie das generelle Vorgehen konkret aussieht. Es zeigt, wie Änderungen an Kundendaten durchgeführt werden.

```
// Objektkontext erstellen
using (TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities())
{

    // Kunde auswählen...
    Kunden Kunde = TonisTortenTraumContext.Kunden
        .Where(k => k.Name ==
            "Müller AG").FirstOrDefault();
    if (Kunde != null)
    {
        // ...und Änderungen an Daten durchführen, wenn Kunde gefunden wurde
        Kunde.Name = "Müller GmbH";
        Kunde.Ort = "Frankfurt";
    }

    try
    {
        // Zunächst der Versuch, die Änderungen zu speichern

        TonisTortenTraumContext.SaveChanges();
    }
    catch (OptimisticConcurrencyException)
    {
        // Die Parallelitätsverletzungen auflösen
        TonisTortenTraumContext.Refresh(RefreshMode.ClientWins, Kunde);

        // Nun können die Daten gespeichert werden. Dabei kann es theoretisch
        // wieder zu einer OptimisticConcurrencyException-Ausnahme kommen.
        // Diese führt dann zum Abbruch des Speicherns
        TonisTortenTraumContext.SaveChanges();
    }
}
```

Listing 1.22 Das Schreiben von Änderungen (C#)

Und als VB.NET-Quellcode:

```
' Objektkontext erstellen
Using TonisTortenTraumContext As New TonisTortenTraumEntities()
    ' Kunde auswählen...
```

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

```
    Dim Kunde As Kunden = TonisTortenTraumContext.Kunden.Where(Function(k) k.Name = "Müller
AG").FirstOrDefault()

    If Kunde IsNot Nothing Then
        ' ...und Änderungen an Daten durchführen, wenn Kunde gefunden wurde
        Kunde.Name = "Müller GmbH"
        Kunde.Ort = "Frankfurt"
    End If

    Try
        ' Zunächst der Versuch, die Änderungen zu speichern
        TonisTortenTraumContext.SaveChanges()
    Catch ex As OptimisticConcurrencyException
        ' Die Parallelitätsverletzungen auflösen
        TonisTortenTraumContext.Refresh(RefreshMode.ClientWins, Kunde)

        ' Nun können die Daten gespeichert werden. Dabei kann es theoretisch
        ' wieder zu einer OptimisticConcurrencyException-Ausnahme kommen.
        ' Diese führt dann zum Abbruch des Speicherns
        TonisTortenTraumContext.SaveChanges()
        End Try
End Using
```

Listing 1.23 Das Schreiben von Änderungen (VB.NET)

Durch dieses Vorgehen kann auch für unterschiedliche Entitäten von Fall zu Fall `ClientsWins` oder `StoreWins` angegeben werden. Wichtig ist nur, dass alle Konflikte beseitigt sind, bevor ein erneuter Versuch unternommen wird, um die Änderungen zu speichern.
Ist nicht bekannt, um welche Entität es sich handelt, so kann diese Information aus der `OptimisticConcurrencyException`-Ausnahme gewonnen werden. Zu diesem Zweck steht die `StateEntries`-Auflistung zur Verfügung. Diese enthält alle Entitäten und deren Zustand, die in einem Konflikt stehen.

```
...
    try
    {
        // Zunächst der Versuch, die Änderungen zu speichern

        TonisTortenTraumContext.SaveChanges();
    }
    catch (OptimisticConcurrencyException ex)
    {
        // Die Parallelitätsverletzungen auflösen
       foreach (ObjectStateEntry entry in ex.StateEntries)
           TonisTortenTraumContext.Refresh(RefreshMode.ClientWins, entry.Entity);

        // Nun können die Daten gespeichert werden. Dabei kann es theoretisch
        // wieder zu einer OptimisticConcurrencyException-Ausnahme kommen.
        // Diese führt dann zum Abbruch des Speicherns
        TonisTortenTraumContext.SaveChanges();
    }
...
```

Listing 1.24 Änderungskonflikte flexibel lösen (C#)

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Und als VB.NET-Quellcode:

```
...
    Try
        ' Zunächst der Versuch, die Änderungen zu speichern
        TonisTortenTraumContext.SaveChanges()
    Catch ex As OptimisticConcurrencyException
        ' Die Parallelitätsverletzungen auflösen, indem die SateEntires-Auflistung durchlaufen wird
        For Each entry as ObjectStateEntry In ex.StateEntries
            TonisTortenTraumContext.Refresh(RefreshMode.ClientWins, entry.Entity)
        Next

        ' Nun können die Daten gespeichert werden. Dabei kann es theoretisch
        ' wieder zu einer OptimisticConcurrencyException-Ausnahme kommen.
        ' Diese führt dann zum Abbruch des Speicherns
        TonisTortenTraumContext.SaveChanges()
        End Try
...
```

Listing 1.25 Änderungskonflikte flexibel lösen (VB.NET)

Standardwert (Default Value)

Bestimmt den Standardwert für die Entitäten-Eigenschaft bei neu erstellten Entitäten. Der Wert muss zum Typ der Eigenschaft passen, darf aber auch leer sein. In einem solchen Fall muss `(None)` ausgewählt werden. Letzteres ist auch der Standardwert.[2]

Dokumentation (Documentation)

Diese Eigenschaft gestattet die Angabe einer erklärenden Beschreibung in Langform (`Long Description`) und eine kurze Zusammenfassung (`Summary`).

Entitätsschlüssel (Entity Key)

Mit dieser Eigenschaft kann bestimmt werden, ob diese Entitäten-Eigenschaft Teil des Schlüssels ist, mit dem auf eine bestimmte Entität zugegriffen werden kann. Jede Entität muss mindestens eine solche Eigenschaft besitzen, es können aber durchaus auch mehrere sein. In einem solchen Fall handelt es sich um einen zusammengesetzten Schlüssel.

Im Datenbank-Jargon wird ein solcher Schlüssel als Primärschlüssel bezeichnet.

Feste Länge (Fixed Length)

Legt fest, die ob Entitäten-Eigenschaft von einer festen Länge ist oder nicht. Ist eine feste Länge gewünscht, so gibt die `Max Length`–Eigenschaft diese an.

---

[2] Achtung, nicht durcheinander kommen. Gemeint ist der Standardwert der Eigenschaft, welche den Standardwert der Entitäten-Eigenschaft bestimmt!

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Getter

Hiermit lässt sich der Zugriffslevel des Getters der Entitäten-Eigenschaft bestimmen. Damit wird also festgelegt, von *wo* aus die Eigenschaft gelesen werden kann. Mögliche Werte sind hierfür:

- `Internal/Friend`
- `Private`
- `Protected`
- `Public`

Der Wert `Public` ist der Standard, mit dem der größte mögliche lesende Zugriff für die Entitäten-Eigenschaft gewährleistet wird. Über die `Setter`-Eigenschaft kann Gleiches für den schreibenden Zugriff gesteuert werden.

Maximale Länge (Max Length)

Gibt die maximal erlaubte Länge des Inhalts an. Erlaubt sind die folgenden Werte:

- **`(None)`** Keine Begrenzung schränkt die Länge ein
- **`Max`** Maximale Länge, die der Datenanbieter bereitstellen kann
- Jeder ganzzahlige Wert größer 0

Der Wert `None` ist der Standard, sodass die maximale Länge für Zeichenketten die maximale Länge darstellt – quasi unbegrenzt also.

Name

Legt den eindeutigen Namen der Eigenschaft innerhalb der Entität fest. Dieser Name muss den Regeln eines Bezeichners für C#/VB.NET genügen. Außerdem darf sich der Name nicht nur durch Groß-/ Kleinschreibung von dem einer anderen Eigenschaft der Entität unterscheiden.

HINWEIS Sie können den Namen auch direkt im grafischen Designer ändern, indem Sie auf den Namen klicken, sodass eine kleine Textbox an der Stelle eingeblendet wird, die Ihre Eingaben entgegen nimmt.

Nullable

Gibt an, ob die Eigenschaft `null` oder `Nothing` zulässt oder nicht.

Präzision (Precision)

Diese Eigenschaft erlaubt die Angabe der Präzision für Datumswerte und numerische Werte mit Nachkommastellen (`Decimal`).

Setter

Hiermit lässt sich der Zugriffslevel des Setters der Entitäten-Eigenschaft bestimmen. Damit wird also festgelegt, von *wo* aus die Eigenschaft geschrieben werden kann. Mögliche Werte sind hierfür:

- `Internal/Friend`
- `Private`

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

- `Protected`
- `Public`

Der Wert `Public` ist hier der Standard. Über die `Getter`-Eigenschaft kann Gleiches für den lesenden Zugriff gesteuert werden.

