

32 SQL Server 2005 und ADO.NET

984	SQL Server 2005 Express
992	Grundsätzliches zu Datenbanken
998	Programmieren mit ADO.NET
1009	Ändern und Ergänzen von Daten in Datentabellen
1021	DataSets und typisierte DataSets
1037	Und so geht es weiter

Wenn Anwendungen entwickelt werden, dann müssen diese in 80 % aller Fälle mit Daten hantieren. In der Regel greift dann nicht nur *ein* Computer auf die Daten zu, sondern *mehrere* Computer müssen zur gleichen Zeit mit einer Datenquelle arbeiten. Zwar ist das Framework durchaus in der Lage, auch komplexere Datenstrukturen über Rechengrenzen hinaus zu verwalten, doch der Einsatz von Datenbanksystemen ist für solche Lösungen angezeigt. Doch auch für Einzelplatzlösungen ist der Einsatz von Datenbankanwendungen eine sinnvolle Angelegenheit. Dank SQL (*Structured Query Language*, etwa: strukturierte Abfragesprache) können Daten auf einfache Weise sortiert und gefiltert werden, und da Daten sowieso nach einer Arbeitssitzung gespeichert werden müssen, bietet sich der Einsatz einer Datenbank auch für Insellösungen mehr als an.

Im Framework gibt es ein mächtiges Werkzeug zum Abfragen und Verwalten von Daten in Datenbanken namens ADO.NET. ADO ist die Abkürzung von *ActiveX Data Objects* (etwa: *ActiveX-Datenobjekte*), und es stellt eine Klassenbibliothek zur Verfügung, mit deren Hilfe die verschiedensten Datenbanksysteme gesteuert werden können – sei es Microsofts SQL Server,¹ Oracle oder Access, um nur einige wenige zu nennen.

Durch eine weitestgehend vorhandene Standardisierung von SQL auf der einen und einem konsequenten Einsatz von Klassen und Klassenvererbung im Framework auf der anderen Seite, unterscheiden sich die Vorgehensweisen bei der Programmierung mit den verschiedenen Datenanbietern obendrein nur marginal.

¹ Sprich: »Biehkwl sörwer«.

HINWEIS: Einige der in diesem Kapitel vorgestellten Funktionen können Sie leider nicht mit der Visual Basic Express Edition durchführen, weil ihr die entsprechende Funktionalität fehlt (wie beispielsweise der Server-Explorer). Das bedeutet natürlich nicht, dass Sie keine Datenbankprogrammierung mit Visual Basic Express durchführen können! Visual Basic Express leistet sprachtechnisch das Gleiche, wie jede andere Version von Visual Basic bzw. Visual Studio, und dazu gehört natürlich auch ADO.NET.

SQL Server 2005 Express

SQL Server Express ist der Nachfolger der Microsoft SQL Server Desktop Engine, vielen eher bekannt unter dem Namen MSDE. Beide basieren auf ihren größeren Brüdern, SQL Server 2005 bzw. SQL Server 2000, ihnen fehlt jedoch einiges in Sachen Funktionalität für Administrierungsaufgaben. Darüber hinaus erfuhren sie ebenfalls ein paar Einschnitte in ihrer Leistungsfähigkeit, aber dafür stellt Microsoft beide Systeme unendgeldlich zur Verfügung, und sie lassen sich damit als professionelle Datenbankplattformen in Ihren eigenen Anwendungen einsetzen.

SQL Server Express erfuhrt die folgenden Einschränkungen:

- Datenbanken sind auf maximal 4 GByte beschränkt.
- Maximal ein Prozessor eines Systems wird verwendet.
- Maximal ein Gigabyte Hauptspeicher eines Systems kommt zur Anwendung.
- Maximal fünf so genannter Pipes werden gleichzeitig verarbeitet. Damit lässt sich, natürlich je nach wirklicher Auslastung, eine die Datenbank durchschnittlich in Anspruch nehmende Büroanwendung durchaus mit 20, 30 Clients betreiben, bevor Leistungseinschnitte bemerkbar werden.

HINWEIS: Denken Sie daran, dass SQL Server Express sich auch auf Computern installieren lässt, die über mehr Hauptspeicher oder mehrere Prozessoren verfügen. Diese werden dann lediglich nicht genutzt.

Denken Sie auch daran, dass Sie sich bei Microsoft explizit registrieren lassen müssen, bevor Sie eine eigene Anwendung auf Basis von SQL Server Express vertreiben. Das kostet zwar nichts (Stand der Drucklegung dieses Buches), Sie müssen es aber halt machen. Der IntelliLink *G3201* bringt Sie direkt zu dieser Registrierungsseite (Sie benötigen dafür ein Microsoft Passport Konto).

Von diesen Einschränkungen abgesehen, steht Ihnen mit SQL Server Express eine Datenbankplattform zur Verfügung, die dem großen Bruder – was die relationale Datenbank angeht – in beinahe nichts nachsteht. Umfangreiche, höchst performante Datenbanken lassen sich mit dieser Software realisieren und die volle Programmierfähigkeit mit gespeicherten Prozeduren (*Stored Procedures*), Funktionen und auch die Ausführungsfähigkeit von verwaltetem Code steht Ihnen mit SQL Server Express zur Verfügung.

Sie können SQL-Express direkt von der Microsoft Website herunterladen. Folgen Sie dazu dem IntelliLink *G3201*.

Einsatz von SQL Server Express in eigenen Anwendungen

SQL Server Express wird ab Visual Studio 2005 Standard Edition direkt mitinstalliert. Solange, wie Sie sich quasi »zu Hause«, in Ihrer eigenen Entwicklungsumgebung befinden, werden Ihnen daher keine Probleme beim Aufbau, der Programmierung oder der Anwendung einer Datenbank begegnen.

Anders sieht das schon aus, wenn Sie SQL Server Express später in seiner Produktivumgebung einsetzen wollen. Dort begegnen Ihnen dann unter Umständen Herausforderungen, mit denen Sie sich nicht erst zum Zeitpunkt der Installation Ihrer Anwendung beschäftigen sollten, denn:

Anwendungen, die auf SQL Server Express basieren, lassen sich solange problemlos installieren, wie SQL Server Express auf dem gleichen Rechner wie Ihre Anwendung selbst läuft. Etwas aufwändiger wird es, wenn

- Ihre Anwendung von mehreren Rechnern aus auf eine SQL Server Express-Instanz zugreifen soll.
- Ihnen kein Server-Betriebssystem, das als Active Directory eingerichtet ist, als Plattform für SQL Server Express zur Verfügung steht, oder Sie SQL Server Express nicht zumindest auf einem Computer (Windows 2000 SP4, XP Professional, eine Windows Vista-Version mit Domänenanbindungsmöglichkeit) installieren können, der Mitglied einer Domäne ist.

Der Grund: Standardmäßig wird SQL Server Express (wie übrigens auch seine großen Brüder) so installiert, dass es die integrierte Sicherheit des Betriebssystems für Anmeldungen an einer SQL Server-Instanz oder einer Datenbank nutzt. Solange wie ein Benutzer (oder eine Anwendung) sich auf dem gleichen System an der SQL Server-Instanz anmeldet, auf dem SQL Server Express selbst auch installiert wurde, ist das kein Problem. Problematisch wird bei dieser Anmeldemethode die Anmeldung von Rechnern des Netzwerkes, wenn kein Domänencontroller zur Verfügung steht, der eine vertraute Verbindung zwischen dem Client-Computer und dem Computer, der die SQL Server-Instanz beherbergt, bestätigen könnte.²

Für solche Fälle ist die so genannte *Gemischter Modus*-Authentifizierung angezeigt. Bei dieser Art der Authentifizierung können Sie sowohl die integrierte Sicherheit von Windows als auch die Authentifizierungsmethode verwenden, die vom SQL Server-System selbst zur Verfügung gestellt wird. Der Nachteil: Letzterer besteht in geringeren Sicherheitsstandards. Um eine Verbindung auf diese Weise zu einem SQL Server aufzubauen, müssen Kennwort und Benutzername beim Verbindungsaufbau angegeben werden. Die Wahrscheinlichkeit, dass Anmeldedaten dadurch kompromittiert werden könnten, ist daher ungleich höher, wenn sie nicht auf geschickte Weise in Ihrer Software verschlüsselt werden.

Ein weiterer Nachteil ist, dass Sie in Netzwerken, in denen Active Directory nicht zur Verfügung steht, keine automatisierte Installation (die so genannte »stille Installation« von SQL Server Express) durchführen sollten, da diese standardmäßig *Integrated Security* als Zugangsart einrichtet. Um sich spätere Konfigurationsarbeit zu sparen, ergibt es mehr Sinn, schon während der Installation ein Administratorkennwort für die Anmeldung an der Instanz ohne integrierte Sicherheit vergeben und auch die Anmeldemethode direkt auf *Gemischter Modus* einstellen. Da entsprechende Administrationswerkzeuge fehlen, ist die spätere Umstellung nur mit entsprechenden Systemaufrufen durch

² Jedenfalls wenn Sie als Administrator am Rechner angemeldet sind – was die meisten Entwickler ja leider häufig sind. Als »normaler« Benutzer muss man auch erst einmal gezielt Zugriff vom SQL Server erteilt bekommen... `CREATE LOGIN` bzw. `CREATE USER FROM LOGIN` lauten hier die Stichworte, die genauer zu beschreiben den Rahmen an dieser Stelle allerdings sprengen würden.

gespeicherte Prozeduren zu bewerkstelligen und damit für ungeübte Daten-Banker nicht unbedingt so ohne weiteres machbar.³

Installation von SQL Server 2005 Express unter Windows XP im »gemischten Modus«

Die folgende Schritt-für-Schritt-Anleitung zeigt, wie Sie SQL Server Express unter Windows XP mit vorhandenem Service Pack 2 (SP2) installieren, wenn Ihnen kein Domänencontroller zur Verfügung steht. Sie haben die Möglichkeit, eine »stille Installation« durchführen zu lassen, oder aber die Konfiguration mithilfe eines Assistenten zu durchlaufen. Die folgenden Abschnitte beschreiben die assistentengestützte Installation. Sie setzen die finale deutsche Version von *SQL Server 2005 Express Edition* voraus.

- Sie starten die manuelle Installation mit einem Doppelklick auf *sqlexpr.exe*. Wenn Sie die Installation von SQL Server Express gestartet haben, sehen Sie anschließend folgenden Dialog:



Abbildung 32.1: Bestätigung des Endanwenderbenutzervertrags

- Klicken Sie auf *Ich stimme den Bedingungen des Lizenzvertrags zu*, um die Lizenzvereinbarung zu akzeptieren und klicken Sie anschließend auf *Weiter*, um zum nächsten Schritt zu gelangen.
- Klicken Sie auf *Installieren*, um mit der Installation zu beginnen. Der Installationsassistent richtet nun die erforderlichen Komponenten ein, die für die weitere Installation erforderlich sind.
- Klicken Sie anschließend auf *Weiter*. Wenn die System-Konfigurationsüberprüfung abgeschlossen wurde, sehen Sie die eigentlichen Installationsassistenten von SQL Server Express, etwa wie in der folgenden Grafik zu sehen.

³ Im Internet sind aber auch »Antwortdateien« für die stille Installation verfügbar, die SQL Server direkt im gemischten Modus einrichten.



Abbildung 32.2: Die Willkommensmeldung zur SQL Server 2005 Express-Installation

- Klicken Sie auf *Weiter*, um zur erweiterten Systemkonfigurationsüberprüfung zu gelangen. Auf Maschinen mit weniger als 512 MByte-Hauptspeicher sehen Sie möglicherweise eine *Mindestanforderung an die Hardware*-Warnung. Für kleinere Datenbankanwendungen reicht ein Hauptspeicher von 256 MByte (Windows 2000) bzw. 386 MByte (Windows XP, Windows 2003) aber aus, so dass Sie in diesem Fall die Warnung ignorieren können.⁴ Klicken Sie auf *Weiter*.



Abbildung 32.3: Entfernen Sie in diesem Dialog das Häkchen für den Zugriff auf erweiterte Konfigurationsoptionen

⁴ Das gilt nur, wenn keine anderen speicherintensiven Anwendungen auf diesem Rechner aktiv sind.

- Im nächsten Dialog, den Sie in Abbildung 32.3 sehen, geben Sie die gewünschten Daten ein. Wichtig: Achten Sie dabei darauf, das Häkchen *Erweiterte Konfigurationsoptionen ausblenden* zu entfernen, damit Sie während der weiteren Installation Zugriff auf die notwendigen erweiterten Installationsoptionen nehmen können.

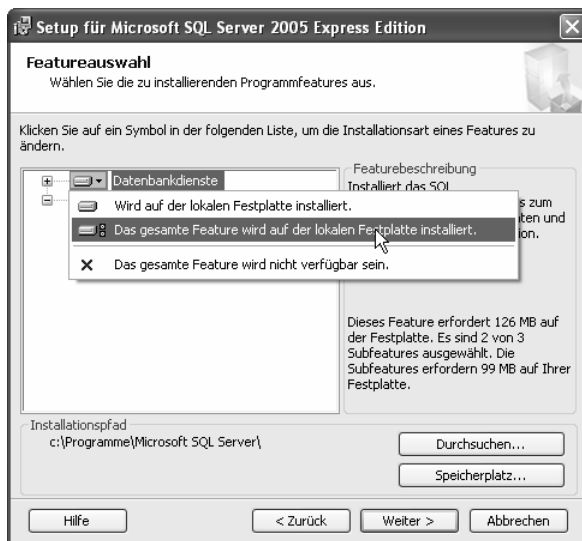


Abbildung 32.4: Bestimmen Sie in diesem Dialog den Installationsumfang

- Klicken Sie anschließend auf *Weiter*.
- Ändern Sie im folgenden Dialog den Installationsumfang, etwa wie in Abbildung 32.4 gezeigt. Verfahren Sie genauso für die Installation der Client-Komponenten.



Abbildung 32.5: In diesem Dialog bestimmen Sie, unter welchem Namen die SQL Server-Instanz später zu erreichen ist

- Klicken Sie anschließend auf *Weiter*.
- Bestimmen Sie im nächsten Dialog den Namen der SQL Server-Instanz (Abbildung 32.5), die Ihre Datenbanken später verwalten soll. Sie können die Voreinstellung *Benannte Instanz: SQLEXPRESS* belassen wie sie ist, wenn es nicht bereits eine SQL Server Instanz unter diesem Namen auf demselben Rechner gibt.

TIPP: Falls Sie sich mit einer »alten« Anwendung, die beispielsweise ODBC verwendet, gegen den SQL Server verbinden wollen, kann die Benutzung einer benannten Instanz Probleme bereiten. Benutzen Sie dann die Standardinstanz. Falls Sie aber ausschließlich mit .NET und ADO.NET arbeiten, bieten die benannten Instanzen eher mehr Möglichkeiten. So kann man alle Datenbanken zum Verkauf in der Instanz »Verkauf« unterbringen usw. Solche Instanzen arbeiten dann wie unabhängige SQL Server und können einzeln gestartet und gestoppt werden, sie verfügen über eine eigene Benutzerverwaltung, etc.

- Klicken Sie auf *Weiter*.