StoreGeneratedPattern

Legt das Verfahren fest, mit dem der Wert dieser Eigenschaft von dem Datenanbieter erzeugt wird. Möglich sind:

- **`Computed`** Der Wert wird vom Datenanbieter berechnet
- **`Identity`** Der Wert ist ein Identitätswert (Autowert) aus der Datenbank
- **`None`** Der Wert wird nicht vom Datenanbieter geliefert

Der Standardwert, der für die meisten Entitäten-Eigenschaften passend sein dürfte, ist `None`, da die wenigsten Werte von der Datenbank vorgegeben/berechnet werden.

Type

Diese Eigenschaft legt den Datentyp der Entitäten-Eigenschaft fest.

- `Binary`
- `Bool/Boolean`
- `Byte`
- `DateTime`
- `DateTimeOffeset`
- `Decimal`
- `Double`
- `Guid`
- `Int16`
- `Int32/Integer`
- `Int64`
- `SByte`
- `Single`
- `String`
- `Time`

Diese Eigenschaft ist maßgebend für die `Default Value`-Eigenschaft und bestimmt indirekt, welche anderen Eigenschaften für diese Entitäten-Eigenschaft im Designer angezeigt werden. Solche, die für die aktuelle Auswahl keine Bedeutung haben, werden komfortablerweise ausgeblendet. Der Standardwert für diese Eigenschaft ist `String`.

Unicode

Für Zeichenketten kann hier bestimmt werden, ob die Zeichen als Unicode abgelegt werden sollen oder nur als ASCII. Der Standardwert ist `true/True`, was sich mit dem `System.String`-Datentyp des .NET Framework deckt, der ebenfalls mit Unicode arbeitet.

Komplexe Eigenschaften

Nicht alle Eigenschaften bestehen nur aus einer einzelnen skalaren Information, sondern setzen sich aus mehreren Daten zusammen. So besteht eine Ortsangabe in einfacher Form aus einem Land, einer PLZ und dem Namen des Ortes/der Stadt. Die Eigenschaft *Ortsangabe* wiederum kann in unterschiedlichsten Entitäten verwendet werden. Jedoch sind immer alle einzelnen Eigenschaften, hier *Land*, *PLZ* und *Ort*, mit von der Partie. Komplexe Eigenschaften setzen sich also aus mehreren skalaren oder anderen komplexen Eigenschaften zusammen.

Um komplexe Eigenschaften nutzen zu können, müssen diese erst im konzeptionellen Modell definiert werden.

Abbildung 1.8 Komplexe Typen im konzeptionellen Modell

Dies geschieht einfach über das Kontextmenü des Modellbrowsers. Ist dies geschehen, so können die komplexen Eigenschaften wie einfache skalare Eigenschaften einer Entität verwendet werden. Zu diesem Zweck steht im Kontextmenü einer Entität der Befehl *Hinzufügen/Komplexe Eigenschaft* zur Auswahl.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

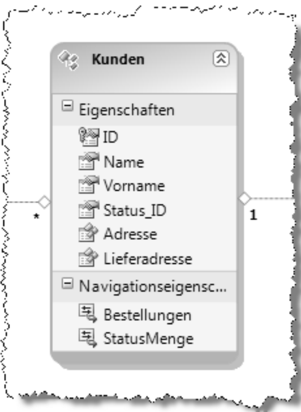

Abbildung 1.9 Die komplexe Eigenschaft zur Verwendung in einer Entität

Komplexe Eigenschaften werden als *Unterobjekte* als Klassen im .NET Code generiert. Allerdings müssen beim Erstellen neuer Instanzen einer Entitätenklasse diese nicht gesondert erstellt werden – dies geschieht automatisch. Folgender Codeausschnitt ist daher kein Problem.

```
// Kunden im Speicher erzeugen
Kunden k = new Kunden();

// Name festlegen
k.Name = "Kansy AG";

// Adresse (komplexe Eigenschaft) festlegen
k.Adresse.Ort = "Nidderau";
k.Adresse.PLZ = "61130";
k.Adresse.Strasse = "Wasserweg";

// Lieferadresse (ebenfalls komplex) festlegen
k.Lieferadresse.Ort = "Frankfurt a. M.";
k.Lieferadresse.PLZ = "60329 ";
k.Lieferadresse.Strasse = "Untermainkai";

// Weiter mit dem Objekt arbeiten
// ...
```

Listing 1.26 Der Zugriff auf komplexe Eigenschaften einer Entität (C#)

Und als VB.NET-Quellcode:

```
' Kunden im Speicher erzeugen
Dim k As New Kunden()

' Name festlegen
k.Name = "Kansy AG"

' Adresse (komplexe Eigenschaft) festlegen
k.Adresse.Ort = "Nidderau"
k.Adresse.PLZ = "61130"
k.Adresse.Strasse = "Wasserweg"
```

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

```
' Lieferadresse (ebenfalls komplex) festlegen
k.Lieferadresse.Ort = "Frankfurt a. M."
k.Lieferadresse.PLZ = "60329 "
k.Lieferadresse.Strasse = "Untermainkai"

' Weiter mit dem Objekt arbeiten
' ...
```

Listing 1.27 Der Zugriff auf komplexe Eigenschaften einer Entität (VB.NET)

In der Datenbank werden dann für die einzelnen skalaren Eigenschaften einer komplexen Eigenschaft wie gewohnt Spalten angelegt – eine Art Normalisierung, d.h. das Auslagern in eine eigene Daten findet nicht statt.

HINWEIS Dies bedeutet auch, dass Entitätenmodelle, die später mit einer solchen Datenbank erzeugt werden, nichts von den komplexen Eigenschaften wissen. In diesen Modellen finden sich nur die einfachen skalaren Eigenschaften wieder. Um Szenarien abzubilden, in denen z.B. alle Adressen in einer Tabelle gesammelt (normalisiert) werden sollen, müssen mehrere Entitäten mit entsprechenden Beziehungen zueinander erzeugt werden.

In der Datenbank sieht dies, hier ein Beispiel für den SQL Server, so aus:

Spaltenname	Datentyp	NULL zula...
ID	int	☐
Name	nvarchar(50)	☐
Vorname	nvarchar(50)	☐
Status_ID	int	☐
Adresse_Ort	nvarchar(MAX)	☐
Adresse_PLZ	nvarchar(MAX)	☐
Adresse_Strasse	nvarchar(MAX)	☐
Lieferadresse_Ort	nvarchar(MAX)	☐
Lieferadresse_PLZ	nvarchar(MAX)	☐
Lieferadresse_Strasse	nvarchar(MAX)	☐
		☐

Abbildung 1.10 Tabelle der Entität mit komplexen Eigenschaften in der Datenbank (SQL Server)

Die teilweise recht großen Datentypen der Spalten (`NVARCHAR(MAX)`) rühren daher, dass für diese Beispiele keine Längenbeschränkungen definiert wurden.

Navigationseigenschaften

Navigationseigenschaften sind, wie der Name schon andeutet, besondere Eigenschaften die es erlauben, von einer Entität zu einer anderen zu »navigieren« – natürlich nicht beliebig, sondern nur so wie es durch Beziehungen im Entitätenmodell festgelegt wurde. So kann z.B. ein Kunde über beliebig viele Bestellungen verfügen, die ihrerseits wiederum über eine beliebige Anzahl von Bestellungspositionen verfügen können.

In diesem Fall kann von einem Kunden zu seinen Bestellungen (eine Auflistung) und von einer speziellen Bestellung zu deren Positionen (ebenfalls eine Auflistung) navigiert werden. Der umgekehrte Weg ist auch möglich: So kann von einer bestimmten Bestellposition zu einer einzelnen Bestellung hin und von dort zu einem einzelnen Kunden navigiert werden.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Navigationseigenschaften werden im grafischen Designer von Visual Studio abgegrenzt im unteren Teil einer Entität dargestellt.