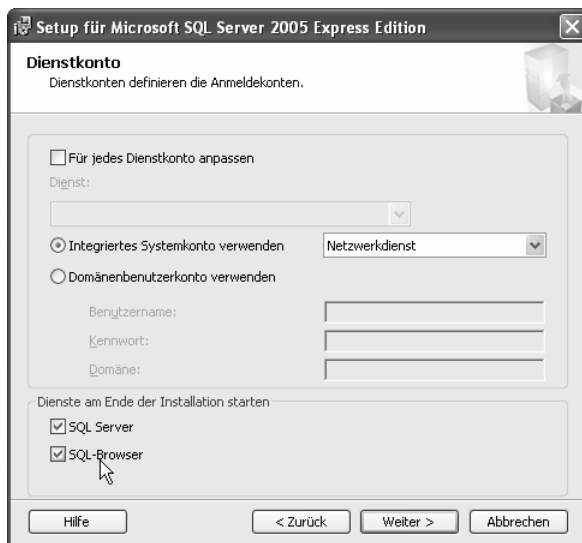


Abbildung 32.6: Bestimmen Sie die Aktivierung des SQL-Browser-Dienstes, damit diese SQL-Instanz später von anderen Computern des Netzwerkes aus per Instanznamen gefunden werden kann

- Damit Sie die SQL Server-Instanz später beispielsweise mit dem Verbindungsdialog des Server-Explorers von Visual Studio finden können, schalten Sie den SQL Browser-Dienst auf diesem Computer direkt mit ein, indem Sie das entsprechende Häkchen setzen.

TIPP: Für den professionellen Einsatz von SQL Server sollten Sie Folgendes beherzigen: Die Vorgabe »Netzwerkdienst« ist das Konto mit den geringsten Rechten. Damit wird es böswilligen Angreifern erheblich erschwert, über den SQL Server Zugriff auf ihr System zu erlangen. Der Netzwerkdienst ist daher dem ebenfalls verfügbaren Konto »Lokales System« auf jeden Fall vorzuziehen. Für die weitergehenden Möglichkeiten sollten Sie jedoch ein eigenes so genanntes »Dienstkonto« einrichten und hier im Setup bestimmen. Man könnte auf die Idee kommen, dass auch ein späterer Wechsel auf ein Dienstkonto z.B. über die Systemsteuerung und dem Eintrag »Dienste« möglich wäre. Dies ist auf den ersten Blick auch richtig; das Dienstkonto für den SQL Server benötigt aber für bestimmte Aufgaben sehr »spezielle« Rechte, wie zum Beispiel für das Ausführen von Festplattenwartungen (nur dann etwa funktioniert das neue Feature der schnellen Dateiinitalisierung).

Alle benötigten Rechte werden aber hier im Setup automatisch richtig vergeben: Sie sparen sich also viel Zeit und Mühe, wenn Sie hier direkt das gewünschte Konto als Dienstkonto eingeben. Falls Sie erwägen, solche Möglichkeiten wie Replikation (also den automatisierten Datenabgleich zwischen mehreren SQL Servern im Netz) zu benutzen, sollten Sie hier sogar ein Domänenkonto angeben.

- Klicken Sie anschließend auf *Weiter*.

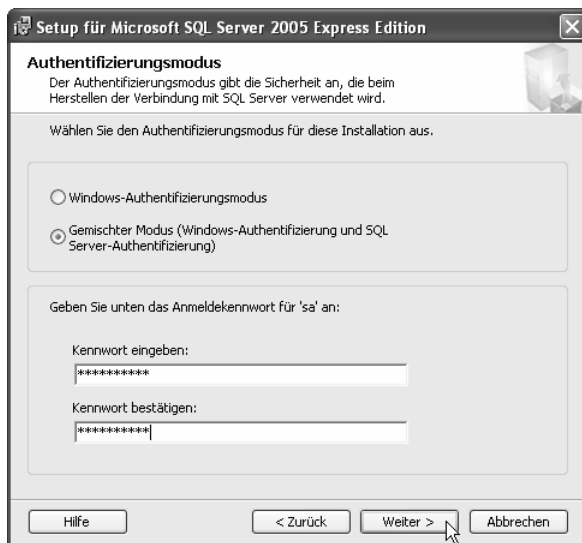


Abbildung 32.7: Verwenden Sie *Gemischter Modus*, wenn Sie die SQL Server-Instanz in einem Nicht-Domänen-Netzwerk von anderen Computern des Netzwerks erreichen wollen

- Im nächsten Dialog (Abbildung 32.7) bestimmen Sie den Authentifizierungsmodus, mit dem sich Clients an der Instanz und an von ihr verwalteten Datenbanken anmelden können. Wenn Ihre Anwendung ausschließlich auf der gleichen Maschine läuft, die auch die SQL Server-Instanz enthält oder SQL Server auf einer Maschine läuft, die Teil einer Active Directory Domäne ist, wählen Sie die sichere Authentifizierungsmethode *Windows-Authentifizierungsmodus*. Ist keine Domäne vorhanden, müssen Sie zur Anmeldung an einer SQL Server Instanz Anmeldeinformationen übergeben; das funktioniert nur, wenn die Instanz zuvor in den *gemischten Modus* geschaltet wurde. In diesem Fall bestimmen Sie ebenfalls ein Systemadministrator-Kennwort (»sa«), mit dem Sie sich später an der Instanz anmelden können.

HINWEIS: Ein sehr bekannter Virus (SQL Slammer) benutzte beim SQL Server 2000 die menschlich verständliche Schwäche, dass Benutzer damals zumeist KEIN Kennwort für das *sa*-Konto angaben; schon deshalb warnt das Setup hier deutlich. Vergeben Sie daher auf jeden Fall ein sicheres Kennwort (mind. 6 Zeichen, Groß- und Kleinschreibung, Sonderzeichen und Zahlen, kein Wort der deutschen Sprache, also beispielsweise im Stil von »P@a\$\$w0rd«)

- Klicken Sie anschließend auf *Weiter*.

Die Einstellungen der nächsten drei Dialoge können Sie so belassen, wie sie vorgegeben sind. Klicken Sie dreimal auf *Weiter*, und anschließend auf *Installieren*, um den eigentlichen Installationsvorgang zu starten und die Einrichtung abzuschließen.

Konfiguration der Netzwerkeinstellungen und Einrichten der Firewall unter Windows XP SP2

Damit eine SQL Server-Instanz im Netzwerk erreichbar ist, müssen Sie die Netzwerkprotokolle konfigurieren und ggf. einschalten. Sollten Sie SQL Server Express auf einem Windows XP-System mit Service Pack 2 betreiben und dieser Rechner nicht Teilnehmer einer Active Directory Domäne sein, müssen Sie obendrein die Firewall konfigurieren:

- Um die Netzwerkprotokolle einzurichten, starten Sie den SQL Configuration Manager aus dem Startmenü (*Start/Alle Programme/Microsoft SQL Server 2005/Konfigurationstools/SQL Server Configuration Manager*).
- Öffnen Sie in der linken Baumstruktur den Zweig *SQL Server 2005-Netzwerkkonfiguration*, und klicken Sie auf *Protokolle für 'SQLEXPRESS'*.

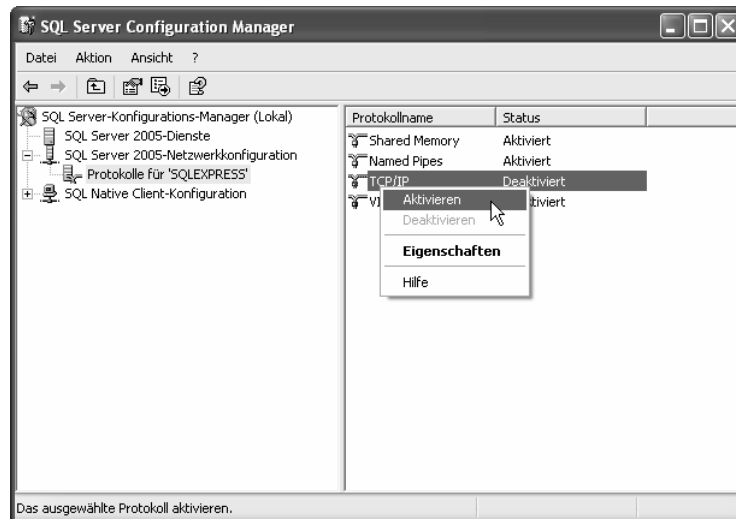


Abbildung 32.8: Aktivieren Sie in diesem Dialog die entsprechenden Protokolle, wenn Sie von anderen Rechnern des Netzwerkes aus auf diese SQL Server-Instanz zugreifen wollen

- Öffnen Sie in der rechten Liste das Kontextmenü über dem Eintrag *Named Pipes*. Wählen Sie aus dem Kontextmenü den Eintrag *Aktivieren*.
- Öffnen Sie das Kontextmenü in der rechten Liste über dem Eintrag *TCP/IP*. Wählen Sie aus dem Kontextmenü den Eintrag *Aktivieren*.

Firewall-Einstellungen unter Windows XP SP2

Falls sich SQL Server auf einem Windows XP-SP2-Rechner befindet, der nicht zu einer Active Directory-Domäne gehört, schalten Sie die entsprechenden Ports für den SQL Server- bzw. SQL Browser-Dienst frei. Dazu wählen Sie aus der Systemsteuerung *Windows Firewall*.



Abbildung 32.9: Öffnen Sie TCP-Port 1433 sowie UDP-Port 1434, um »von Außen« den SQL Server erreichen zu können. Öffnen Sie ebenfalls den Port für die Datei- und Druckerfreigabe, um über Named Pipes auf SQL Server zugreifen zu können (Port 445, bereits in der Ausnahmenliste vorhanden).

- Aktivieren Sie in der Ausnahmenliste die Datei- und Druckerfreigabe, da nur so die Ansteuerung über Named Pipes funktionieren wird (Port 445). Dieser Port ist standardmäßig in der Ausnahmenliste vorhanden.
- Klicken Sie auf der Registerkarte *Ausnahmen* auf *Port*, und geben Sie den TCP Port 1433 frei. Dieser Port wird standardmäßig für die erste SQL Server-Instanz vergeben.
- Öffnen Sie zusätzlich den UDP-Port 1434.

Grundsätzliches zu Datenbanken

Bevor Sie beginnen, mit Datenbanken zu programmieren, sollten Ihnen die Grundlagen zu diesem Thema bekannt sein. Bitte haben Sie dabei Verständnis dafür, dass ich an dieser Stelle aus Platzgründen nicht sehr ausführlich auf das Thema eingehen kann. Was die verschiedenen Datenbanksysteme anbelangt, so haben sie eines auf jeden Fall gemeinsam: Sie organisieren die zu verwaltenden Daten in Zeilen und Spalten verschiedener Tabellen. Die Spalten enthalten dabei verschiedene Datenfelder, wie etwa Name, Vorname, Personal-Nr. oder Postleitzahl und Ort bei einer Adressentabelle. Die einzelnen Zeilen entsprechen den eigentlichen Datensätzen. Am deutlichsten wird dieser Zusammenhang am konkreten Beispiel.

Aufbau der Beispieldatenbank

Die folgenden Beispiele verwenden eine Beispieldatenbank, die verschiedene Daten eines fiktiven Unternehmens abbildet. Dazu gehören:

- Berater, die im Unternehmen arbeiten, und mit der Bearbeitung bestimmter Projekte wie beispielsweise Softwareentwicklung oder dem Schreiben von Dokumentationen oder Büchern betraut sind.
- Projekte, die die Berater derzeit bearbeiten.
- Eine Zeittabelle, die darüber Auskunft gibt, welcher Mitarbeiter an welchem Projekt zu welcher Zeit gearbeitet hat.

Diese Daten wurden zufällig erzeugt. Dazu diente ein Programm, das nicht nur in der Lage ist, diese Daten zu generieren, sondern das Ihnen auch einen Einblick in die Struktur der Daten geben kann.

BEGLEITDATEIEN: Sie finden die Projektmappe für dieses Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\H - Database\AdoDemoSetup\AdoDemoSetup.sln`. Voraussetzung dafür ist, dass Sie von dem Computer, auf dem Sie dieses Beispiel ausprobieren möchten, einen Zugriff auf eine SQL Server 2005-Instanz haben.

Nach dem ersten Start fragt Sie das Programm nach einer SQL Server 2005-Instanz. Verwenden Sie SQL Server 2005 Express Edition auf Ihrem Entwicklungsrechner (was mit Visual Studio ab der Standard Edition automatisch mit installiert wird), dann klicken Sie einfach auf das entsprechende Kontrollkästchen, um die Standardinstanz zu verwenden, und beenden Sie den Dialog mit OK.

Die Demo-Datenbank wird anschließend automatisch erstellt, und Sie gelangen in einen Dialog, mit dem Sie die Möglichkeit haben, die Datenbank »mit Fleisch«, also mit fiktiven Daten zu füllen.

Anschließend (und bei jedem Neustart nach der ersten Einrichtung) bietet sich Ihnen ein Bild, wie Sie es auch in der folgenden Abbildung sehen können. Über das *Ansicht*-Menü haben Sie hier die Möglichkeit zu bestimmen, welche der drei Datentabellen, aus denen die Datenbank besteht, Sie in dem DataGridView-Steuerelement darstellen lassen möchten.

ID	Name	Vorname	Straße	Plz	Ort
1	Berater-Datentabelle	Gabriele	Kleine Gasse	19597	Stirpe
2	Projekte-Datentabelle	Momo	Absturzpfad	65980	Rheda
3	Zeiten-Datentabelle	José	Dorfgasse	95757	Rheda
4	Weichel	Daja	Reiterweg	01646	Lippetal
5	Schindler	Momo	Hauptstraße	55694	Rheda
6	Vielstedde	Guido	Thiemannweg	84029	Lippetal
7	Hoffmann	Franz	Kurgartenweg	63790	Dortmund
8	Jungemann	José	Ausfallstraße	42504	Bad Waldliesborn
9	Hollmann	Guido	Ausfallstraße	31502	Straubing
10	Trouw	Alfred	Gassenstraße	85114	Straubing
11	Weichel	Rainer	Nording	77374	Liebenburg
12	Weichel	Melanie	Nording	68405	Bielefeld
13	Weichelt	Jürgen	Am Tor	72963	Lippetal
14	Müller	Britta	Stadtstraße	84824	Wiesbaden
15	Neumann	Anne	Thiemannweg	16694	Bielefeld
16	Thiemann	Lothar	Lange Straße	75343	Lippetal

Abbildung 32.10: Dieses Programm ist nicht nur in der Lage die Demodatenbank zu generieren, sondern stellt die Daten der einzelnen Tabellen auch dar

Anhand dieser Abbildung können Sie genau sehen, wie die Datentabelle zeilen- und spaltenweise organisiert ist. Die einzelnen Datenzeilen werden übrigens auch *Datensätze* genannt (eine einzige Datenzeile ist ein *Datensatz*).

TIPP: Falls Sie im Laufe der Zeit zu viel mit der Datenbank herumgespielt und sie damit sehr in Unordnung gebracht haben, wählen Sie aus dem Menü *Datei* den Menübefehl *Vorhandene Daten löschen und neu erstellen*.

Klärung grundsätzlicher Begriffe

Für die verschiedenen Elemente, mit denen Sie bei der Programmierung von Datenbanken arbeiten, gibt es spezielle Begriffe, die Sie sich einprägen sollten:

Verbindungen zur Datenbank über Connection-Objekte

Bevor Sie auch nur ein einziges Datum aus einer Datenbank herauskitzeln oder der Datenbank-Engine mitteilen können, dass sie selbst irgendwelche Operationen durchführen soll, müssen Sie eine *Verbindung* zur Datenbank herstellen. In ADO.NET verwenden Sie dafür das *Connection*-Objekt, um eine Verbindung zu einem *Daten-Provider* (etwa: *Datenanbieter*) aufzubauen (wie beispielsweise Microsoft Access oder SQL Server 2005). Es gibt die verschiedensten Datenbankprovider, und bei Daten-Provider abhängigen Objekten stehen die eigentlichen Provider bei der Namensgebung der Klassen Pate. So verwenden wir für alle folgenden Beispiele ausschließlich SQL Server als Datenprovider – die *Connection*-Klasse für diesen Provider nennt sich deshalb *SqlConnection*. Griffen Sie über *OleDb* beispielsweise auf eine Access-Datenbank zu, würden Sie dazu die *OleDbConnection*-Klasse verwenden.

Befehle an die Datenbank mit dem Command-Objekt übermitteln

Nachdem Sie eine Verbindung zur Datenbank hergestellt haben, können Sie über *Command*-Objekte Befehle an die Datenbank senden. Diese Befehle dienen entweder dazu, Daten bestimmter Tabellen (die auch über SQL-Befehle miteinander verknüpft werden können) abzufragen oder andere Befehle – zum Beispiel zum Löschen von Datensätzen – an die Datenbank abzusetzen. Im Falle von SQL Server als Daten-Provider nennt sich das entsprechende *Command*-Objekt *SqlCommand*.

TIPP: *SqlCommand*-Objekte verwenden Sie ebenfalls, um gespeicherte Prozeduren auf einer SQL Server-Datenbank zu initiieren.

Ermitteln von Resultsets durch SQL-Abfragen

Wenn über *Command*-Objekte bestimmte Befehle an die Datenbank übermittelt werden, die Daten zurückliefern, dann spricht man vom Zurückerhalten eines *Resultsets* – eines Ergebnissatzes (an Daten). Resultsets können beispielsweise mit einem *DataReader*-Objekt (*SqlDataReader* im Falle von SQL Server) direkt gelesen oder in ein *DataSet*-bzw. *DataTable*-Objekt übertragen werden.

Prinzipielle Vorgehensweise beim Abfragen und Modifizieren von Daten

ADO.NET kennt zwei grundsätzliche Arbeitsmodi: den unverbundenen und den verbundenen. Beim verbundenen Modus trudeln die Daten nacheinander ein. Sie können Daten dabei nicht verändern, sondern nur lesen. Sie können die einzelnen Datensätze bei dieser Vorgehensweise nur der Abfrage entsprechend nacheinander einlesen; ein Sprung zurück oder ein Überspringen von Datensätzen ist dabei nicht möglich.

Der unverbundene Modus ist die Arbeitsweise, mit der Sie Daten nicht nur stringent auslesen, sondern frei editieren können. Bevor Sie im unverbundenen Modus arbeiten, wird natürlich eine Verbindung zur Datenbank hergestellt. Nach einer erfolgten Abfrage verwenden Sie beispielsweise ein `DataTable`-Objekt, um die abgefragten Daten komplett dort hinein zu laden. Danach stehen die Daten völlig getrennt von der eigentlichen Datenbank im Speicher Ihres Computers und können dort verarbeitet werden. Ab jetzt gibt es keine Verbindung mehr zur Datenbank. Erst wenn Sie die Verarbeitung abgeschlossen haben, schicken Sie sie – erst jetzt sind Sie wieder mit der Datenbank verbunden – zur Datenbank zurück.

Einsehen von Daten mit dem Server-Explorer

Das beste Werkzeug zum Einsehen von Daten oder Tabellenstrukturen in einer Datenbank sind die Werkzeuge selbst, die die Datenbanken für diesen Zweck anbieten. Doch nicht immer stehen diese Tools auf dem Rechner zur Verfügung, auf dem man entwickelt. Für diese Fälle stellt Visual Studio (ab der Standard-Edition) den *Server-Explorer* zur Verfügung, der in Visual Studio 2005 die datenbankeigenen Tools eigentlich überflüssig macht.

Den Server-Explorer finden Sie unterhalb der Toolbox, wenn Sie die Standardeinstellung nicht geändert haben. Falls er nicht von vorne herein angezeigt wird, schalten Sie ihn einfach ein, indem Sie aus dem Menü *Ansicht* den Eintrag *Server-Explorer* auswählen.

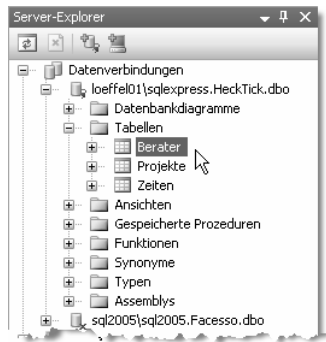


Abbildung 32.11: Mit dem Server-Explorer können Sie auf einfache Weise Einsicht in Daten und Tabellenstrukturen nehmen

Um eine Verbindung wie in unserem Beispiel zur lokalen SQL Server-Express-Demodatenbank aufzubauen, verfahren Sie folgendermaßen:

- Im Server-Explorer klicken Sie mit der rechten Maustaste auf *Datenverbindungen*, und aus dem Kontextmenü, das jetzt erscheint, wählen Sie den Eintrag *Verbindung hinzufügen*.



Abbildung 32.12: So bauen Sie eine Verbindung zu einer Datenbank auf der lokalen SQL Server Express-Instanz auf und testen die Verbindung

- Im Dialog, der jetzt erscheint (siehe Abbildung 32.12), überprüfen Sie die Datenquelle, die *Microsoft SQL Server (SqlClient)* lauten muss. Falls das nicht der Fall ist, klicken Sie auf *Ändern*.



Abbildung 32.13: Wählen Sie aus diesem Dialog die Datenquelle aus, die Sie verwenden möchten

- Wählen Sie aus der Liste *Microsoft SQL Server* aus. Wählen Sie *.NET Framework-Datenanbieter für SQL Server* aus der Aufklappliste *Datenanbieter*, und bestätigen Sie den Dialog mit *OK*.
- Geben Sie – für den Fall, dass Sie auf die lokale SQL Server-Express-Instanz zugreifen wollen – als Servernamen *.\SQLEXPRESS* ein. Wählen Sie in diesem Fall *Windows-Authentifizierung verwenden*, um ihr Windows-Benutzerkonto auch für die Anmeldung am SQL Server zu verwenden.
- Im Bereich *Mit Datenbank verbinden*, klappen Sie die Aufklappliste der Datenbanknamen aus, und wählen damit die zu bearbeitende Datenbank – in unserem Beispiel die Datenbank *HeckTick*.

Anekdoten aus der Praxis

Ein lieber Kollege von mir – Jürgen *Heckhuis*, und achten Sie auf seinen Nachnamen – seines Zeichens begnadeter Netzwerktechniker, kam eines Tages zu mir, und bat mich, ich möge ihm doch ein Beispiel für Visual Basic 2005 zum Üben geben, er möchte diese Programmiersprache nämlich erlernen. Gemeinsam bestimmten wir ein Zeiterfassungs- und Auswertungsprogramm als bestes Beispiel für viele der zu verwendenden Techniken in Visual Basic 2005 aus. Nachdem er sich einige Zeit – so glaubte ich jedenfalls – mit den ersten Startproblemen beschäftigt hatte, tönnte es »Heureka!« aus seinem Büro. Ich eilte sofort dort hin und fragte ihn, ob er die erste Version bereits fertig habe. »Nein, nein«, entgegnete er, »ich hab noch gar nichts gemacht. Aber ich habe einen genialen Namen für das Programm: *HeckTick!!!*«

Dieser Kalauer blieb so sehr hängen, dass ich ihn für dieses Beispiel verwenden musste – nur für den Fall, dass Sie sich wundern ...

- Abschließend können Sie die Verbindung zur Datenbank mit der gleichnamigen Schaltfläche testen. Ansonsten klicken Sie auf *OK*, um den Vorgang abzuschließen.

Die Verbindung zur Datenbank wurde nun zur Liste der Datenverbindungen im Server-Explorer hinzugefügt.

Berater: Abfra...press.Hecktick)		Startseite	HeckTickDataSet.Designer.vb	frmDemoDaten.vb	frmMain.vb	fr
IDBerater	Nachname	Vorname	Straße	Plz	Ort	
1	Weichel	Daja	Aue	90562	Wuppertal	
2	Sonntag	José	Nordring	25019	Wuppertal	
3	Meier	Uwe	Stadtstraße	36878	Liegenburg	
4	Weichel	Axel	Crash Ave	93063	Lippetal	
5	Plenge	Barbara	Stadtstraße	12073	Lippetal	
6	Heckhuis	Momo	Kleine Gasse	81377	Lippetal	
7	Westermann	Melanie	Am Brunnen	52420	Bielefeld	
▶	Englisch	Barbara	Dorfgasse	44852	Unterschleißheim	
9	Plenge	Anne	Crash Ave	48132	Straubing	
10	Ademmer	Barbara	Kleine Gasse	06572	Wiesbaden	
11	Tiemann	Daja	Reiterweg	48938	Soest	
12	Braun	Alfred	Parkstraße	01878	Wuppertal	
13	Hörstmann	Thomas	Crash Ave	36519	Wiesbaden	
14	Vielstedde	Anne	Main Road	78164	Liegenburg	
15	Braun	Hans	Alter Postweg	23034	Stirpe	
16	Rode	Lothar	Am Tor	31792	Braunschweig	
17	Weigel	Guido	Gassenstraße	05897	Unterschleißheim	
18	Weigel	José	Windpockenallee	33201	Unterschleißheim	
19	Westermann	Katrin	Dorfplatz	43673	Lippetal	
20	Thiemann	José	Windpockenallee	76122	München	
*	NULL	NULL	NULL	NULL	NULL	

Abbildung 32.14: Ein Doppelklick auf eine Tabelle macht Daten und Struktur sichtbar

Sie können anschließend den nun vorhandenen Zweig unter Datenverbindungen öffnen. Die Verbindung zur gerade ausgewählten SQL Server-Datenbank ist jetzt dort zu sehen. Öffnen Sie auch die nächsten Zweige, werden die Tabellen sichtbar. Ein Doppelklick auf eine Tabelle zeigt Ihnen die Tabellenstruktur und die Daten an, etwa wie in Abbildung 32.14 zu sehen.

Sie können die Daten in dieser Ansicht ändern und auch neue Datensätze hinzufügen. Über die Kontextmenüs der einzelnen Elemente der Baumstruktur des Server-Explorers haben Sie ferner die Mög-

lichkeit, weitere Funktionen abzurufen, wie beispielsweise das Erstellen neuer Tabellen, das Verändern von Tabellenstrukturen, das Erstellen von gespeicherten Prozeduren und vieles mehr.

Wenn Sie die Datenvorschau nicht mehr benötigen, schließen Sie das Fenster einfach.

Programmieren mit ADO.NET

Soviel zur theoretischen Vorbereitung. In diesem Abschnitt erfahren Sie nun Grundlegendes darüber, wie Sie die verschiedenen Objekte, die ADO.NET zur Verfügung stellt, nutzen können, um in eigenen Programmen Datenabfragen durchzuführen und Daten in Tabellen zu verändern und zu ergänzen.

Verbindungen zur Datenbank mit der Connection-Klasse herstellen und Befehle mit der Command-Klasse ausführen

Um eine Verbindung zu einer Datenbank herzustellen, benötigen Sie ein *Connection*-Objekt. Um anschließend einen Befehl an die Datenbank (oder SQL Server-Instanz) zu senden, benötigen Sie ein *Command*-Objekt, das diese Befehle kapselt. Im Falle von SQL Server nennen sich diese beiden Klassen *SqlConnection* und *SqlCommand*. Die grundsätzliche Vorgehensweise, um mit der Datenbank in Kommunikation zu treten, lautet folgendermaßen (die relevanten Teile sind fett hervorgehoben):

'Wichtig: In diesem Namespace befinden sich die SQL Server ADO-Klassen
Imports System.Data.SqlClient

Module mdlMain

```
Sub Main()  
    Dim locConnection As SqlConnection  
    Dim locCommand As SqlCommand  
    Dim locDataReader As SqlDataReader  
  
    'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann  
    locConnection = New SqlConnection _  
        ("Data Source=.\SQLExpress;Initial Catalog=Hecktick;Integrated Security=True")  
  
    'Würden Sie den "gemischten Modus" verwenden, wäre Folgendes die richtige Wahl.  
    'Aber aufgepasst: Das Passwort steht dann im Quellcode und kann leicht gefunden werden!  
    'Eine Verschlüsselung des Passwortes wenigstens mit einfachen Mitteln wäre dann angezeigt.  
    locConnection = New SqlConnection _  
    ('      ("Data Source=.\SQLExpress;Initial Catalog=Hecktick;User ID=sa; Password=Irgendwas")  
  
    'Wichtig: Verbindung muss geöffnet sein  
    locConnection.Open()  
  
    'Command-Objekt erstellen, mit der ein Befehl an die Datenbank abgesetzt  
    'und ein ResultSet eingeholt werden kann  
    locCommand = New SqlCommand("SQLBefehle", locConnection)  
  
    'Mit Using sorgen wir dafür, dass die Verbindung wieder geschlossen wird,  
    'wenn locConnection aus dem Using-Scope herausläuft.  
    Using locConnection
```

```

        'Command-Objekt einweisen, dass es den SQL-Befehl ausführt
        locDataReader = locCommand.ExecuteReader()
    End Using
End Sub
End Module

```

In diesem Beispiel würde das verwendete SqlCommand-Objekt natürlich zu einer Ausnahme führen, da es keinen gültigen Befehl enthält. Dieser kleine Codeabschnitt soll aber auch lediglich demonstrieren, wie Sie grundsätzlich vorgehen, um eine Verbindung zur Datenbank herzustellen und entsprechende Befehle abzusetzen oder Daten durch Select-Abfragen anzufordern.

Der nächste Abschnitt zeigt das am konkreten Beispiel.

Datenverbindungen herstellen und Resultsets mit der DataReader-Klasse auslesen

Das nachfolgend beschriebene Projekt macht nichts weiter, als mit dem SqlConnection-Objekt eine Verbindung zur Demo-Datenbank aufzubauen. Anschließend verwendet es ein SqlCommand-Objekt, um eine SELECT-Anweisung an die Datenbank abzusetzen und das von der Datenbank zurück gelieferte *Resultset* mit dem SqlDataReader auszulesen. Die Beispielanwendung ist der Einfachheit halber als Konsolenanwendung konzipiert.