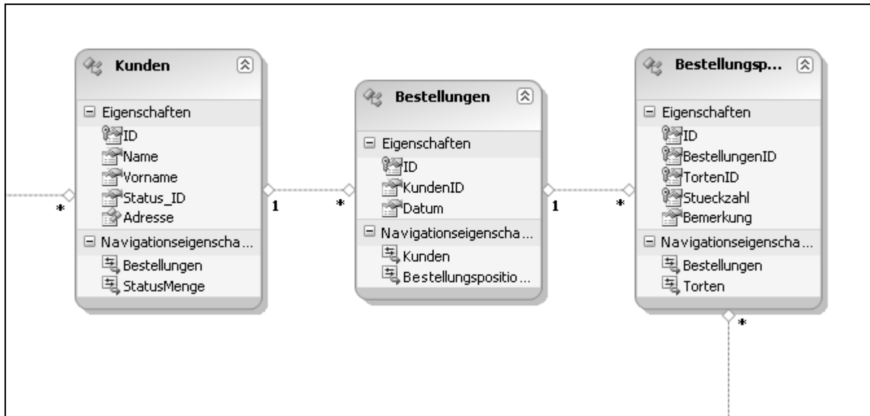

Abbildung 1.11 Die drei Entitäten und deren Navigationseigenschaften

Der Code, um diese Beziehungen zu nutzen, kann wie folgt aussehen. Zunächst in der Richtung vom Kunden zu den Positionen seiner Bestellungen.

```
// Objektkontext erstellen
TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities();

// Einen Kunden ermitteln
Kunden kunde = (from k in TonisTortenTraumContext.Kunden select k).FirstOrDefault();

// Anzeigen, wenn vorhanden
if (kunde != null)
{
    // Kundenname ausgeben
    Debug.Print("Kunde: {0}", kunde.Name);

    // Bestellungen des Kunden ausgeben
    Debug.Print("Kunden hat {0} Bestellungen", kunde.Bestellungen.Count);

    foreach (Bestellungen bestellung in kunde.Bestellungen)
    {
        // ID der Bestellung ausgeben
        Debug.Print("Bestellung: {0}", bestellung.ID);

        // Positionen der aktuellen Bestellung ausgeben
        Debug.Print("Bestellung hat {0} Positionen", bestellung.Bestellungspositionen.Count);

        foreach (Bestellungspositionen position in bestellung.Bestellungspositionen)
            Debug.Print(position.Bemerkung);
    }
}
```

Listing 1.28 Die Navigation vom Kunden hin zur Bestellungsposition ... (C#)

Und als VB.NET-Quellcode:

```
' Objektkontext erstellen
Dim TonisTortenTraumContext As New TonisTortenTraumEntities()

' Einen Kunden ermitteln
Dim kunde As Kunden = (From k In TonisTortenTraumContext.Kunden).FirstOrDefault()

' Anzeigen, wenn vorhanden
If kunde IsNot Nothing Then

    ' Kundenname ausgeben
    Debug.Print("Kunde: {0}", kunde.Name)

    ' Bestellungen des Kunden ausgeben
    Debug.Print("Kunden hat {0} Bestellungen", kunde.Bestellungen.Count)

    For Each bestellung As Bestellungen In kunde.Bestellungen
        Debug.Print("Bestellung: {0}", bestellung.ID)

        ' Positionen der aktuellen Bestellung ausgeben
        Debug.Print("Bestellung hat {0} Positionen", bestellung.Bestellungspositionen.Count)

        For Each position As Bestellungspositionen In bestellung.Bestellungspositionen
            Debug.Print(position.Bemerkung)
        Next
    Next
End If
```

Listing 1.29 Die Navigation vom Kunden hin zur Bestellungsposition ... (VB.NET)

Und nun in der umgekehrten Richtung von einer Bestellungsposition hin zum Kunden.

```
// Objektkontext erstellen
TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities();

// Eine Bestellungsposition ermitteln
Bestellungspositionen position =
    (from b in TonisTortenTraumContext.Bestellungspositionen select b).FirstOrDefault();

// Navigation nur möglich, wenn die Position gefunden wurde
if (position != null)
{
    // Bestellung abfragen (muss vorhanden sein!)
    Bestellungen bestellung = position.Bestellungen;

    // Kunde abfragen (muss vorhanden sein!)
    Kunden kunde = bestellung.Kunden;

    // Kundenname ausgeben
    Debug.Print("Kunde: {0}", kunde.Name);
}
```

Listing 1.30 ...und die Navigation von der Position zum Kunden (C#)

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Und als VB.NET-Quellcode:

```
' Objektkontext erstellen
Dim TonisTortenTraumContext As New TonisTortenTraumEntities()

' Eine Bestellungsposition ermitteln
Dim position As Bestellungspositionen =
    (From b In TonisTortenTraumContext.Bestellungspositionen).FirstOrDefault()

' Navigation nur möglich, wenn die Position gefunden wurde
If position IsNot Nothing Then

    ' Bestellung abfragen (muss vorhanden sein!)
    Dim bestellung As Bestellungen = position.Bestellungen

    ' Kunde abfragen (muss vorhanden sein!)
    Dim kunde As Kunden = bestellung.Kunden

    ' Kundenname ausgeben
    Debug.Print("Kunde: {0}", kunde.Name)
End If
```

Listing 1.31 ...und die Navigation von der Position zum Kunden (VB.NET)

HINWEIS In einer relationalen Datenbank werden solche Zusammenhänge bei Abfragen über SQL-Joins hergestellt. Solche Joins sind bei Entity SQL zwar auch möglich, jedoch meist nicht notwendig, soweit die Beziehung im Entitätenmodell bereits definiert ist.

Beziehungen

Beziehungen, auch Assoziationen genannt, sind das, was in einer relationalen Datenbank Relationen sind. Wie schon unter »Navigationseigenschaften« erläutert, wird so festgelegt, wie die einzelnen Entitäten zusammenhängen. So kann ein Kunde über beliebig viele Bestellungen verfügen, die ihrerseits wiederum über beliebige Bestellungspositionen verfügen. Beziehungen können von oben nach unten (hier von Kunde zu Bestellungen) als auch umgekehrt von unten nach oben (hier von einer Bestellung zum Kunden) verfolgt werden. Die geschieht über die bereits beschriebenen Navigationseigenschaften, welche für die Entitäten angelegt werden. Eine Beziehung »steht« dabei immer zwischen zwei Entitäten. Und je nachdem, wie viele Entitäten auf einer Seite und welche Anzahl von Entitäten auf der anderen Seite möglich sind, spricht man von unterschiedlichen Multiplizitäten (INDEX). Um bei dem Beispiel zu bleiben: Da ein Kunde beliebig viele Bestellungen haben kann, sind die Multiplizitäten auf der Seite der Kunden »1« und auf der der Bestellungen »*«, was für beliebig viele steht. Zusammen ist dies also »1:*«, was allerdings auch die Möglichkeit eröffnet, dass ein Kunde gar keiner Bestellung zugeordnet werden kann.[3]

Für jedes der beiden Enden sind die folgenden Multiplizitäten möglich, sodass deren Kombinationen die Möglichkeiten für Beziehungen darstellen:

3 Ob es sich dann dennoch um einen Kunden handelt, sollen Philosophen entscheiden.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Symbol/ Notation	Bedeutung
1	Genau eine Entität. Nicht mehr und nicht weniger.
0..1	Entweder eine oder keine
*	Beliebig viele Entitäten. Das schließt keine, eine und jede andere Anzahl ein.

Tabelle 1.3 Die erlaubten Multiplizitäten für Beziehungen zwischen Entitäten

Im grafischen Designer von Visual Studio finden sich zur leichteren Übersicht entsprechende Symbole dargestellt.

Abbildung 1.12 Die Beziehung zwischen Kunden (links) und Bestellungen (rechts)

HINWEIS Neu sind *:*-Beziehungen, die mit dem alten ADO.NET Entity Framework, welches mit .NET Framework 3.5 SP1 verfügbar war, nicht direkt möglich waren. Vielmehr musste ein umständlicher Weg über eine Behelfsentität gegangen werden, welche jeweils eine Beziehung zu den beiden Entitäten besaß, die in einer *:*-Beziehung standen. Das klingt nicht nur umständlich, das war es auch – besonders bei Abfragen.

Lazy Loading

Bei der Auseinandersetzung mit Beziehungen zwischen Entitäten eines Modells kommt schnell eine Frage auf: Was wird aus der Persistenzschicht geladen, wenn z.B. auf einen Kunden zugegriffen wird? Nur der Kunde oder gleich alle seine Bestellungen, deren Positionen usw.? Beide Ansätze haben Vor- und Nachteile. Während das komplette Lesen aus der Datenbank natürlich dazu führt, dass (zu) viele Daten gelesen werden, die möglicherweise an dieser Stelle nicht benötigt werden, können sich in Situationen, in denen sowieso alle Details der Kundenbestellungen benötigt werden, weitere Zugriffe auf die Datenbank erübrigen. Das Verhalten des Entity Framework sieht so aus, dass standardmäßig nur die direkten Objekte geladen werden, nicht jedoch diejenigen, die von diesen abhängig sind. In dem genannten Beispiel also nur der Kunde.