BEGLEITDATEIEN: Sie finden dieses Beispiel im Verzeichnis `.\\VB 2005 - Entwicklerbuch\\G - SmartClient\\Kap32\\DataReader\\`.

```

'Wichtig: In diesem Namespace befinden sich die SQL Server ADO-Klassen
Imports System.Data.SqlClient

Module mdlMain

    Sub Main()
        Dim locConnection As SqlConnection
        Dim locCommand As SqlCommand
        Dim locDataReader As SqlDataReader

        'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
        locConnection = New _
            SqlConnection("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")

        'Würden Sie den "gemischten Modus" verwenden, wäre Folgendes die richtige Wahl.
        'Aber aufgepasst: Das Passwort steht dann im Quellcode und kann leicht gefunden werden!
        'Eine Verschlüsselung des Passwortes wenigstens mit einfachen Mitteln wäre dann angezeigt.
        'locConnection =
        'New SqlConnection("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;User ID=sa; Password=Irgendwas")

        'Command-Objekt erstellen, mit der ein Befehl an die Datenbank abgesetzt
        'und ein ResultSet eingeholt werden kann
        locCommand = New SqlCommand("SELECT TOP 15 * FROM Berater ORDER BY [Nachname]", locConnection)

        'Wichtig für den DataReader: Verbindung muss geöffnet sein
        locConnection.Open()
    End Sub
End Module

```

```

'Damit sorgen wir dafür, dass die Verbindung wieder geschlossen wird,
'wenn locConnection aus dem Using-Scope herausläuft.
Using locConnection
Console.WriteLine("Inhalt der Tabelle Mitarbeiter (erste 15. Datensätze, nach Nachnamen sortiert)")

'Command-Objekt einweisen, dass er ein Reader-Objekt mit Datenzugriff erstellt
locDataReader = locCommand.ExecuteReader()

'Feststellen, ob überhaupt Daten vorhanden sind:
If locDataReader.HasRows Then
'Read holt den jeweils nächsten Datensatz und wird False, wenn es keinen weiteren gibt
Do While locDataReader.Read
'Alle Datenspalten durchlaufen
For c As Integer = 0 To locDataReader.FieldCount - 1
'GetValue holt das eigentliche Datum als Object
Console.Write(locDataReader.GetValue(c).ToString + vbTab)
Next
'Datensatz komplett dargestellt, Zeilenwechsel
Console.WriteLine()
Loop
'Auf Tastatureingabe warten
Console.ReadLine()
End If
End Using
End Sub

End Module

```

Um an die Daten heranzukommen, muss natürlich als erstes eine Verbindung zur SQL Server-Instanz hergestellt werden – dabei verwendet das Beispiel das Muster aus dem vorherigen Abschnitt.

Durch die ExecuteReader-Methode des SqlCommand-Objektes wird der im SqlCommand-Objekt gekapselte Befehl einerseits an den Server geschickt, andererseits erstellt diese Methode ein *DataReader*-Objekt vom Typ SqlDataReader, mit dem Sie anschließend Zugriff auf die Daten nehmen können, so die SQL-Datenbank tatsächlich Daten zurückgeliefert hat (was übrigens, wie im Beispiel zu sehen, mit der Eigenschaft HasRows des SqlDataReader-Objekts festgestellt werden kann).

HINWEIS: Ein *DataReader*-Objekt kann nicht direkt instanziiert werden; Sie müssen es von einem Command-Objekt erstellen lassen, um es – wie im Beispiel zu sehen – anschließend verwenden zu können.

Die Read-Funktion des DataReader-Objektes sorgt dann dafür, dass der jeweils nächste Datensatz aus der Datenbank gelesen wird. Wichtig: Die Daten werden dabei tatsächlich zeilenweise von der Datenbank gelesen – mit jedem Aufruf der Read-Methode werden also die nächsten Daten von der Datenbank übertragen und können anschließend mit den GetXXX-Funktionen des *DataReader* abgerufen werden. Die Read-Methode liefert als Funktionsergebnis False zurück, wenn keine weiteren Datensätze mehr im Resultset zur Verfügung stehen. Beim Öffnen des DataReader steht der interne Zeiger also VOR dem ersten Datensatz, die erste Read-Anweisung verschiebt das Lesen auf die erste Zeile. Daher ist auch eine Prüfung, ob überhaupt Daten zurückgegeben wurden sehr einfach. Sollte Ihrer Abfrage gar keine Daten entsprechen, würde die erste Read-Anweisung schon False zurückgeben. Diese Vorgehensweise macht übrigens grundsätzlich dann Sinn, wenn Sie Daten lediglich auslesen müssen und dabei möglichst wenig Datenverkehr produzieren wollen.

Wenn Sie dieses Programm starten, zeigt es in etwa das folgende Ergebnis auf dem Bildschirm an:

Inhalt	Tabelle	Berater	(ersten 15 Datensätze, nach Nachnamen sortiert)		
10	Ademmer	Barbara	Kleine Gasse	06572	Wiesbaden
15	Braun	Hans	Alter Postweg	23034	Stirpe
12	Braun	Alfred	Parkstraße	01878	Wuppertal
8	Englisch	Barbara	Dorfgasse	44852	Unterschleißheim
6	Heckhuis	Momo	Kleine Gasse	81377	Lippetal
13	Hörstmann	Thomas	Crash Ave	36519	Wiesbaden
3	Meier	Uwe	Stadtstraße	36878	Liebenburg
9	Plenge	Anne	Crash Ave	48132	Straubing
5	Plenge	Barbara	Stadtstraße	12073	Lippetal
16	Rode	Lothar	Am Tor	31792	Braunschweig
2	Sonntag	José	Nordring	25019	Wuppertal
20	Thiemann	José	Windpockenallee	76122	München
11	Tiemann	Daja	Reiterweg	48938	Soest
14	Vielstedde	Anne	Main Road	78164	Liebenburg
4	Weichel	Axel	Crash Ave	93063	Lippetal

TIPP: In den Beispielen dieses Kapitels wird das `SqlConnection`-Objekt mit `Using` benutzt. Daher wird automatisch `Dispose` für das `SqlConnection`-Objekt aufgerufen, und diese Vorgehensweise für die meisten Datenbankanwendungen auch okay. In einem Datenbankprojekt, das wirklich performant sein muss, bietet es sich unter Umständen an, die Verbindung selber zu pflegen: Sie muss geöffnet werden, bevor Sie mit einem `DataReader` Daten lesen können und sollte auch nach dem Beenden des Lesevorgangs wieder geschlossen werden (aber das Verbindungsobjekt wird dabei nicht mit `Dispose` zum Entsorgen freigegeben).

Die `ExecuteReader` Methode unterstützt eine Überladung, bei der Sie beispielsweise angeben können, dass mit dem Schließen des `DataReader` auch die benutzte Verbindung geschlossen wird. Das Öffnen und Schließen einer Verbindung ist vergleichsweise »teuer«, was die Benutzung von Ressourcen auch auf dem SQL Server angeht. ADO.NET »poolt« daher standardmäßig Verbindungen, d.h. wenn Sie im Programm die Verbindung mit `Close` schließen, wird diese nicht endgültig geschlossen, sondern in den Pool zurückgestellt und beim nächsten `Open` wieder verwendet – was übrigens nur dann gilt, wenn die Verbindungszeichenfolge exakt (!) der vorherigen entsprochen hat. Dennoch wirkt es sich auf die Performance positiv aus, das Objekt nicht mit `Dispose` (oder durch die Verwendung mit `Using`) zu entsorgen, obgleich dieser Performancegewinn bei »normalen« Datenbankanwendungen nur marginal ist.

Unverbundene Daten mit dem `DataTable`-Objekt verwalten

Wenn Sie Daten mit dem `DataReader` aus einer Datenbanktabelle lesen, hat das eigentlich nur einen Vorteil: Sie erhalten die Daten paketweise, verursachen damit wenig Datenverkehr und belasten den Hauptspeicher nur gering. Der große Nachteil: Daten sind mit dem `DataReader` äußerst umständlich zu handhaben, das haben Sie in dem kleinen Beispiel schon feststellen dürfen.

Mit dem `DataTable`-Objekt haben Sie es da schon viel einfacher. Es ist der erste Schritt zum Arbeiten mit nicht verbundenen Daten. Sie stellen sich das `DataTable`-Objekt am besten als eine Auflistung mit Elementen vor, deren Datenstruktur von den gespeicherten Daten der Datenbanktabelle abhängt (oder besser: des Resultsets, denn durch `SELECT`-Abfragen können auch Kombinationen mehrerer Tabellen oder nicht alle Felder einer Tabelle zurückgegeben werden). Die einzelnen Elemente einer `DataTable` können, wie bei anderen Auflistungen auch, durch die `Items`-Eigenschaft ihrer `Rows`-Auflistung angesprochen werden, die die eigentlichen Daten als Auflistung aus so genannten `DataRow`-

Objekten erhält. Ein einzelnes Item ist also grundsätzlich vom Typ DataRow. Ein DataRow-Objekt erlaubt dann schließlich den Zugriff auf die eigentlichen Daten.

Die DataAdapter-Klasse

Um ein DataTable-Objekt mit Inhalt zu füllen, benötigen Sie eine besondere Schnittstelle zur Datenbank, den so genannten *DataAdapter*.⁵ Ein einzelnes *Command*-Objekt ist hier nicht mehr ausreichend, da es nur einen *DataReader* erzeugen kann, um die Schemainformationen einer Tabelle zu ermitteln und die Daten aufgrund der gegebenen Abfrage einzulesen. Doch Daten müssen gegebenenfalls auch aktualisiert oder neue Daten in die Datenbank eingefügt werden können. Und dafür sind mehrere Kommandos erforderlich, die der *DataAdapter* alle unter einem Hut vereint – wie das genau funktioniert, dazu später mehr.

Betrachten wir die Vorgehensweise, die ADO.NET verwendet, wenn Daten einer SQL Server-Tabelle mithilfe von *SqlDataAdapter* und *DataTable* abgerufen werden sollen:

- Das *SqlDataAdapter*-Objekt öffnet die Verbindung zur Datenbank im Bedarfsfall.
- Es weist das *SqlCommand*-Objekt an, das es aus seiner *SelectCommand*-Eigenschaft ermittelt, die Schemainformationen des Resultsets einzuholen, das zuvor beispielsweise durch eine *SELECT*-Anweisung abgefragt wurde.
- Es baut aus diesen Schemainformationen die Grundstruktur eines neuen *DataTable*-Objekt auf.
- Es verwendet anschließend ein *SqlDataReader*-Objekt, das durch das *SqlCommand*-Objekt angelegt wurde, um das *DataTable*-Objekt mit Daten zu füllen.
- Es schließt die Verbindung zur Datenbank.

Das Ergebnis: Die Daten stehen anschließend – losgelöst von der eigentlichen Datenbank – im Speicher des Computers, und sie sind durch die *Rows*-Eigenschaft (die die *DataRow*-Auflistung zur Verfügung stellt) des *DataTable*-Objektes abrufbar.

Diese Vorgehensweise wird durch das folgende Beispiel demonstriert, das eine Erweiterung des vorherigen Beispielprojektes darstellt. Mit diesem Programm können Sie also nicht nur die Struktur der Datenbank oder ihrer Tabellen einsehen, sondern auch Zugriff auf die gespeicherten Daten selbst nehmen.

Da Sie im Programm die *SELECT*-Anweisung frei verändern können, bietet es sich darüber hinaus auch zum Herumexperimentieren mit verschiedenen SQL-Abfragen an.

BEGLEITDATEIEN: Sie finden dieses Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap32\Data-TableDemo\`.

⁵ Und wenn Sie jetzt genau hinschauen, stellen Sie fest, dass *DataTable* und *DataAdapter* auf unterschiedliche Weise hier im Text formatiert sind. *DataTable* ist datenbankunabhängig, deswegen ist es keine Klassenkategorie sondern eine konkrete Klasse und in Listingschrift formatiert. *DataAdapter* hingegen ist wie *Connection* und *DataReader* Daten-Provider abhängig – im Falle von SQL Server heißt die entsprechende Klasse *SqlDataAdapter* und ist deswegen nicht in Listingschrift sondern kursiv formatiert.

Wenn Sie dieses Beispielprojekt laden und starten, erlaubt es Ihnen, eine Abfrage zu formulieren, und das Ergebnis dieser Abfrage in Tabellenform im darunter liegenden DataGridView-Steuerelement sichtbar zu machen (siehe Abbildung 32.15).

Ein Blick auf das Listing offenbart, wie einfach der Einsatz mit dem DataTable-Objekt im Grunde genommen ist:

```
Imports System.Data.SqlClient

Public Class Form1

    Private Sub btnAusführen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnAusführen.Click
        Dim locConnection As SqlConnection

        'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
        locConnection = New SqlConnection _
            ("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")

        'Würden Sie den "gemischten Modus" verwenden, wäre Folgendes die richtige Wahl.
        'Aber aufgepasst: Das Passwort steht dann im Quellcode und kann leicht gefunden werden!
        'Eine Verschlüsselung des Passwortes wenigstens mit einfachen Mitteln wäre dann angezeigt.
        'locConnection = New SqlConnection _
            ("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;User ID=sa; Password=Irgendwas")

        Dim locAdapter As New SqlDataAdapter(txtSelectString.Text, locConnection)
        Dim locTable As New DataTable
        Try
            locAdapter.Fill(locTable)
        Catch ex As Exception
            MessageBox.Show(ex.Message, "Fehler bei der Befehlsausführung:", _
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        End Try

        'Zur einfachen Darstellung der Daten reicht es auch ohne Hilfe einer
        'BindingSource aus, die DataTable der DataSource-Eigenschaft der DataGridView
        'zuzuweisen. In diesem Fall wird intern ein CurrencyManager-Objekt
        'für die Datenbindung anstelle der BindingSource eingerichtet.
        dgvData.DataSource = locTable
    End Sub
End Class
```

Die entscheidenden Zeilen in diesem Listing sind die fett hervorgehobenen. Beim Erstellen eines neuen SqlDataAdapter-Objektes übergeben Sie seinem Konstruktor sowohl die SELECT-Zeichenfolge als auch die Verbindung, die das Objekt verwenden soll, um einerseits den SELECT-Befehl abzusetzen und andererseits die Daten für die spätere Aufbereitung im DataTable-Objekt zu ermitteln. Sie erstellen anschließend ein neues DataTable-Objekt, und weisen den *DataAdapter* mit seiner *Fill*-Methode an, das DataTable-Objekt mit den abgefragten Daten zu füllen.

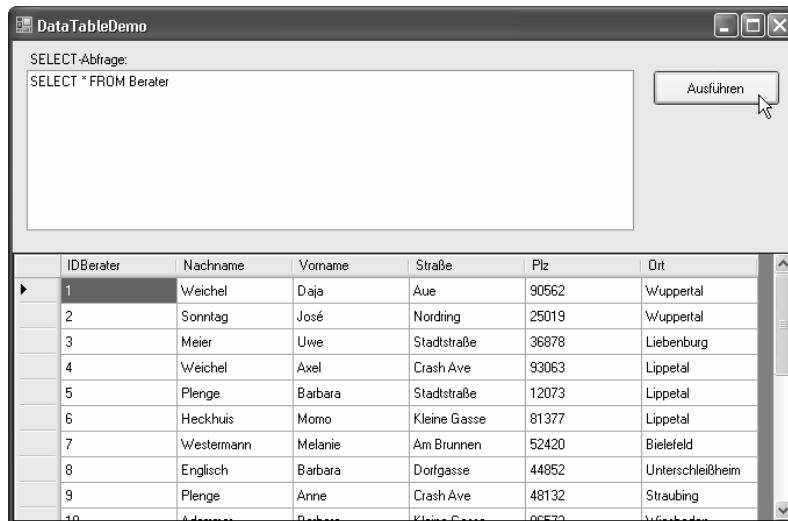


Abbildung 32.15: Mit diesem Beispiel, das den Einsatz der *DataTable*-Klasse demonstriert, können Sie beliebige Abfragen an die Beispieldatenbank stellen und die Resultsets in Tabellenform sichtbar machen

Zugriff auf die einzelnen Datenfeldinhalte über das *DataRow*-Objekt

Um an die einzelnen Inhalte einer Datenzeile programmtechnisch zu gelangen (und diese nicht nur durch Zuweisung an die *DataSource*-Eigenschaft in einem *DataGridView*-Steuerelement darstellen zu lassen), bedienen Sie sich wie schon erwähnt der *DataRow*-Objekte einer *DataTable*, die Sie durch die *Rows*-Eigenschaft ermitteln können.