Das beschriebene Verhalten ist als Lazy Loading bekannt. Manchmal wird auch das genaue Gegenteil mit Eager Loading benannt. Da es nicht immer günstig ist, Lazy Loading zu verwenden oder nicht, kann dieses Verhalten durch eine Eigenschaft des Entitätenmodells gesteuert werden – je nachdem, was in der jeweiligen Situation das Sinnvollste ist.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Abbildung 1.13 Lazy Loading in den Eigenschaften des Entitätenmodells

Um den Standardwert für das Lazy Loading, der durch die Eigenschaften einem Entitätenmodell mitgegeben wird, zur Laufzeit zu verändern oder schlicht den Wert auszulesen, steht eine gleichnamige Eigenschaft zur Verfügung.

```
// Objektkontext erstellen
TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities();

// Wurde Lazy Loading aktiviert?
if (TonisTortenTraumContext.ContextOptions.LazyLoadingEnabled)
    // Aktiviert
else
    // Nicht aktiviert
```

Listing 1.32 Prüfen, ob Lazy Loading aktiviert wurde (C#)

Und als VB.NET-Quellcode

```
' Objektkontext erstellen
Dim TonisTortenTraumContext As New TonisTortenTraumEntities()

' Wurde Lazy Loading aktiviert?
If TonisTortenTraumContext.ContextOptions.LazyLoadingEnabled Then
    ' Aktiviert
Else
    ' Nicht aktiviert
End If
```

Listing 1.33 Prüfen, ob Lazy Loading aktiviert wurde (VB.NET)

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Mappings

Das ADO.NET Entity Framework kennt zwei unterschiedliche Arten von Mappings: Tabellenmappings, die festlegen, welche Entität und Eigenschaft welcher Tabelle und Spalte zugeordnet sein soll (*Tabellenmapping*) und Mappings für eigene Prozeduren für das Einfügen, Ändern und Löschen von Daten in der Persistenzschicht (*Mapping der gespeicherten Prozedur* genannt). Beide Arten finden Sie im Anschluss beschrieben.

Tabellenmappings

Mit so genannten Tabellenmappings lässt sich steuern, welcher Spalte in der Persistenzschicht welche Entität und welche Eigenschaft welcher Spalte zugeordnet ist.

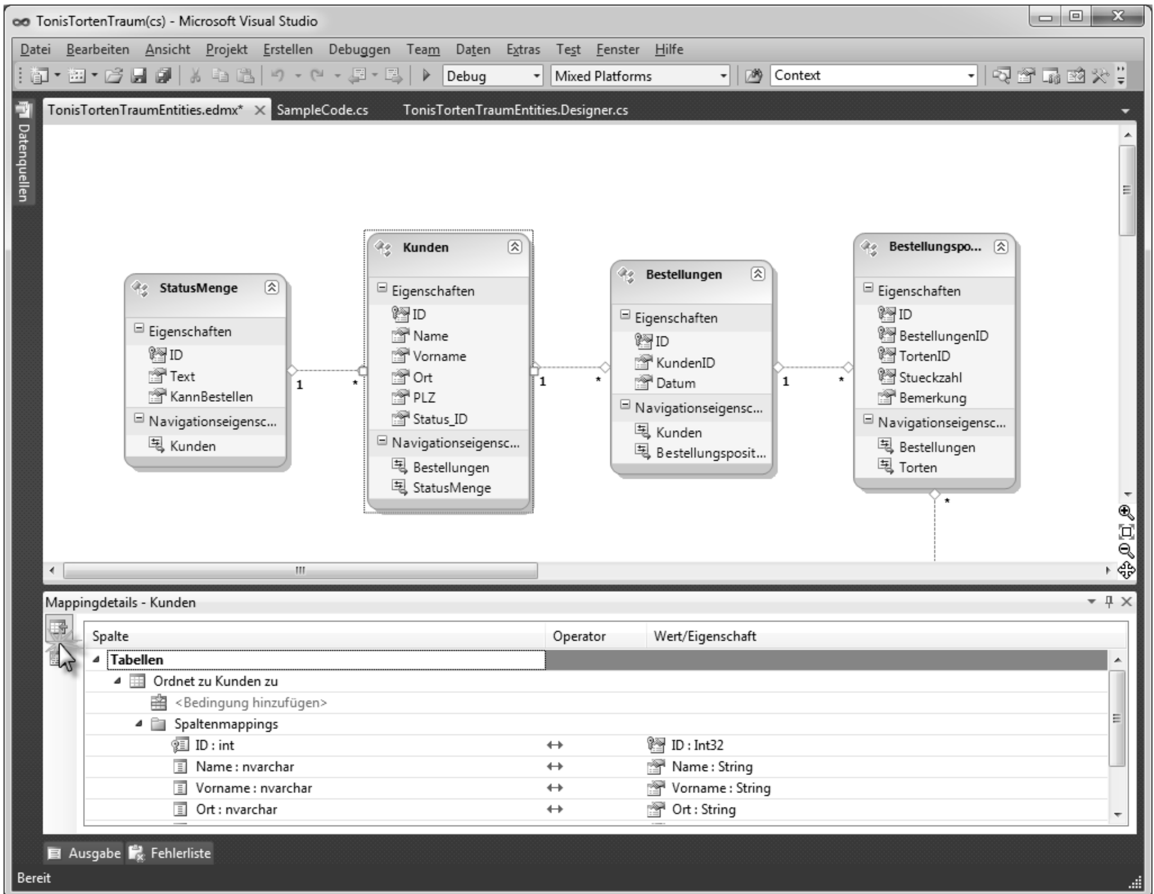

Abbildung 1.14 Tabellenmappings erlauben, Tabellen (und Spalten) aufgrund von Bedingungen für eine Entität festzulegen

Die Tabellenmappings können von Bedingungen abhängig sein, die dann steuern, wann welche Tabelle/Sicht für die Persistierung der Entität verwendet werden soll.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Eigene Prozeduren für das Einfügen, Ändern und Löschen

Wer möchte, kann das Einfügen, Ändern und Löschen von Entitäten in der Datenbank von eigenen Prozeduren erledigen lassen, die zuvor im Entitätenmodell angelegt wurden. Ín Visual Studio wird diese Möglichkeit etwas umständlich *Mapping der gespeicherten Prozedur* (kurz: Prozedurmapping) genannt und meint damit eine explizite Zuordnung für die CUD[4]-Aktionen zu einer Prozedur und im Detail die Zuordnung von Eigenschaften zu deren Parametern.

Notwendig ist diese sicherlich nicht, da dies Aufgabe des Datenanbieters ist. Er sorgt dafür, dass für den richtigen Zweck die jeweils korrekte Anweisung zur Verfügung steht, die dann auch die Datenbanktechnologie verstehen und ausführen kann. Eigene Prozeduren sind daher nicht nur aufwendig in der Entwicklung (schließlich müssen auch alle Änderungen wie zusätzliche/gelöschte Eigenschaften, andere Datentypen, etc. immer mit gepflegt werden), sondern auch ein gewisses Stück weit kontraproduktiv, da so eine Abhängigkeit zu einer bestimmten Datenbanktechnologie erzeugt wird. Sicherlich ist es auch denkbar – jeweils vom gewählten Datenanbieter abhängig – unterschiedliche Anweisungen zu entwerfen, doch ist dies für den Nutzen des Entity Framework der Todesstoß. Abgesehen davon, nimmt es dem Projekt die (zumindest theoretische) Chance, mit bis dato unbekannten Datenanbietern zusammen zu arbeiten. Wer diesen Weg wählen möchte, sollte überlegen, sich gegen das EF zu entscheiden und eine Eigenentwicklung zu nutzen.

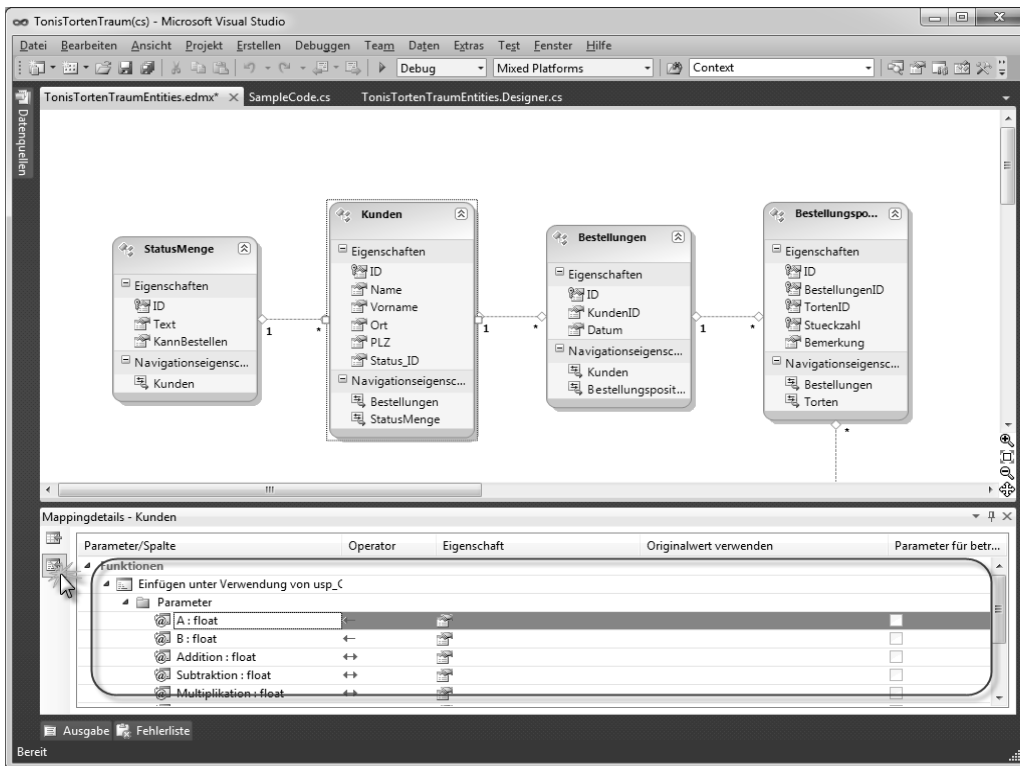

Abbildung 1.15 Prozedurmappings lassen eigene Prozeduren für das Einfügen, Ändern und Löschen zu

4 CUD: Create, Update & Delete

Sie erreichen die Prozedurmappings einer Entität, die standardmäßig im unteren Teil von Visual Studio angezeigt werden, über deren Kontextmenü (Befehl: *Mapping der gespeicherten Prozedur*). Beachten Sie, dass Sie links, an der Stelle, an der in Abbildung 1.15 der Mauszeiger dargestellt ist, zwischen den Mappings umschalten können/müssen.

Da das Erstellen eigener Anweisungen oder Prozeduren aus den angeführten Gründen eher von geringer Bedeutung ist, wird dieses Buch es bei der Erwähnung der Möglichkeit belassen.

Prozeduren/Funktionen

Unterstützt der Datenanbieter gespeicherte Prozeduren und Funkionen, so wie es z.B. Microsoft SQL Server tut, so ist es möglich, diese zum Importieren und als »gewöhnliche« Methoden und Funktionen für C#/VB.NET nutzbar zu machen. Im Modellbrowser sind die Funktionen sowohl im konzeptionellen als auch im physikalischen Modell gut zu erkennen.

Abbildung 1.16 Prozeduren/Funktionen der Datenbank im Modellbrowser

Per Code lässt sich diese Funktion dann wie eine normale Methode ansprechen:

```
// Objektkontext erzeugen
TonisTortenTraumEntities TonisTortenTraumContext = new TonisTortenTraumEntities();

// Funktionsaufruf durchführen
int Anzahl = TonisTortenTraumContext.fn_CountCustomersByPLZ("61130");
```

Listing 1.34 Der Aufruf einer importierten Funktion (C#)

Und als VB.NET-Quellcode:

```
' Objektkontext erzeugen
Dim TonisTortenTraumContext As New TonisTortenTraumEntities()

'Funktionsaufruf durchführen
Dim Anzahl As Integer = TonisTortenTraumContext.fn_CountCustomersByPLZ("61130")
```

Listing 1.35 Der Aufruf einer importierten Funktion (VB.NET)

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

Bei den Rückgabewerten von Funktionen kann es sich um eine der folgenden Möglichkeiten handeln.

Skalare

Einfache Werte wie Zahlen, Zeichenketten, etc. werden als Skalare zurückgeliefert.

Komplex

Komplexe Datentypen, so wie sie auch von komplexen Eigenschaften geliefert werden.

Entität

Liefert eine Entität oder genauer gesagt nicht eine einzelne, sondern eine Auflistung aus Entitäten zurück. Durch z.B. die `FirstOrDefault()`-Methode ist es aber auch ohne Weiteres möglich, das erste (und einzige) Element zu erhalten – schließlich ist ein einzelnes Element nur eine Sonderform einer recht kleinen Auflistung.

Praktischer Entwurf eines Entitätenmodells

Für den Entwurf eines Entitätenmodells stehen zwei unterschiedliche Ansätze zur Auswahl. Welcher der richtige ist, hängt davon ab, ob die Datenbank für die Entitäten bereits fertig ist (*Database-first*) oder ob zunächst das Entitätenmodell vorhanden ist, also erstellt wurde oder wird, und anschließend die entsprechende Datenbank erstellt werden soll.

In beiden Fällen wird das Entitätenmodell, wie auch gewöhnliche Klassen, Schnittstellen, Formen, etc. als neues Element einem Projekt zugefügt. Dies kann alternativ über das Kontextmenü des Projekts geschehen oder alternativ durch die Tastenkombination Strg+⇧+A.

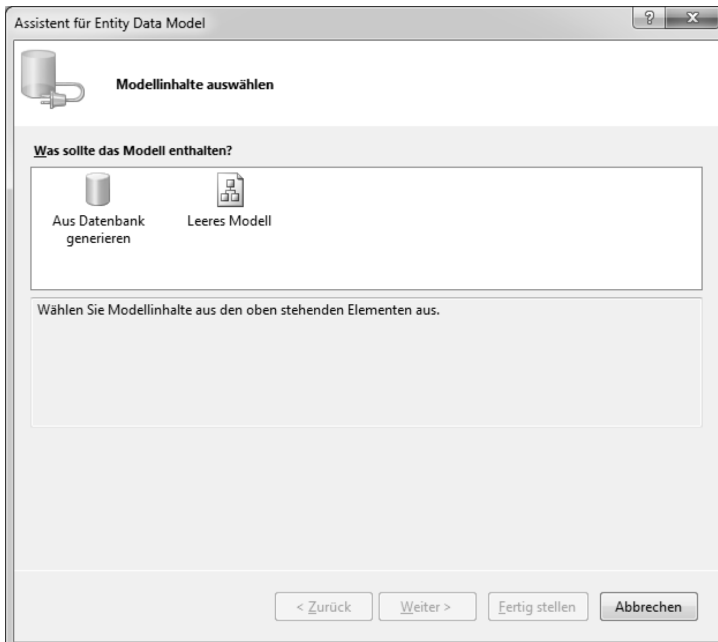

Abbildung 1.17 Entitätenmodelle werden wie andere Elemente auch einem Projekt hinzugefügt

Auch wenn die deutsche Übersetzung nicht ganz einwandfrei ist, stehen diese beiden grundlegenden Alternativen zur Auswahl:

- *Aus Datenbank generieren* ermöglicht es, das Modell aus einer bestehenden Datenbank generieren zu lassen. Da die Datenbank zuerst existiert, entspricht dies dem *Database-first-Ansatz*.
- *Leeres Modell* ist die richtige Wahl, wenn das Modell zunächst erst einmal erstellt und schließlich auf die Datenbank übertragen werden soll. Dies ist der *Model-First-Ansatz*

Beide Ansätze werden hier kurz erläutert und in Kapitel 2 näher beschrieben.

HINWEIS Auch zur Laufzeit kann über den Objektkontext die benötigte Datenbank erzeugt, gelöscht und geprüft werden, ob diese vorhanden ist. Weitere Details zum genauen Vorgehen finden Sie in Kapitel 3.

Model-first-Ansatz

Existiert zunächst das Entitätenmodell, wurde dies also quasi am Reißbrett erzeugt und ist der Entwurf abgeschlossen, so kann aus dem Modell eine Datenbank erzeugt werden. Werden später weitere Änderungen an dem Modell vorgenommen, so macht dies das erneute Erzeugen der Datenbank notwendig. Ein »Update« ist zurzeit nicht möglich.

Database-first-Ansatz

Ist die Datenbank bereits vorhanden, so kann nach deren Aufbau das Entitätenmodell erzeugt werden. Neben Tabellen, Sichten und Prozeduren/Funktionen können auch Beziehungen zwischen Tabellen/Sichten berücksichtigt werden.

Welcher Ansatz ist der richtige?