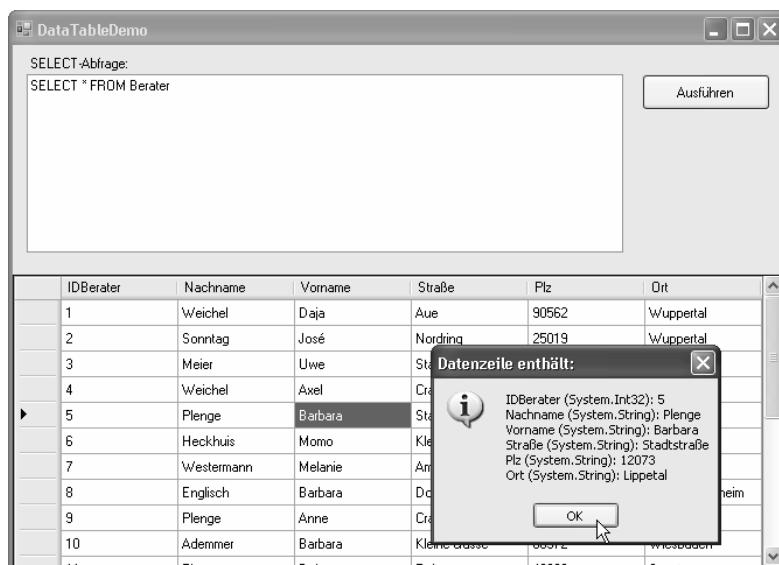


Abbildung 32.16: Per Doppelklick auf eine Tabellenzelle können Sie sich die Details eines Datensatzes in einem Nachrichtefeld anzeigen lassen

Im Beispielprojekt haben Sie die Möglichkeit, sich durch Doppelklick auf eine Tabellenzelle die entsprechende Zeile en détail in einem Meldungsfeld ausgeben zu lassen.

Der entsprechende Code, der das erledigt, sieht folgendermaßen aus:

```
'Wird ausgelöst, wenn der Anwender auf eine Zelle doppelt klickt.
Private Sub dgvData_CellMouseDoubleClick(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) Handles dgvData.CellMouseDoubleClick

    'Die Datentabelle existiert noch, wir haben sie schließlich der
    'DataSource der DataGridView zugeordnet.
    Dim locDataTable As DataTable = DirectCast(dgvData.DataSource, DataTable)

    'Jetzt ermitteln wir die entsprechende DataRow der DataTable
    'Die folgende Methode funktioniert auch, wenn die Tabelle sortiert wurde,
    'da jede Zeile der DataGridView das "Original"-Objekt enthält,
    'aus der sie entstanden ist. In diesem Fall ist das ein DataRowView-Objekt,
    'mit dem die ursprüngliche Zeile der DataTable abrufbar ist.
    Dim locO As Object = dgvData.Rows(e.RowIndex)
    Dim locDataRowView As DataRowView = TryCast(dgvData.Rows(e.RowIndex).DataBoundItem, DataRowView)

    'War kein DataRowView-Objekt, dann beenden.
    If locDataRowView Is Nothing Then
        Return
    End If

    'Daraus können wir jetzt die eigentliche DataRow ableiten
    Dim locDataRow As DataRow = locDataRowView.Row

    'Und die Daten, die sich in der DataRow befinden,
    'geben wir anschließend in einer MessageBox aus:
    'Das StringBuilder-Objekt verwenden wir zum Zusammenbauen des Strings.
    Dim locSb As New System.Text.StringBuilder

    'Durch die einzelnen Spalten der Tabelle iterieren:
    For Each locSpalte As DataColumn In locDataRow.Table.Columns
        'String zusammenbauen. Erst den Spaltennamen,
        locSb.Append(locSpalte.ColumnName)
        'dann den Spaltentyp in Klammern,
        locSb.Append(" (" & locSpalte.DataType.ToString & "): ")
        'dann den eigentlichen Inhalt des Datenfeldes
        locSb.Append(locDataRow(locSpalte.ColumnName).ToString)
        'und schließlich einen Zeilenumbruch.
        locSb.Append(vbNewLine)
    Next

    'Alles in einer MessageBox ausgeben.
    MessageBox.Show(locSb.ToString, "Datenzeile enthält:", _
        MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
```

Einige SELECT-Beispiele:

Sie können, wie zuvor schon angedeutet, die SELECT-Zeichenfolge des vorherigen Beispielprojektes durch Eingabe in der entsprechenden TextBox völlig frei gestalten, um beliebige Abfragen zu erstellen.

Viele Entwickler verstehen anfangs vergleichsweise schnell das Prinzip solcher Abfragen, doch Details machen Probleme. Zwar können ein paar Beispiele, die Sie an dieser Stelle finden, natürlich nicht eine vernünftige Einführung in die T-SQL-Sprache von SQL Server ersetzen, aber diese Beispiele helfen Ihnen auf jeden Fall über die ersten Hürden bei der Ausformulierung von Abfragen hinweg und ersparen Ihnen, so denke ich, einige Stunden Suchen im Internet.

Die folgende Bildreihe gibt Ihnen ein paar Einblicke in die Abfragen und ihre möglichen Ergebnisse (die Demodatenbank bei den Beispielen immer vorausgesetzt).

SELECT-Beispiel 1 – Bereichsabfrage zwischen zwei Datumswerten

Aufgabe: Sie möchten alle Zeiten der Zeitentabelle ermitteln, die an einem bestimmten Buchungsdatum aufgetreten sind (im Beispiel am 15.3.2006):

SELECT-Command-String:

```
SELECT * FROM Zeiten WHERE StartZeit > CONVERT(datetime,'15.03.2006 00:00:00',104)
        AND StartZeit < CONVERT(datetime,'15.03.2006 23:59:59',104)
```

Anmerkung: Da die Buchungsdaten unserer Beispieldatenbank auch den Datumspart enthalten, selektieren Sie die Startzeit eines Datums *zwischen* 0:00:00 Uhr und 23:59:59; nur mit der reinen Angabe des Datums würden Sie natürlich keinen einzigen Datensatz zurückerhalten, es sei denn die Buchung begänne exakt um 0:00 Uhr.

Und ebenfalls wichtig zu wissen: verwenden Sie die CONVERT-Funktion von T-SQL (wie im Beispiel zu sehen), um eine Datumskonstante, die als Zeichenkette vorliegt, in den eigentlichen Datumswert umzuwandeln. Geben Sie der CONVERT-Funktion als dritten Parameter dabei den Stilwert 104 an, der bestimmt, dass es sich um ein deutsches Datumsformat handelt. Auf diese Weise haben Sie keine Probleme mit kulturabhängigen Abfragen wie dieser auf Plattformen anderer als der deutschen Kultur. Ließen Sie eine implizite Konvertierung vornehmen, indem Sie die Zeichenkette ohne zu Hilfe-nahme der CONVERT-Funktion verwendeten, würde die Abfrage auf einem englischsprachigen System fehlschlagen, da hier einerseits Monat und Tag vertauscht sind und andererseits Zeiten nicht im 24-Stundenformat sondern mit AM/PM-Designatoren angegeben werden. Mit der Angabe von 104 zwingen Sie SQL Server quasi das deutsche Datumsformat für solche Abfrage auf.

Aussehen des Resultsets:

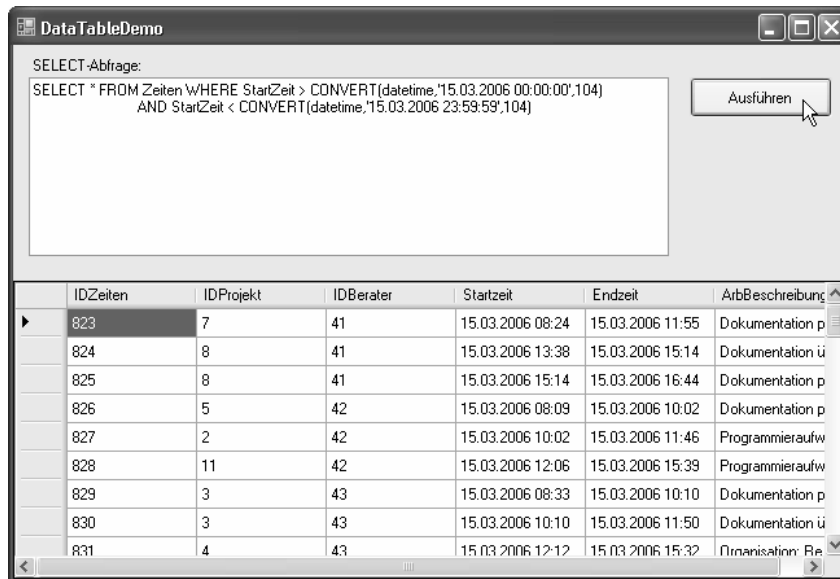


Abbildung 32.17: Resultset, das sich aus der Abfrage des entsprechenden Zeitbereichs ergibt

SELECT-Beispiel 2 – Gezieltes Auswählen von Feldern einer Tabelle

Aufgabe: Sie möchten nicht alle Felder einer Tabelle in ein Resultset aufnehmen, sondern nur gezielte. Im folgenden Beispiel ermitteln Sie nur die Berater-ID der Mitarbeiter, die am 17.3.2006 vor 13:00 Uhr gearbeitet haben, die Projekt-ID der Projekte, an denen sie gearbeitet haben sowie die Anfangszeit.

Anmerkung: Wenn Sie bei SELECT-Abfragen die Feldnamen einzeln angeben (nicht über das Joker-Zeichen »*«), werden nur die angegebenen Felder als Spalten im Resultset angelegt. Natürlich können Sie dennoch alle anderen Felder für Selektierungen (mit WHERE) und Sortierungen (mit ORDER BY) verwenden.

SELECT-Command-String:

```
SELECT [IDBerater], [IDProjekt], [StartZeit] FROM [Zeiten]
WHERE StartZeit > CONVERT(datetime,'17.03.2006 00:00:00',104)
AND StartZeit <= CONVERT(datetime,'17.03.2006 13:00:00',104)
```

Aussehen des Resultsets:

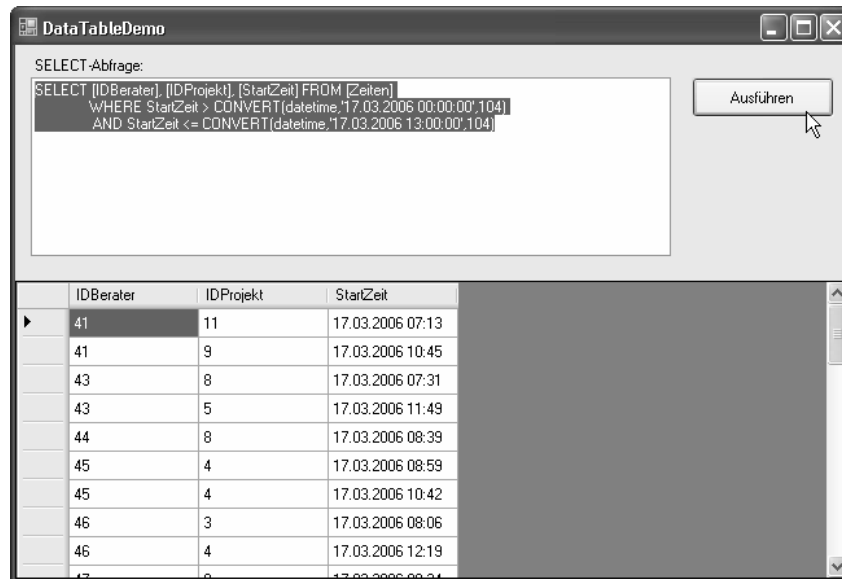


Abbildung 32.18: Resultset, das sich aus der Abfrage des entsprechenden Zeitbereichs ergibt, wobei nur die angegebenen Spalten in das Resultset einfließen

SELECT-Beispiel 3 – Verknüpfung mehrere Tabellen mit Join-Abfragen

Aufgabe: Sie möchten eine Liste mit Personalnummer, Vornamen und Nachnamen der Mitarbeiter ermitteln, die in der Woche vom 15.3. bis zum 20.3.2004 vor 8 Uhr morgens zu arbeiten begonnen haben. Die Anfangszeit soll ebenfalls mit in der Liste enthalten sein. Geordnet werden soll die Liste nach Mitarbeiternachnamen.

Anmerkung: Mit *Join*-Abfragen, die mehrere Tabellen in der *SELECT*-Abfrage beinhalten, können Sie Resultsets erzeugen, die Daten aus verschiedenen Tabellen kombinieren.

SELECT-Command-String:

```
SELECT      Zeiten.Startzeit, Zeiten.ArbBeschreibung, Berater.Nachname, Berater.Vorname,
           Zeiten.IDBerater

FROM        Zeiten INNER JOIN
           Berater ON Zeiten.IDBerater = Berater.IDBerater

WHERE       (Zeiten.Startzeit > CONVERT(datetime, '17.03.2006 00:00:00', 104))
           AND (Zeiten.Startzeit <= CONVERT(datetime, '17.03.2006 13:00:00', 104))

ORDER BY   Berater.Nachname
```

Aussehen des Resultsets:

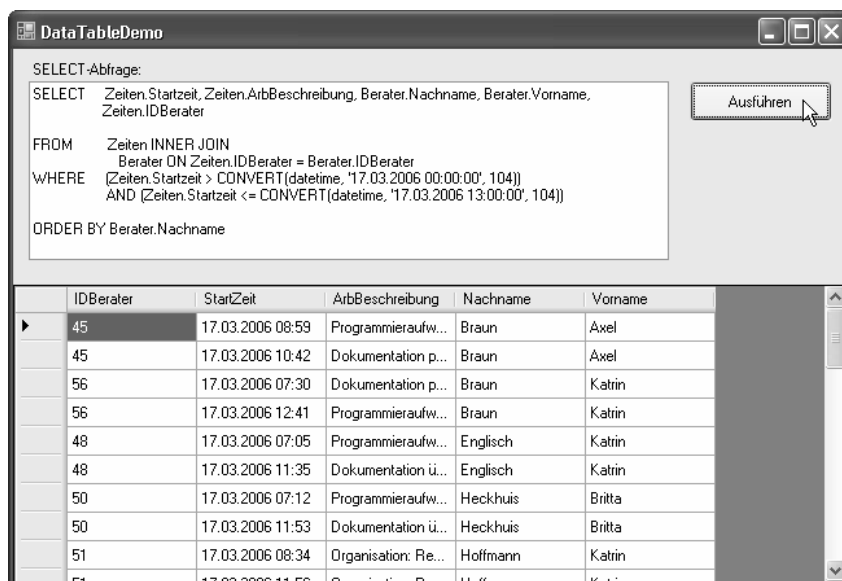


Abbildung 32.19: Resultset, das sich aus der Abfrage des entsprechenden Zeitbereichs ergibt, wobei nur die angegebenen Spalten in das Resultset einfließen; zwei Tabellen fließen in die Abfrage ein, die mit einer *JOIN*-Abfrage verknüpft sind

HINWEIS: Feldnamen in eckigen Klammern zu platzieren ist bei SELECT-Anweisungen nicht unbedingt notwendig. Auf diese Weise verhindern Sie aber, dass der SQL-Parser der jeweiligen Datenbank-Engine Feldnamen von SQL-Befehlen nicht unterscheiden kann oder dass die Namen aufgrund von Sonderzeichen wie Leerzeichen in »My Customer« nicht zu erkennen sind. Um ganz auf der sicheren Seite zu sein, sollten Sie nicht nur Feldnamen in eckigen Klammern einschließen, sondern auch nach Möglichkeit auf Feldnamen verzichten, die SQL-Anweisungen darstellen. Gerade der Feldname *Name* ist bei vielen fehlschlagenden SQL-Anweisungen Grund für stundenlange Fehlersuche – deswegen heißt das Feld in der Beispieldatenbank auch *Nachname*.