Sicherlich ist das eine sehr pauschale Frage und die Antwort mitunter nicht weniger pauschal, doch gibt es bei der aktuellen Version des Entity Framework und von Visual Studio einige Dinge zu bedenken, bevor der Entwurf beginnt. Der Model-First-Ansatz ist sicherlich der richtige für die Anfangsphase. Steht fest, was benötigt wird, so kann das Modell bequem erstellt werden. Die benötigte Datenbank für die Entwicklung ist nach Abschluss auch schnell erstellt. Müssen im Fortgang der Entwicklung jedoch Änderungen am Entitätenmodell vorgenommen werden, die sich auf die Datenbank auswirken, so müssen in der Datenbank mindestens die betroffenen Tabellen gelöscht und neu erstellt werden – und etwaige Daten müssen erneut eingespielt werden! Führt dies zu unangemessen großem Aufwand, so besteht die ratsame Möglichkeit, erst die Datenbank entsprechend anzupassen und anschließend das Modell entsprechend zu aktualisieren. Ist für die Abfragen größtenteils LINQ verwendet worden, so wird der Compiler an all den Stellen Fehler melden, an denen die Anpassungen die Kompatibilität zwischen Alt und Neu zerstört haben.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

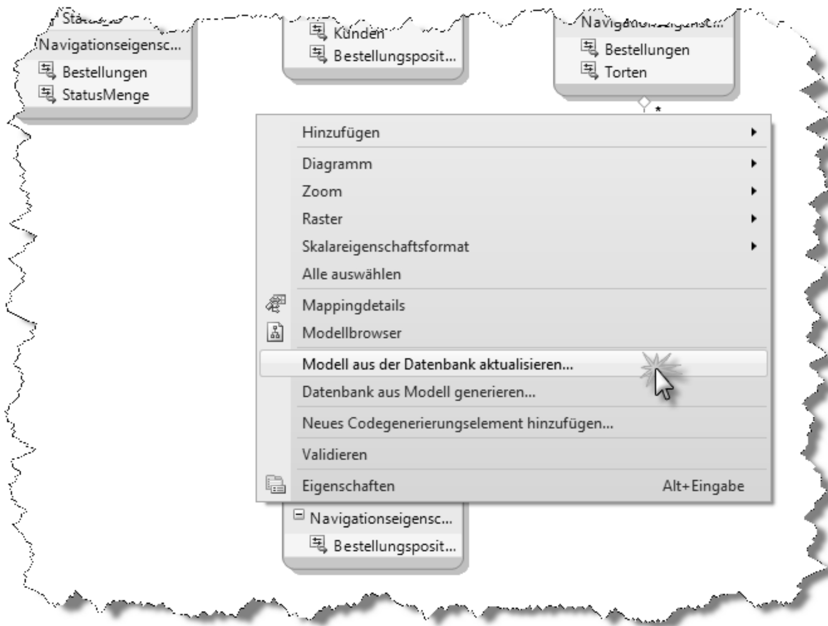

Abbildung 1.18 Das Entitätenmodell kann aus der Datenbank aktualisiert werden

Bei Migrationsprojekten, welche das Entity Framework einführen, ist der Database-First-Ansatz unschlagbar, da er viel monotone Arbeit bei der Nachbildung der Tabellen in Entitäten erspart. Jedoch wird auch hier der Punkt kommen, wo Entitätenmodell/Datenbank angepasst werden müssen. An dieser Stelle ist es in den meisten Situationen das Beste, erst die Datenbank anzupassen und anschließend das Modell zu aktualisieren.

T4-Vorlagen

T4-Vorlagen sind, wie schon an anderer Stelle betont, nichts anderes als Code um Code zu erstellen. Visual Studio führt nach jeder Änderung am Entitätenmodell diesen Code aus und erzeugt damit alle notwendigen Klassen – vom Objektkontext bis hin zu den Entitätenklassen. Der Weg, diese mit einem Code anzupassen (direkte Änderungen verbieten sich von selbst), ist das Anlegen anderer partieller Klassenteile in getrennten Dateien (mittels `partial`/`Partial`-Schlüsselwort), vervollständigt durch die Implementierung partieller Methoden.

HINWEIS Mehr zu partiellen Klassen und Methoden finden Sie in Kapitel 3 jeweils unter dem Abschnitt der einzelnen Klassen unter »Genereller Aufbau der Designer-Code-Klassen«. Die Dateierweiterung von T4-Vorlagen ist *.tt*.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

TIPP T4-Vorlagen gab es unter Umwegen bereits in Visual Studio 2008. Dort reichte es, die Erweiterung einer bestehenden Datei im Projekt in »tt« zu ändern, schon wurde sie als T4-Vorlage behandelt – entsprechenden Inhalt natürlich vorausgesetzt.

Auf diesem Weg konnte man sich diese nützliche Technologie auch vor Visual Studio 2010 nutzbar machen.

Ein weiterer Trick bezogen auf T4-Vorlagen ist der, dass sie für das Entity Framework ausgetauscht werden können. Aus einem Modell können so unterschiedliche Codedateien erzeugt werden. Während der Standard die gewohnten Klassen für Objektkontext, Entitäten, etc. erzeugt, reicht ein Austauschen, um aus dem gleichen Entitätenmodell z.B. *Entitäten zur Selbstnachverfolgung* (besser unter ihrem englischen Namen *Self-tracking-entitities* bekannt) generieren zu lassen.

Praktisch geschieht dies über das Kontextmenü des Entitätenmodells (leerer Bereich) und den Befehl *Neues Codegenerierungselement hinzufügen*. Die Dialogfelder unterscheiden sich zwischen C# und VB.NET nur geringfügig:

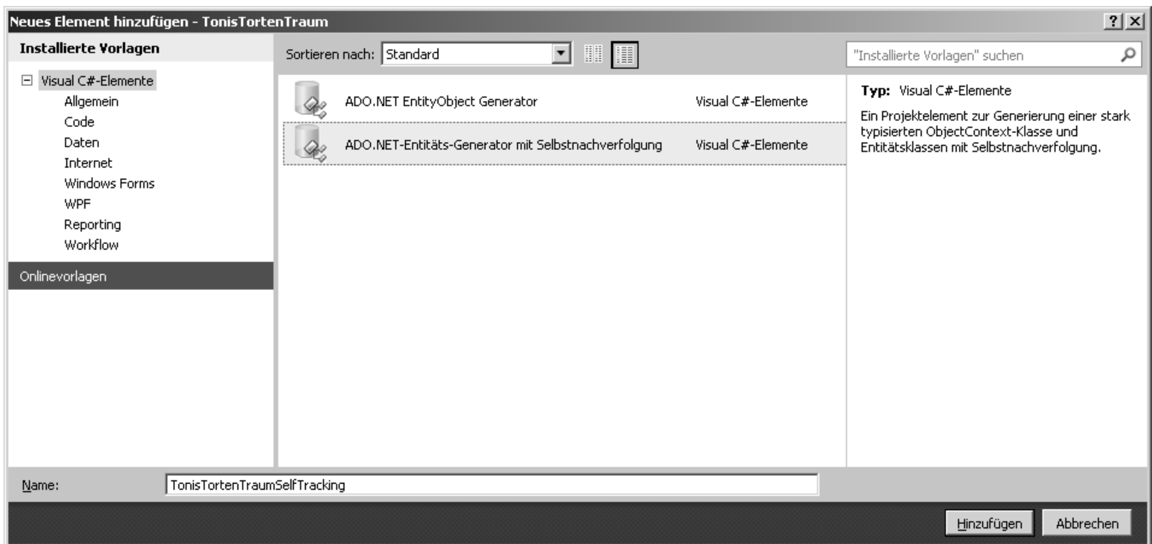

Abbildung 1.19 Diese Codegenerierungselemente stehen aktuell bei der Visual Studio 2010-Installation zur Auswahl

HINWEIS Visual Studio 2010 zeigt eine Sicherheitswarnung an, die vor potenziellen Gefahren warnt. Der offensichtliche Grund liegt darin, dass die Freiheit des Codes in den T4-Vorlagen auch für schädliche Aktionen auf dem Computer genutzt werden könnte.

Im Projektmappen-Explorer sieht das später dann so aus:

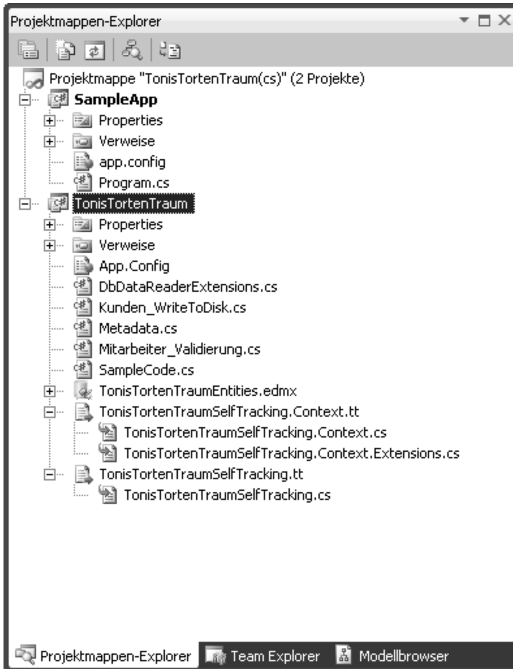

Abbildung 1.20 Das Ergebnis im Projektmappen-Explorer

Da das Erstellen von T4-Vorlagen mit den Bordmitteln von Visual Studio 2010 etwas mühsam und fehleranfällig ist, gibt es eine reiche Auswahl an Tools von Drittherstellern, welche sowohl die Entwicklung als auch die Fehlersuche unterstützen.

Im Anschluss finden Sie in ein paar Beispielzeilen, wie eine solche T4-Vorlage (in C# und VB.NET, je nach Projektsprache) aussieht. Zwischen den Tags `<#` und `#>` befinden sich die Befehle, die ausgeführt werden. In welcher Programmiersprache diese vorliegen müssen, wird über das Attribut `template language` bestimmt.

```
<#@ template language="C#" debug="false" hostspecific="true"#>
<#@ include file="EF.Utility.CS.ttinclude"#><#@
 output extension=".cs"#><#
// Copyright (c) Microsoft Corporation.  All rights reserved.

CodeGenerationTools code = new CodeGenerationTools(this);
MetadataTools ef = new MetadataTools(this);
MetadataLoader loader = new MetadataLoader(this);
CodeRegion region = new CodeRegion(this);

EntityFrameworkTemplateFileManager fileManager = EntityFrameworkTemplateFileManager.Create(this);

string inputFile = @"$edmxInputFile$";
EdmItemCollection ItemCollection = loader.CreateEdmItemCollection(inputFile);
string namespaceName = code.VsNamespaceSuggestion();

EntityContainer container = ItemCollection.GetItems<EntityContainer>().FirstOrDefault();
if (container == null)
{
    return "//$Localized_STECtx_Comment_10$";
}
```

```
WriteHeader(fileManager);
BeginNamespace(namespaceName, code);

#>
<#=Accessibility.ForType(container)#> partial class <#=code.Escape(container)#> : ObjectContext
{
public const string ConnectionString = "name=<#=container.Name#>";
public const string ContainerName = "<#=container.Name#>";

#region Constructors

public <#=code.Escape(container)#>()
    : base(ConnectionString, ContainerName)
{
    Initialize();
}

public <#=code.Escape(container)#>(string connectionString)
    : base(connectionString, ContainerName)
{
    Initialize();
}

public <#=code.Escape(container)#>(EntityConnection connection)
    : base(connection, ContainerName)
{
    Initialize();
}

private void Initialize()
{
    //$Localized_STECtx_Comment_50$
    //$Localized_STECtx_Comment_60$
    ContextOptions.ProxyCreationEnabled = false;
    ObjectMaterialized += new ObjectMaterializedEventHandler(HandleObjectMaterialized);
}

private void HandleObjectMaterialized(object sender, ObjectMaterializedEventArgs e)
{
    var entity = e.Entity as IObjectWithChangeTracker;
if (entity != null)
    {
        bool changeTrackingEnabled = entity.ChangeTracker.ChangeTrackingEnabled;
        try
        {
            entity.MarkAsUnchanged();
        }
        finally
        {
        entity.ChangeTracker.ChangeTrackingEnabled = changeTrackingEnabled;
        }
        this.StoreReferenceKeyValues(entity);
    }
}
```

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

```
#endregion

<#
    region.Begin("$Localized_STECtx_Comment_70$", 2);

    foreach (EntitySet entitySet in container.BaseEntitySets.OfType<EntitySet>())
    {
#>
…
```

Listing 1.36 Auszug auf dem T4-Skript zur Generierung des Objektkontextes (C#)

Und in VB.NET:

```
<#@ template language="VB" debug="false" hostspecific="true"#>
<#@ include file="EF.Utility.VB.ttinclude"#><#@
 output extension=".vb"#><#
' Copyright (c) Microsoft Corporation.  All rights reserved.

Dim code As New CodeGenerationTools(Me)
Dim ef As New MetadataTools(Me)
Dim loader As New MetadataLoader(Me)
Dim region As New CodeRegion(Me)

Dim fileManager As EntityFrameworkTemplateFileManager = EntityFrameworkTemplateFileManager.Create(Me)

Dim inputFile As String = "$edmxInputFile$"
Dim ItemCollection As EdmItemCollection = loader.CreateEdmItemCollection(inputFile)
Dim namespaceName As String = code.VsNamespaceSuggestion()

Dim container As EntityContainer = ItemCollection.GetItems(Of EntityContainer)().FirstOrDefault()
If container Is Nothing Then
    Return "'$Localized_STECtx_Comment_10$"
End If

WriteHeader(fileManager)
BeginNamespace(namespaceName, code)

#>
Partial <#=Accessibility.ForType(container)#> Class <#=code.Escape(container)#>
    Inherits ObjectContext
    Public Const SettingsConnectionString As String = "name=<#=container.Name#>"
    Public Const ContainerName As String = "<#=container.Name#>"

<#
region.Begin("$Localized_STECtx_Comment_50$")
#>

    Public Sub New()
        MyBase.New(SettingsConnectionString, ContainerName)
        Initialize()
    End Sub
```

```
    Public Sub New(ByVal connectionString As String)
        MyBase.New(connectionString, ContainerName)
        Initialize()
    End Sub

    Public Sub New(ByVal connection As EntityConnection)
        MyBase.New(connection, ContainerName)
        Initialize()
    End Sub

    Private Sub Initialize()
        '$Localized_STECtx_Comment_60$
        '$Localized_STECtx_Comment_70$
        ContextOptions.ProxyCreationEnabled = False
        AddHandler ObjectMaterialized, AddressOf HandleObjectMaterialized
    End Sub

    Private Sub HandleObjectMaterialized(ByVal sender As Object, ByVal e As ObjectMaterializedEventArgs)
        Dim entity As IObjectWithChangeTracker = TryCast(e.Entity, IObjectWithChangeTracker)
        If entity IsNot Nothing Then
            Dim changeTrackingEnabled As Boolean = entity.ChangeTracker.ChangeTrackingEnabled
            Try
                entity.MarkAsUnchanged()
            Finally
                entity.ChangeTracker.ChangeTrackingEnabled = changeTrackingEnabled
            End Try
            Me.StoreReferenceKeyValues(entity)
        End If
    End Sub
<#
region.End()

region.Begin("$Localized_STECtx_Comment_80$")
…
```

Listing 1.37 Auszug auf dem T4-Skript zur Generierung des Objektkontextes (VB.NET)

Die »Ausgabe« dieser T4-Vorlagen, welche automatisch im Hintergrund kompiliert und ausgeführt werden, sind wiederum Codedateien in C# oder VB.NET, welche wie gewöhnlich kompiliert werden.

HINWEIS Die Ausgabe ist keineswegs auf Codedateien beschränkt und schon gar nicht ausschließlich auf solche in C#/VB.NET. Denkbar ist auch, auf diesem Wege anderen Inhalt zu erzeugen. Doch dürften Quelltexte in einer .NET-Sprache wohl das häufigste Endergebnis sein – schließlich wurde T4 zu diesem Zweck entwickelt.

Metadata Workspace

Der so genannte Metadata Workspace, oft auch nur als *Metadata* bezeichnet, beinhaltet alle deklarativen Bestandteile des Entitätenmodells. Vereinfacht ausgedrückt bedeutet dies, dass hier im XML-Format alle Informationen für das konzeptionelle und für das physikalische Modell untergebracht sind, welche das Entitätenmodell ausmachen. Beide Informationsarten werden zur Laufzeit benötigt. Aus dem konzeptionellen Modell werden zudem via T4-Skript (siehe Abschnitt »T4-Vorlagen« in diesem Kapitel) alle Klassen für Objektkontext, Entitäten, etc. generiert.

HINWEIS Zusätzlich zu diesen auch zur Laufzeit benötigten Angaben, werden hier einige Informationen abgelegt, die »nur« für den grafischen Designer von Visual Studio 2010 von Belang sind und festlegen, wo in der Darstellung welche Entität liegt und wo welche Beziehungslinie verläuft.