Ändern und Ergänzen von Daten in Datentabellen

Ich möchte Ihnen nicht die Illusionen rauben, aber wenn Sie in Ihrer längeren Programmierkarriere schon mit Vorläufern von ADO.NET (ADO, DAO, RDO) gearbeitet haben, dann wird Ihnen die folgende Behauptung vielleicht sehr merkwürdig und unglaublich vorkommen:

»Es gibt nur jeweils eine Möglichkeit, Daten in einer Datenbanktabelle von außen zu verändern oder eine Tabelle um Daten zu ergänzen, nämlich mit der INSERT-, DELETE- und der UPDATE-SQL-Anweisung.«⁶

⁶ Bei SQL Server 2005 bzw. diesem im Zusammenspiel mit ADO.NET 2.0 ist das nicht mehr ganz korrekt, da es auch die Möglichkeit gibt, große Massen von Daten mit so genannten Bulk-Kopierfunktionen in eine SQL Server-Datenbank zu »pumpen«. Doch dieses Thema würde in diesem Umfeld zu weit führen.

»Moment«, werden Sie vielleicht jetzt sagen, »UPDATE schön und gut, aber wieso SQL? Es gab doch schon zu ADO und DAO Recordset-Objekte, mit denen das Ergänzen und Verändern von Daten einfach mit Edit- und Update der entsprechenden Recordset-Objekte funktionierte oder nicht?«.

Sie haben Recht. Und auch nicht. Denn was vielen Programmierern immer verborgen blieb, war die Weise, wie Daten in einer Datenbanktabelle tatsächlich ergänzt oder verändert wurden, und das folgende Beispiel gibt Ihnen einen kleinen Vorgeschmack auf die wirkliche Realität.⁷

BEGLEITDATEIEN: Sie finden dieses Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - SmartClient\Kap32\Insert Update manuell\`.

Dieses Beispielprogramm ist eine Konsolenanwendung, also – dank wenig Overhead – relativ leicht zu durchschauen. Um nachvollziehen zu können, was beim Ablauf des Programms passiert, werden Sie eine Verbindung vom Server-Explorer zur Beispieldatenbank benötigen. Wie Sie eine Verbindung herstellen können, erfahren Sie im ► Abschnitt »Einsehen von Daten mit dem Server-Explorer« ab Seite 995.

Wenn Sie das Programm starten, sehen Sie zunächst die folgende Bildschirmausgabe (hier zum besseren Verständnis ein wenig leserlicher formatiert)

```
INSERT INTO [Projekte] ([Projektname],[Projektbeschreibung],[Startzeitpunkt],
                        [Endzeitpunkt],[Ausführungsort])
VALUES ('Orcas',' (VS 2007) Schreiben des Entwicklerbuchs','01.02.2007','31.12.2007','Büro')

INSERT INTO [Projekte] ([Projektname],[Projektbeschreibung],[Startzeitpunkt],
                        [Endzeitpunkt],[Ausführungsort])
VALUES ('Katmai','(SQL Server 2005) Schreiben des Crash-Kurs','01.03.2007','31.01.2008','Büro')
```

Betrachten Sie die hinzugefügten Daten nun im Server-Explorer
Drücken Sie anschließend Return

Wechseln Sie nun, **ohne** das Programm zunächst zu beenden, zum Server-Explorer. Falls der Server-Explorer im Laufzeitmodus nicht angezeigt werden sollte, aktivieren Sie ihn, indem Sie aus dem Menü *Ansicht* den Menüpunkt *Server-Explorer* auswählen.

Wenn Sie die Verbindung zur Demodatenbank hergestellt haben, öffnen Sie alle Zweige und doppelklicken Sie auf die Tabelle *Projekte*. Sie sehen die neuen Datensätze nun in der Tabelle, etwa wie in Abbildung 32.20 zu sehen.

⁷ So viel zu roter und blauer Pille ...

Projekte: Abfra...press.Hecktick) mdlMain.vb						
IDProjekte	Projektname	Projektbeschreibung	Startzeitpunkt	Endzeitpunkt	Ausführungsort	
1	Visual Basic 200...	Erstellung eines 1.100 Seiten Buches für Mic...	01.02.2005 00:00:00	15.03.2006 00:00:00	Office	
2	Adresso.NET	Erstellung einer Client-/Server-fähigen Adre...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
3	ASP.NET Fachle...	Fachlektorat von Holger Schwichtenbergs A...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
4	HeckTick	Erstellung einer Client-/Server-fähigen Soft...	24.12.2005 00:00:00	30.06.2006 00:00:00	Vor Ort	
5	Visual Basic 200...	Erstellung eines 1.100 Seiten Buches für Mic...	01.02.2005 00:00:00	15.03.2006 00:00:00	Office	
6	Adresso.NET	Erstellung einer Client-/Server-fähigen Adre...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
7	ASP.NET Fachle...	Fachlektorat von Holger Schwichtenbergs A...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
8	HeckTick	Erstellung einer Client-/Server-fähigen Soft...	24.12.2005 00:00:00	30.06.2006 00:00:00	Vor Ort	
9	Visual Basic 200...	Erstellung eines 1.100 Seiten Buches für Mic...	01.02.2005 00:00:00	15.03.2006 00:00:00	Office	
10	Adresso.NET	Erstellung einer Client-/Server-fähigen Adre...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
11	ASP.NET Fachle...	Fachlektorat von Holger Schwichtenbergs A...	24.12.2005 00:00:00	30.06.2006 00:00:00	Office	
12	HeckTick	Erstellung einer Client-/Server-fähigen Soft...	24.12.2005 00:00:00	30.06.2006 00:00:00	Vor Ort	
18	Orcas	(VS 2007) Schreiben des Entwicklerbuchs	01.02.2007 00:00:00	31.12.2007 00:00:00	Büro	
19	Katmai	(SQL Server 2005) Schreiben des Crash-Kurs	01.03.2007 00:00:00	31.01.2008 00:00:00	Büro	
*	NULL	NULL	NULL	NULL	NULL	

Abbildung 32.20: Mit den zwei INSERT-SQL-Anweisungen wurden die Datensätze der Tabelle hinzugefügt

Wechseln Sie anschließend zurück zum Konsolenfenster, und drücken Sie **Return**. Die Bildschirmausgabe wird anschließend um die folgenden Zeilen ergänzt:

```
UPDATE [Projekte] SET [Projektbeschreibung]='(SQL Server 2007) Schreiben des Crash-Kurses',
                    [Startzeitpunkt]='01.02.2007',
                    [Endzeitpunkt]='31.10.2007',
                    [Ausführungsort]='Ruprechts Büro'
WHERE Projektname='Katmai'
```

Betrachten Sie die geänderten Daten nun im Server-Explorer
Drücken Sie anschließend Return

Wenn Sie anschließend zurück zur Datenansicht der Projekttable wechseln, dort das Kontextmenü öffnen und den Eintrag *Ausführen* wählen, aktualisiert der Server-Explorer das angezeigte Resultset. Sie sehen dann, dass sich der letzte Datensatz in verschiedenen Feldern geändert hat.

Was machte das »alte« ADO?

Wie war das jetzt mit ADO und dem überaus einfachen Updaten von Daten in Recordset-Objekten? Eine simple SELECT-Abfrage genügte doch damals, um das Resultset einer Abfrage zu erhalten. Anschließend konnte man einfach durch Ändern der Feldeigenschaften oder AddNew-Befehle Änderungen und Ergänzungen an die Datenbank zurück übermitteln.

Tatsache ist: Auch das alte ADO hat mit UPDATE- und INSERT-SQL-Anweisungen gearbeitet, und zwar genauso, wie Sie es im vorherigen Beispiel beobachten konnten. Auf Grund des Resultset-Schemas, das ADO im Rahmen einer SELECT-SQL-Anweisung erhalten konnte, hat ADO hinter den Kulissen die zum Aktualisieren oder Ergänzen notwendigen SQL-Befehle selbst zusammengestellt und anschließend verwendet. Diese Vorgehensweise hatte für den Programmierer zwar den Vorteil, sehr schnell zum Ergebnis zu kommen, allerdings waren die Abfragen, da sie keinen Eingriff von außen zuließen, starr und unflexibel.

Das hat sich mit ADO.NET grundlegend geändert.

Verwenden von DataAdapter, DataTable und Command-Objekten zur Übermittlung von Aktualisierungen

Sie werden mir zustimmen, dass die Vorgehensweise aus dem letzten Beispiel zum Hinzufügen von Daten zu einer Tabelle bzw. zum Ändern von Daten in einer Tabelle zwar zum Ziel führt, aber viel zu aufwändig ist, um für größere Vorhaben geeignet zu sein. Es wäre einfach undenkbar, müsste man, nachdem man beispielsweise Änderungen an einem DataTable-Objekt selber vorgenommen hat, danach zusätzlich noch dafür sorgen, dass das eigene Programm entsprechende SQL-Anweisungen generiert, die die Tabelle durchsucht und auf diese Weise Änderungen an die Datenbank übermittelt.

Prinzipiell funktioniert die Vorgehensweise zum Übermitteln von Änderungen zwar genauso, doch Sie müssen sie nicht selber machen. *DataAdapter* und *DataTable* spielen hier in einem Team zusammen, und Sie geben nur die groben Rahmenparameter vor, damit Übermittlungen von Änderungen an Datentabellen erfolgreich zum Ziel führen.

Die Kunst, die ADO.NET dabei zu finden versucht, ist, dem Entwickler auf der einen Seite zwar die Änderungen, die er über Vorgaben im *DataTable*-Objekt macht, möglichst automatisch in der Datenbank via DELETE-, INSERT- bzw. UPDATE-Anweisungen umzusetzen, ihn aber dennoch nicht durch zu starre Vorgaben einzuschränken.

Das heißt für das, was ADO.NET leisten muss: Der Entwickler sollte Änderungen an den Daten bequem in den *DataRow*-Objekten einer *DataTable* vornehmen können. Nachdem eine *DataTable* durch *Fill* oder *FillSchema* gefüllt bzw. vorbereitet wurde, kann er mit Modifizierungen beginnen.

- Wenn Zeilen zur *DataTable* hinzugefügt worden sind, muss der *DataAdapter* diese Zeilen in einer *DataTable* als neu erkennen und automatisch eine entsprechende INSERT-Anweisung verwenden, um die Zeilen in der Datenbanktabelle einzufügen.
- Wenn Daten einer *DataTable* geändert wurden, muss der *DataAdapter* die geänderten Zeilen (*DataRow*-Objekte) ebenfalls erkennen. Er muss aber auch die ursprünglichen Daten noch herausfinden können (die Daten der jeweiligen *DataRow vor* der Änderung), damit diese im WHERE-Teil einer UPDATE-Anweisung eingesetzt werden können. Nur so kann der *DataAdapter* die Datensätze in der Datenbank überhaupt finden, die mit UPDATE geändert werden sollen (siehe 2. Bildschirmausgabe des vorherigen Beispiels)
- Wenn Zeilen einer *DataTable* gelöscht wurden, muss der *DataAdapter* dennoch noch Zugriff auf die gelöschten *DataRow*-Objekte der *DataTable* haben, damit er eine DELETE-Anweisung verwenden und mit entsprechenden Parametern versehen kann, die die Zeilen in der Tabelle löschen.

Das alte ADO sorgte dafür, dass alle SQL-Anweisungen zur Datenänderung in Datenbanktabellen hinter den Kulissen erstellt wurden. Wollte ein erfahrener Datenbankspezialist selbst Hand anlegen und optimierte Aktualisierungslogiken oder gar – für SQL Server – gespeicherte Prozeduren entwerfen, schaute er in die Röhre.

Der *DataAdapter* von ADO.NET schlägt hier eine Brücke zwischen Flexibilität und einfacher Handhabung: Sie können die Abfragelogik auf der einen Seite zwar komplett selbst implementieren, kön-

nen aber dennoch ein `DataTable`⁸-Objekt verwenden, um Änderungen an Resultsets auf einfachste Weise durchzuführen und sie zur Datenbank übertragen.

Die Gratwanderung, die ADO.NET – oder besser: der *DataAdapter* – beschreiten muss, ist, dabei ein halbwegs handhabbares Verfahren zur Verfügung zu stellen, mit dem INSERT-, UPDATE- und DELETE-Anweisungen parametrisiert werden können. Anstatt also eine vollständige Logik vorzugeben, etwa wie,

```
UPDATE [Projekte] SET [Projektname]='Katmai',
                    [Projektbeschreibung]='(SQL Server 2007) Schreiben des Crash-Kurses',
                    [Startzeitpunkt]='01.02.2007',
                    [Endzeitpunkt]='31.10.2007',
                    [Ausführungsort]='Ruprechts Büro'
WHERE Projektname='Katmai'
```

ergibt es mehr Sinn, die eigentlichen Parameter zunächst auszulassen, etwa wie im folgenden Beispiel:

```
UPDATE [Projekte] SET [Projektname]=@Var1,
                    [Projektbeschreibung]=@Var2,
                    [Startzeitpunkt]=@Var3,
                    [Endzeitpunkt]=@Var4,
                    [Ausführungsort]=@Var5
WHERE ([Projektname]=@Var6) AND ([Projektbeschreibung]=@Var7) AND
      ([Startzeitpunkt]=@Var8) AND ([Endzeitpunkt]=@Var9) AND
      ([Ausführungsort]=@Var10)
```

Diese Abfrage ist nun von den eigentlichen Parametern unabhängig (auch wenn sie in dieser Form zunächst nicht funktionieren würde) und kann an mehreren Stellen in einer Anwendung für unterschiedliche Modifikationen und nicht nur eine einzige Modifikation verwendet werden. Voraussetzung dafür ist natürlich, dass es eine Instanz gibt, die aus den einzelnen mit @ beginnenden Platzhaltervariablen wieder richtige Werte macht, die es auf der einen Seite abzufragen (Suchkriterien für den ursprünglichen Datensatz werden hinter WHERE angegeben, um den zu ändernden Datensatz in der Tabelle zu finden) und auf der anderen Seite zu aktualisieren gilt (der SET-Teil bestimmt die neuen Werte des Datensatzes). Diese Instanz muss aber nicht nur die Werte einsetzen, sie muss dazu auch wissen, welchen Typs die Werte sein müssen, die @-Variablen in der allgemeingültigen Abfrage ersetzen sollen.

Das *Command*-Objekt, das einen solchen Aktualisierungs-SQL-Befehl kapselt, stellt aus diesem Grund ein Parameters-Array zur Verfügung, das die eigentlichen Parameter in der Reihenfolge ihres Auftretens sozusagen »ver-typt«. Dabei wird ebenfalls festgehalten, welche Parameter für eine neue Wertbestimmung dienen (im Beispiel die ersten fünf @-Variablen) und welche für die Datensatzidentifizierung zuständig sind (die letzten @-Variablen).

Damit ist die Grundvoraussetzung geschaffen, um eine Verbindung zwischen völlig von der Datenbank losgelösten Datenzeilen und der Datenbank herzustellen. Wenn die Datenzeilen, also die `DataRow`-Objekte einer `DataTable`, anschließend aktualisiert, gelöscht oder ergänzt werden, greift der

⁸ Das `DataTable`-Objekt ist nicht das einzige Objekt. Es ist aber das wohl am häufigsten verwendete (wenn auch in vielen Fällen nur indirekt über das *DataSet*-Objekt). Im Rahmen dieses Buches möchte ich mich auf das `DataTable`-Objekt beschränken.

DataAdapter auf insgesamt drei verschiedene *Command*-Objekte zurück, um die Änderungen an die Datenbank zu übermitteln. Er verfährt dabei folgendermaßen:

- Wenn er Zeilen in der *DataTable* findet, die verändert worden sind, greift er auf den *UpdateCommand* des *DataAdapter* zurück, um zunächst einmal die allgemeingültige Aktualisierungslogik (`UPDATE [Tabellenname] SET ... WHERE ...`) zu ermitteln. Er setzt dann – und dabei nimmt er die Parameters-Auflistung des *Commands* zu Hilfe – anstelle der *@*-Variablen die neuen Werte der *DataRow* ein. Um den zu verändernden Datensatz zu finden, ersetzt er ferner die hinter der *WHERE*-Klausel stehenden Fragezeichen durch die Originalwerte, die in einem *DataRow*-Objekt erhalten bleiben, auch wenn neue Werte zugewiesen wurden.⁹
- Findet er Zeilen in der *DataTable*, die neu hinzugekommen sind, greift er auf die *InsertCommand*-Eigenschaft des *DataAdapters* zurück, um die allgemeingültige Aktualisierungslogik zu ermitteln (`INSERT INTO ...`). Für die Parameter verfährt er anschließend wie bei der Erstellung des *UPDATE*-Befehls, mit dem Unterschied, dass der Part für die Originalwertbehandlung nicht zur Anwendung kommt (wozu auch – bei neuen Datenzeilen gibt es keine alten Originalwerte).
- Für gelöschte Zeilen in der *DataTable* greift er auf die *DeleteCommand*-Eigenschaft des *DataAdapter* zurück und verfährt für die weitere Aufbereitung der *DELETE*-SQL-Anweisung auf gleiche Weise.

Schauen wir uns an, wie sich die Anwendung dieser Verfahren in der Praxis darstellt.

BEGLEITDATEIEN: Sie finden das folgende Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32\Insert Update DataAdapter\`.

Das Programm macht exakt das Gleiche wie das vorherige Beispiel – es verwendet nur die gerade vorgestellte Vorgehensweise.

HINWEIS: Bitte achten Sie darauf, dass Sie vor dem Programmstart die vom vorherigen Beispiel hinzugefügten Datensätze wieder löschen, damit das Beispiel reibungslos funktionieren kann. Verwenden Sie auch dafür am besten wieder den Server-Explorer.

```
Imports System.Data.SqlClient
```

```
Module md1Main
```

```
    Sub Main()
```

```
        Dim locConnection As SqlConnection
```

```
        'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
```

```
        locConnection = New _  
            SqlConnection("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")
```

```
        'Verbindung öffnen
```

```
        locConnection.Open()
```

⁹ Sie können den ursprünglichen Wert eines *DataRow*-Feldinhaltes abrufen, indem Sie der *Item*-Eigenschaft nicht nur einen Index in Form des Feldnamens oder einer Ordnungszahl übergeben, sondern auch die Konstante *DataRowVersion.Original*.

```

'Damit sorgen wir dafür, dass die Verbindung wieder geschlossen wird,
'wenn locConnection aus dem Using-Scope herausläuft.
Using locConnection

'Alle Commands im DataAdapter definieren
Dim locAdapter As SqlDataAdapter = DataAdapterVorbereiten(locConnection)

'Neue Datentabelle
Dim locDataTable As New DataTable

'Könnten wir selber definieren, aber so geht's schneller.
locAdapter.FillSchema(locDataTable, SchemaType.Source)

'Neue Datenzeile
Dim locDataRow As DataRow = locDataTable.NewRow()

'Entsprechend des Datenschemas der Tabelle mit Datenfüllen.
'(IDProjekte wird von Sql Server selbst geschrieben, da es ein
' AutoIncrement-Feld ist. Deswegen übergeben wir hier 'Nothing')
locDataRow.ItemArray = New Object() {Nothing, "Orcas", "(VS 2007) Schreiben des Entwicklerbuchs", _
                                     #2/1/2007#, #12/31/2007#, "Büro"}
'Die Zeile der Tabelle hinzufügen
locDataTable.Rows.Add(locDataRow)

'Das Gleiche mit einer weiteren Zeile
locDataRow = locDataTable.NewRow
locDataRow.ItemArray = New Object() {Nothing, "Katmai", "(SQL Server 2005) Schreiben des Crash-Kurs", _
                                     #3/1/2007#, #1/31/2008#, "Büro"}
locDataTable.Rows.Add(locDataRow)

'Und nun die neue Tabelle mit der Datenbank synchronisieren.
'Hier wird SQL-INSERT zweimal hintereinander ausgeführt.
locAdapter.Update(locDataTable)

Console.WriteLine("Betrachten Sie die hinzugefügten Daten nun im Server-Explorer")
Console.WriteLine("Drücken Sie anschließend Return")
Console.WriteLine()
Console.ReadLine()

'DataTable neu einlesen
locDataTable.Clear()
locAdapter.Fill(locDataTable)

'Jetzt verwenden wir eine View, um die 'Katmai'-Zeile zu ermitteln.
'Das ist die, die wir als letztes geändert haben!

Dim locDataView As New DataView(locDataTable, "Projektname='Katmai'", _
                                "Projektname", DataViewRowState.CurrentRows)

'Eigentlich können wir nur genau eines zurückbekommen.
locDataRow = locDataView(0).Row

```

```

'Daten ändern.
locDataRow.ItemArray = New Object() {Nothing, "Katmai", "(SQL Server 2007) Schreiben des Crash-Kurses",
                                     #2/1/2007#, #10/31/2007#, "Ruprechts Büro"}

'Änderungen zurückschreiben.
locAdapter.Update(locDataTable)

Console.WriteLine("Betrachten Sie die geänderten Daten nun im Server-Explorer")
Console.WriteLine("Drücken Sie anschließend Return")
Console.ReadLine()
End Using

End Sub

```

Sie sehen, dass die Daten über die `DataTable` sehr viel besser zu handhaben sind. Das Hinzufügen geht genau wie das Ändern von Daten recht einfach von der Hand.

Filtern und Sortieren von Zeilen einer `DataTable` mit der `DataRowView`-Klasse

Eine kurze Anmerkung möchte ich aber an dieser Stelle über das Ermitteln des `DataRow`-Objektes verlieren, an dem anschließend das Ändern von Daten demonstriert wird. Zu diesem Zweck verwendet das Beispiel nämlich die so genannte `DataRowView`-Klasse, die Sie sich wie eine Abfrage-Engine für `DataTable`-Objekte vorstellen können. So, wie Sie mit der `SELECT`-Anweisung nach bestimmten Daten in einer Datenbank direkt selektieren und sortieren können, diese also auch zusätzlich filtern können, verwenden Sie eine `DataRowView` als Filter- oder Sortierhilfe für bereits mit Daten gefüllte `DataTable`-Objekte. Die Filter- und Sortierfunktionen von `DataRowView` sind dabei völlig eigenständig – was auch logisch ist, da ein `DataTable`-Objekt seine einzelnen Datenzeilen auch völlig von der Datenbank getrennt verwaltet, Ein Zugriff auf den »Datenbankmotor« wäre einem `DataRowView`-Objekt schon dadurch gar nicht möglich.

Sie übergeben einer `DataRowView`-Klasse im Konstruktor die zu bearbeitende `DataTable` und optional eine Zeichenkette mit einer Filterdefinition sowie eine weitere, die den Namen der Spalte der `DataTable` enthält, nach der die Daten sortiert werden sollen.

Die `DataRowView` packt beim Filtern oder Sortieren die ursprüngliche Tabelle aber gar nicht an; vielmehr legt Sie scheinbar eine Kopie der Ursprungstabelle an, die aus `DataRowView`-Objekten bestehen, von denen aber jedes einzelne quasi nur »virtuell« existiert. Ein `DataRowView`-Objekt ist nämlich im Grunde genommen nur ein einsortierter (oder bzw. und gefilterter) Zeiger auf die originalen, unangetasteten Datenzeilen der `DataTable`.

Über den Indexer der `DataRowView` können Sie anschließend auf einzelne `DataRowView`-Objekte zugreifen, mit deren jeweiliger `Row`-Eigenschaft Sie die ursprüngliche `DataRow` der zuvor übergebenden `DataTable` ermitteln können.

Im Beispiel nutzen wir ein `DataRowView`-Objekt, um die Datenzeile zu ermitteln, auf die die Filterbedingung `Projektname='Katmai'` zutrifft; das ist nämlich exakt der `Projektname` der Datenzeile, die wir zuvor der Projekttabelle hinzugefügt haben.

Einrichten der Command-Objekte des Beispiels

Der übrige Code befindet sich in einer einzigen Prozedur, die, wie in der Einleitung dieses Abschnittes beschrieben, dafür zuständig ist, die benötigten *Command*-Objekte für die entsprechenden Aufgaben (Aktualisieren, Einfügen und Löschen von Datensätzen) einzurichten.

Sie sehen auf den folgenden drei Codeseiten dabei nicht nur, wie das Einrichten der *Command*-Zeichenketten für die Steuerung der SQL-Befehle an sich funktioniert. Sie erkennen auch das Zusammenspiel mit der Parameters-Auflistung, auf die der *DataAdapter* später, beim Einsetzen der »wirklichen« Werte zurückgreift.

Und Sie sehen noch etwas: Wie vergleichsweise aufwändig die Einrichtung dieser *Command*-Objekte ist. Doch keine Angst: Visual Basic 2005 hält zwei weitere Techniken für Sie bereit, mit denen Sie diesen Aufwand minimieren können, aber ohne etwas von der Flexibilität dieses Konzeptes aufgeben zu müssen! Wie das funktioniert, zeigen die nächsten beiden Abschnitte.

```
Private Function DataAdapterVorbereiten(ByVal Conn As SqlConnection) As SqlDataAdapter

    Dim locDa As New SqlDataAdapter

    locDa = New System.Data.SqlClient.SqlDataAdapter

    locDa.SelectCommand = New SqlCommand("SELECT * FROM [Projekte]")
    locDa.SelectCommand.Connection = Conn

    locDa.DeleteCommand = New System.Data.SqlClient.SqlCommand
    locDa.DeleteCommand.Connection = Conn
    locDa.DeleteCommand.CommandText = "DELETE FROM [dbo].[Projekte] WHERE " & _
        "((([IDProjekte] = @Original_IDProjekte) AND ([Projektname]" & _
        " = @Original_Projektname) AND ((@IsNull_Projektbeschreibung = 1 AND " & _
        "[Projektbeschreibung] IS NULL) OR ([Projektbeschreibung] = @Original_Projektbeschreibung)))" & _
        " AND ([Startzeitpunkt] = @Original_Startzeitpunkt) AND ([Endzeitpunkt]" & _
        "= @Original_Endzeitpunkt) AND ((@IsNull_Ausführungsort = 1 AND [Ausführungsort]" & _
        " IS NULL) OR ([Ausführungsort] = @Original_Ausführungsort)))"
    locDa.DeleteCommand.CommandType = System.Data.CommandType.Text
    locDa.DeleteCommand.Parameters.Add( _
        New System.Data.SqlClient.SqlParameter("@Original_IDProjekte", System.Data.SqlDbType.Int, 0, _
        System.Data.ParameterDirection.Input, 0, 0, _
        "IDProjekte", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
    locDa.DeleteCommand.Parameters.Add( _
        New System.Data.SqlClient.SqlParameter("@Original_Projektname", System.Data.SqlDbType.NVarChar, 0,
        System.Data.ParameterDirection.Input, 0, 0, _
        "Projektname", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
    ' Rest aus Platzgründen weggelassen

    locDa.InsertCommand = New System.Data.SqlClient.SqlCommand
    locDa.InsertCommand.Connection = Conn
    locDa.InsertCommand.CommandText = "INSERT INTO [dbo].[Projekte] ([Projektname], [Projektbeschreibung], " &
        "[Startzeitpunkt], [Endzeitpunkt], [Ausführungsort]) VALUES (@Projektname, " & _
        "@Projektbeschreibung, @Startzeitpunkt, @Endzeitpunkt, @Ausführungsort);"
```

```

locDa.InsertCommand.CommandType = System.Data.CommandType.Text
locDa.InsertCommand.Parameters.Add(
    New System.Data.SqlClient.SqlParameter("@Projektname", System.Data.SqlDbType.NVarChar, 0,
    System.Data.ParameterDirection.Input, 0, 0,
    "Projektname", System.Data.DataRowVersion.Current, False, Nothing, "", "", ""))
locDa.InsertCommand.Parameters.Add(
    New System.Data.SqlClient.SqlParameter("@Projektbeschreibung", System.Data.SqlDbType.NVarChar,
    0, System.Data.ParameterDirection.Input, 0, 0,
    "Projektbeschreibung", System.Data.DataRowVersion.Current, False, Nothing, "", "", ""))
.
. ' Rest aus Platzgründen weggelassen
.

locDa.UpdateCommand = New System.Data.SqlClient.SqlCommand
locDa.UpdateCommand.Connection = Conn
locDa.UpdateCommand.CommandText = "UPDATE [dbo].[Projekte] " &
    "SET [Projektname] = @Projektname, [Projektbeschreibung] = " &
    "@Projektbeschreibung, [Startzeitpunkt] = @Startzeitpunkt, [Endzeitpunkt] = " &
    "@Endzeitpunkt, [Ausführungsort] = @Ausführungsort WHERE (([IDProjekte] = " &
    "@Original_IDProjekte) AND ([Projektname] = @Original_Projektname) AND " &
    "(@IsNull_Projektbeschreibung = 1 AND [Projektbeschreibung] IS NULL) OR ([Projektbeschreibung] " &
    "= @Original_Projektbeschreibung)) AND ([Startzeitpunkt] = @Original_Startzeitpunkt) " &
    "AND ([Endzeitpunkt] = @Original_Endzeitpunkt) AND ((@IsNull_Ausführungsort = 1 AND " &
    "[Ausführungsort] IS NULL) OR ([Ausführungsort] = @Original_Ausführungsort));"
locDa.UpdateCommand.CommandType = System.Data.CommandType.Text
locDa.UpdateCommand.Parameters.Add(
    New System.Data.SqlClient.SqlParameter("@Projektname", System.Data.SqlDbType.NVarChar, 0,
    System.Data.ParameterDirection.Input, 0, 0,
    "Projektname", System.Data.DataRowVersion.Current, False, Nothing, "", "", ""))
locDa.UpdateCommand.Parameters.Add(
    New System.Data.SqlClient.SqlParameter("@Projektbeschreibung", System.Data.SqlDbType.NVarChar,
    0, System.Data.ParameterDirection.Input, 0, 0,
    "Projektbeschreibung", System.Data.DataRowVersion.Current, False, Nothing, "", "", ""))
.
. ' Rest aus Platzgründen weggelassen
.

locDa.UpdateCommand.Parameters.Add(
    New System.Data.SqlClient.SqlParameter("@Original_IDProjekte", System.Data.SqlDbType.Int, 0,
    System.Data.ParameterDirection.Input, 0, 0,
    "IDProjekte", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
locDa.UpdateCommand.Parameters.Add(
    New System.Data.SqlClient.SqlParameter("@Original_Projektname", System.Data.SqlDbType.NVarChar, 0,
    System.Data.ParameterDirection.Input, 0, 0,
    "Projektname", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
locDa.UpdateCommand.Parameters.Add(
    New System.Data.SqlClient.SqlParameter("@IsNull_Projektbeschreibung", System.Data.SqlDbType.Int, 0,
    System.Data.ParameterDirection.Input, 0, 0,
    "Projektbeschreibung", System.Data.DataRowVersion.Original, True, Nothing, "", "", ""))
locDa.UpdateCommand.Parameters.Add(
    New System.Data.SqlClient.SqlParameter("@Original_Projektbeschreibung",
    System.Data.SqlDbType.NVarChar, 0, System.Data.ParameterDirection.Input, 0, 0,
    "Projektbeschreibung", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))

```

```

TocDa.UpdateCommand.Parameters.Add( _
    New System.Data.SqlClient.SqlParameter("@Original_Startzeitpunkt", System.Data.SqlDbType.DateTime,
    0, System.Data.ParameterDirection.Input, 0, 0, _
    "Startzeitpunkt", System.Data.DataRowVersion.Original, False, Nothing, "", "", ""))
TocDa.UpdateCommand.Parameters.Add( _
    New System.Data.SqlClient.SqlParameter("@Original_Endzeitpunkt", System.Data.SqlDbType.DateTime, 0,
.
. ' Rest aus Platzgründen weggelassen
.
End Module

```

Für »Mal-Eben-Abfragen« – die CommandBuilder-Klasse

Nun finden Sie es vielleicht ganz toll, welche enorme Flexibilität Ihnen die verschiedenen *Command*-Objekte des *DataAdapters* einbringen. Möglicherweise benötigen Sie diese Flexibilität aber überhaupt nicht für kleinere Datenbankprojekte, sondern sind ein wenig entsetzt über die Vorleistung, die Sie erbringen müssen, um überhaupt die *Update*-Methode des *DataAdapters* anwenden zu können.