Physikalisches Modell

Im physikalischen Modell werden alle Zuordnungen zwischen Eigenschaften der Entitäten hin zu den Spalten der Tabelle festgelegt. Die dafür erdachte Sprache heißt *Store Schema Definition Language*, kurz SSDL.

```
<edmx:Runtime>
  <!-- SSDL content -->
    <edmx:StorageModels>
    <Schema Namespace="TonisTortenTraumModel.Store" Alias="Self" Provider="System.Data.SqlClient"
ProviderManifestToken="2008"
xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
xmlns="http://schemas.microsoft.com/ado/2009/02/edm/ssdl">
        <EntityContainer Name="TonisTortenTraumModelStoreContainer">
          <EntitySet Name="Bestellungen" EntityType="TonisTortenTraumModel.Store.Bestellungen"
store:Type="Tables" Schema="dbo" />
          <EntitySet Name="Bestellungspositionen"
EntityType="TonisTortenTraumModel.Store.Bestellungspositionen" store:Type="Tables" Schema="dbo" />
          <EntitySet Name="Kunden" EntityType="TonisTortenTraumModel.Store.Kunden" store:Type="Tables"
Schema="dbo" />
          <EntitySet Name="StatusMenge" EntityType="TonisTortenTraumModel.Store.StatusMenge"
store:Type="Tables" Schema="dbo" />
          <EntitySet Name="Torten" EntityType="TonisTortenTraumModel.Store.Torten" store:Type="Tables"
Schema="dbo" />
          <AssociationSet Name="FK_Bestellungen_Kunden"
Association="TonisTortenTraumModel.Store.FK_Bestellungen_Kunden">
            <End Role="Kunden" EntitySet="Kunden" />
            <End Role="Bestellungen" EntitySet="Bestellungen" />
          </AssociationSet>
…
```

Listing 1.38 Auszug aus der Definition des physikalischen Modells

Konzeptionelles Modell

Im konzeptionellen Modell sind alle Entitäten, ihre Eigenschaften und Beziehungen definiert. Die dafür erdachte Sprache heißt *Conceptual Schema Definition Language*, kurz CSDL.

```
<edmx:Runtime>
...
    <!-- C-S mapping content -->
    <edmx:Mappings>
      <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2008/09/mapping/cs">
        <EntityContainerMapping StorageEntityContainer="TonisTortenTraumModelStoreContainer"
CdmEntityContainer="TonisTortenTraumEntities">
```

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

```
          <EntitySetMapping Name="Bestellungen"><EntityTypeMapping
TypeName="TonisTortenTraumModel.Bestellungen"><MappingFragment StoreEntitySet="Bestellungen">
            <ScalarProperty Name="ID" ColumnName="ID" />
            <ScalarProperty Name="KundenID" ColumnName="KundenID" />
            <ScalarProperty Name="Datum" ColumnName="Datum" />
          </MappingFragment></EntityTypeMapping></EntitySetMapping>
          <EntitySetMapping Name="Bestellungspositionen"><EntityTypeMapping
TypeName="TonisTortenTraumModel.Bestellungspositionen"><MappingFragment
StoreEntitySet="Bestellungspositionen">
            <ScalarProperty Name="ID" ColumnName="ID" />
            <ScalarProperty Name="BestellungenID" ColumnName="BestellungenID" />
            <ScalarProperty Name="TortenID" ColumnName="TortenID" />
            <ScalarProperty Name="Stueckzahl" ColumnName="Stueckzahl" />
            <ScalarProperty Name="Bemerkung" ColumnName="Bemerkung" />
          </MappingFragment></EntityTypeMapping></EntitySetMapping>
          <EntitySetMapping Name="StatusMenge"><EntityTypeMapping
TypeName="TonisTortenTraumModel.StatusMenge"><MappingFragment StoreEntitySet="StatusMenge">
            <ScalarProperty Name="ID" ColumnName="ID" />
            <ScalarProperty Name="Text" ColumnName="Text" />
...
```

Listing 1.39 Auszug aus der Definition des konzeptionellen Modells

Windows Workflow Foundation

Nanu? Ein Abschnitt über die Windows Workflow Foundation (WF) in diesem Buch? Hat sich da der Autor nicht vertan? Mitnichten! Die WF, die zur einfachen Erstellung und Ausführung von Workflows Bestandteil des .NET Framework ist, wird intern bei der Generierung der Datenbank (der Skripte) für deren Erstellung verwendet. Erkennbar ist dies an der Eigenschaft *Workflow zur Datenbankgenerierung*, die eine XAML[5]-Datei aufführt.

Diese auf XML basierende Beschreibungssprache definiert den Workflow, der für die erwähnte Generierung herangezogen wird.

[5] Extensible Application Markup Language.
Ja, XAML wird nicht ausschließlich in Zusammenhang mit WPF verwendet.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

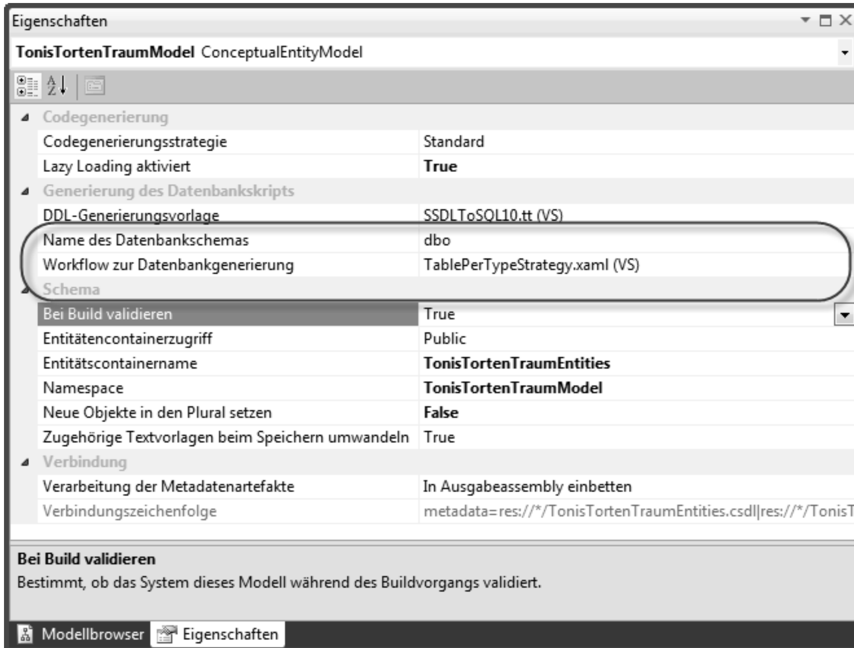

Abbildung 1.21 Der WF-Workflow, der für die Generierung der Datenbank (via Skript) verwendet wird

Wer möchte, kann diese Datei auch auf dem eigenen Computer suchen (sie liegt standardmäßig unter *%ProgramFiles%\Microsoft Visual Studio 10.0\Common7\IDE\Extensions\Microsoft\Entity Framework Tools\DBGen*) und öffnen. Das Öffnen kann sowohl in einem Editor der Wahl oder dem grafischen WF-Designer von Visual Studio 2010 geschehen. Letzteres geschieht über das Hinzufügen eines vorhandenen Elements zu einem bestehenden Projekt und ergibt sicherlich die interessantere Ansicht.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)

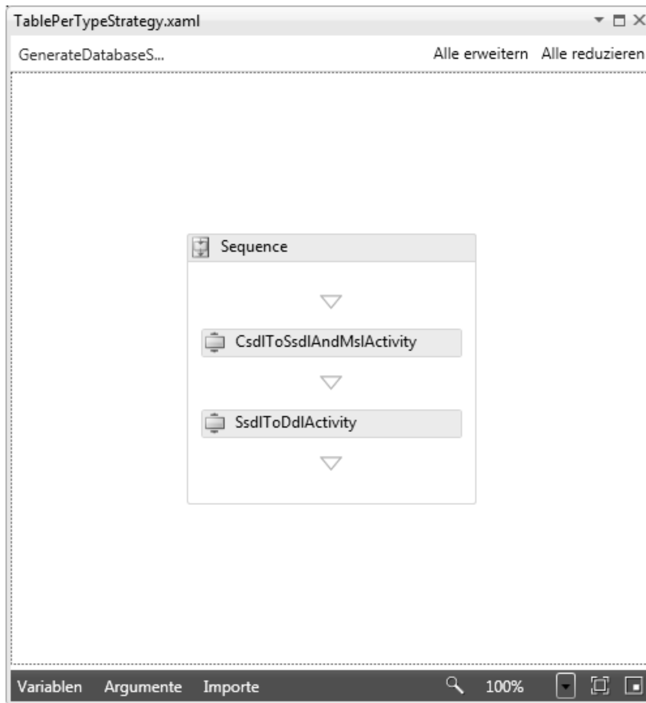

Abbildung 1.22 Der Standard Workflow im grafischen Designer von Visual Studio

HINWEIS Beachten Sie, dass es nach diesem »Experiment« zu Fehlern beim Kompilieren kommen kann, die darauf zurückzuführen sind, dass einige Assembly-Referenzen für WPF im Projekt nicht vorhanden sind. Dies liegt daran, dass Visual Studio die WPF-Kernkomponenten benötigt, um die XAML-Datei fälschlicherweise als WPF-Inhalt zu kompilieren.

Ein Hinweis zum Ende dieses Kapitels. Auch wenn der Autor es nicht selbst ausprobiert hat: Änderungen an diesem Workflow sollten vermieden und – wenn die Neugier mit einem durchgeht – nur mit einem griffbereiten Backup der Datei durchgeführt werden – sonst nimmt es kein gutes Ende.

Thorsten Kansy: Programmieren mit dem ADO.NET Entity Framework. Der O/R-Mapper im Detail,
Microsoft Press 2011 (ISBN 978-3-86645-461-3)