Doch keine Angst, ich kann Sie beruhigen, denn auch für diesen Fall haben die Entwickler von ADO.NET vorgesorgt. Ähnlich wie beim alten ADO, das sowohl Schemata als auch die notwendigen Aktualisierungs-SQL-Befehle selbständig erstellen konnte, besteht auch bei ADO.NET die Möglichkeit, diese Aktualisierungslogiken *ausarbeiten zu lassen*. Der Schlüssel zum Glück ist dabei das so genannte *CommandBuilder*-Objekt.

Mit seiner Hilfe benötigen Sie lediglich eine gültige *SELECT*-Abfrage, aus der der *DataAdapter* ein Schema für eine *DataTable* erstellen kann, das aus einem Resultset hervorgeht. Die übrigen Anweisungen erstellt dann anschließend der *CommandBuilder* – im Falle von SQL Server nennt es sich übrigens *SqlCommandBuilder*.

BEGLEITDATEIEN: Sie finden das folgende Beispiel im Verzeichnis `.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32\Insert Update CommandBuilder`. Bitte achten Sie auch hier wieder darauf, dass Sie vor dem Programmstart die vom vorherigen Beispiel hinzugefügten Datensätze löschen, damit das Beispiel reibungslos funktionieren kann.

Das folgende Codelisting zeigt die Vorgehensweise zur Anwendung des *SqlCommandBuilder*.

```

Imports System.Data.SqlClient

Module mdlMain
    Sub Main()

        Dim locConnection As SqlConnection

        'Connection-Objekt holen, damit die Datenbankverbindung hergestellt werden kann
        locConnection = New _
            SqlConnection("Data Source=.\SQLEXPRESS;Initial Catalog=Hecktick;Integrated Security=True")

        'Verbindung öffnen
        locConnection.Open()

        'Damit sorgen wir dafür, dass die Verbindung wieder geschlossen wird,

```

```

'wenn locConnection aus dem Using-Scope herausläuft.
Using locConnection

'Alle Commands im DataAdapter definieren
Dim locAdapter As New SqlDataAdapter

'Die Selectabfrage für den Adapter definieren,
'dann daraus generiert der CommandBuilder die
'anderen Command-Objekte für Einfügen, Aktualisieren und Löschen
locAdapter.SelectCommand = New SqlCommand("SELECT * FROM Projekte")
locAdapter.SelectCommand.Connection = locConnection

'Den CommandBuilder zuhilfe nehmen und dem Adapter zuweisen.
Dim locCmdBuilder As New SqlCommandBuilder(locAdapter)

'Neue Datentabelle
Dim locDataTable As New DataTable

'Könnten wir selber definieren, aber so geht's schneller.
locAdapter.FillSchema(locDataTable, SchemaType.Source)

'Neue Datenzeile
Dim locDataRow As DataRow = locDataTable.NewRow()

'Entsprechend des Datenschemas der Tabelle mit Datenfüllen.
'(IDProjekte wird von Sql Server selbst geschrieben, da es ein
' AutoIncrement-Feld ist. Deswegen übergeben wir hier 'Nothing')
locDataRow.ItemArray = New Object() _
    {Nothing, "Orcas", " (VS 2007) Schreiben des Entwicklerbuchs", _
    #2/1/2007#, #12/31/2007#, "Büro"}

'Die Zeile der Tabelle hinzufügen
locDataTable.Rows.Add(locDataRow)

'Das Gleiche mit einer weiteren Zeile
locDataRow = locDataTable.NewRow
locDataRow.ItemArray = New Object() _
    {Nothing, "Katmai", "(SQL Server 2005) Schreiben des Crash-Kurs", _
    #3/1/2007#, #1/31/2008#, "Büro"}
locDataTable.Rows.Add(locDataRow)

'Und nun die neue Tabelle mit der Datenbank synchronisieren.
'Hier wird SQL-INSERT zweimal hintereinander ausgeführt.
locAdapter.Update(locDataTable)

Console.WriteLine("Betrachten Sie die hinzugefügten Daten nun im Server-Explorer")
Console.WriteLine("Drücken Sie anschließend Return")
Console.WriteLine()
Console.ReadLine()

'DataTable neu einlesen
locDataTable.Clear()
locAdapter.Fill(locDataTable)

```

```

'Jetzt verwenden wir eine View, um die 'Katmai'-Zeile zu ermitteln.
'Das ist die, die wir als letztes geändert haben!

Dim locDataView As New DataView(locDataTable, "Projektname='Katmai'", _
    "Projektname", DataViewRowState.CurrentRows)

'Eigentlich können wir nur genau ein zurückbekommen.
locDataRow = locDataView(0).Row

'Daten ändern.
locDataRow.ItemArray = New Object() _
    {Nothing, "Katmai", "{SQL Server 2007) Schreiben des Crash-Kurses", _
        #2/1/2007#, #10/31/2007#, "Ruprechts Büro"}

'Änderungen zurückschreiben.
locAdapter.Update(locDataTable)

Console.WriteLine("Betrachten Sie die geänderten Daten nun im Server-Explorer")
Console.WriteLine("Drücken Sie anschließend Return")
Console.ReadLine()
End Using

End Sub
End Module

```

WICHTIG: Das reibungslose Funktionieren von `SqlCommandBuilder` ist jedoch an einige Voraussetzungen gebunden. So muss die Tabelle einen Primärschlüssel (*PrimaryKey*) haben, der Teil der `SELECT`-Anweisung ist. Zudem darf die `SELECT`-Anweisung sich nur auf Spalten einer Tabelle beziehen. Falls Sie verknüpfte Tabellen aktualisieren wollen, müssen Sie daher beide Tabellen einzeln mit einem `DataAdapter` in ein `DataSet` laden und können dort ein `DataRelation`-Objekt erstellen. So können dann auch zwei `CommandBuilder` die verknüpften Tabellen aktualisieren.

DataSets und typisierte DataSets

Von der eigentlich bekanntesten ADO.NET-Klasse, der `DataSet`-Klasse, war bislang mit noch keinem Wort die Rede, aber das hat auch seinen Grund. Die bisherigen Beispiele haben ausschließlich mit dem `DataTable`-Objekt für die von der Datenbank getrennte Speicherung von Daten gearbeitet. Erst wenn es darum geht, Tabellenrelationen zu verarbeiten, dann stellen die so genannten `DataSet`-Objekte in Zusammenarbeit mit den `DataRelation`-Objekten eine gute Alternative zu `JOIN`-SQL-Abfragen dar – in vielen Fällen lässt sich damit sogar eine erhebliche Leistungssteigerung von Abfragen und Bearbeitungen erreichen. Und jetzt ahnen Sie es vielleicht schon: Wenn derlei Aktionen mit `DataSet`-Objekten möglich sind, dann dienen sie notwendigerweise auch als Container für mehrere Tabellen, also `DataTable`-Objekten. Und so ist es auch.

Auf das komplette Spektrum der `DataSet`-Klasse kann und will dieses Buch nicht eingehen, dafür sind sie einfach zu mächtig, und die Beschreibungen ihres Umgangs füllen locker halbe Bücher.

Doch eine Sache dieses großen Themas möchte ich Ihnen dennoch nicht vorenthalten, und das ist das interaktive Erstellen von typisierten `DataSets` mit der Visual Studio IDE.

Der Vorteil von typisierten DataSet-Objekten wird deutlich, wenn Sie sich die folgende Abbildung anschauen.

```
'Die Definitionen für typisierte Tabellen befinden sich in der DataSet-Klasse
Dim locProjekteTable As New HecktickDataSet.ProjekteDataTable

'Die Definitionen für typisierte Adapter für Tabellen befinden sich in
'in einem separaten Namespace
Dim locProjekteAdapter As New HecktickDataSetTableAdapters.ProjekteTableAdapter

'Eine neue Datenzeile hinzufügen. Intellisense hilft Ihnen hier beim
'Finden der richtigen Parameter, weil diese Datenzeile jetzt typisiert ist.
locProjekteTable.AddProjekteRow("Orcas", " (VS 2007) Schreiben des Entwicklerbu
#2/1AddProjekteRow (Projektname As String, Projektbeschreibung As String, Startzeitpunkt As Date, Endzeitpunkt As Date, Ausführungsor
```

Abbildung 32.21: Mit typisierten Datenobjekten wird das Entwickeln von Datenanwendungen dank IntelliSense und Typsicherheit nicht nur zum Kinderspiel, sondern macht auch richtig Spaß und spart eine Menge Zeit!

Tabellen in Form von DataTable-Objekten, die sich in typisierten DataSet-Objekten befinden, lassen sich *typsicher* bearbeiten. Sie arbeiten beim Hinzufügen von Datenzeilen nicht mehr »nur« mit einem DataRow-Objekt, sondern vielleicht mit einem ProjekteRow-Objekt. Und Sie fragen ein Datenfeld nicht mehr mit

```
Dim locString As String = locDataRow("Projektname").ToString
```

sondern typsicher auf die folgende Art und Weise:

```
Dim locProjektname As String = locProjekteRow.Projektname
```

Das Schöne: Das Erstellen von typisierten DataSet-Objekten geht in Visual Basic 2005 einfach wie nie, und der folgende Abschnitt zeigt, wie es geht.

TIPP: Am Ende dieses Kapitels finden Sie als abschließendes Schmankerl noch eine komplett implementierte Anwendung, die Ihnen demonstriert, wie Sie die Demodatenbank dieses Kapitels auch tatsächlich in Form einer voll funktionsfähigen *SmartClient*-Anwendung nutzen können!

Erstellen eines typisierten DataSets mit der Visual Studio IDE

Die folgende Schritt-für-Schritt-Anleitung zeigt, wie Sie mit Visual Studio 2005 (Sie benötigen mindestens die Standard-Edition) ein typisiertes DataSet erstellen.

- Legen Sie ein neues Windows Forms-Projekt unter dem Namen *TypedDataSet an*.
- Wählen Sie aus dem Menü *Daten* den Menübefehl *Datenquelle hinzufügen*.

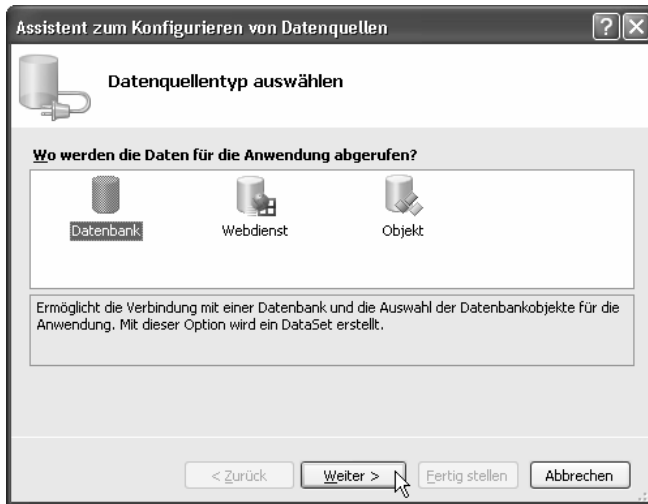


Abbildung 32.22: Dieser Assistent hilft Ihnen beim Erstellen eines neuen typisierten *DataSet*

- Wählen Sie Datenbank, und klicken Sie auf *Weiter*.



Abbildung 32.23: Bestimmen Sie, welche SQL Server-Datenbank als Datenquelle fungieren soll

- Falls Sie bereits eine Datenverbindung in den letzten Beispielen zum Server Explorer hergestellt haben, befindet sich die Datenverbindung bereits in der Aufklappliste, und Sie wählen aus dieser Liste die Beispieldatenbank »HeckTick« aus. Anderenfalls klicken Sie auf *Neue Verbindung*, und richten eine neue Verbindung zur Beispieldatenbank ein. Auf Seite 996 finden Sie die Beschreibung des entsprechenden Dialogs, der Ihnen dabei behilflich ist. Klicken Sie anschließend auf *Weiter*.



Abbildung 32.24: Legen Sie fest, ob die Verbindungszeichenfolge in der Anwendungseinstellungsdatei gespeichert werden soll

- Bestimmen Sie im nächsten Schritt (siehe Abbildung 32.24), ob die Verbindungszeichenfolge in den Einstellungen des Projektes gespeichert werden soll. Für dieses Beispiel habe ich mich dagegen entschieden. Mehr zu Projekteinstellungen erfahren Sie übrigens in ► Kapitel 25, aber auch ► Kapitel 3 und ► Kapitel 26 liefern hierzu wertvolle Tipps.
- Klicken Sie anschließend auf *Weiter*.

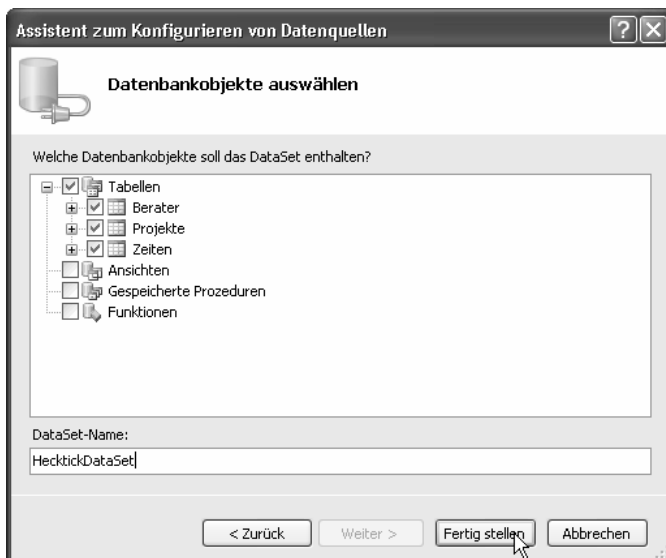


Abbildung 32.25: In diesem Assistentenschritt bestimmen Sie, aus welchen Tabellen *DataTable*-Objekt-Definitionen in das typisierte *DataSet* einfließen sollen

- Im letzten Schritt des Assistenten, den Sie auch in Abbildung 32.25 sehen können, definieren Sie, auf welche Tabellen innerhalb des typisierten *DataSet* Sie zugreifen wollen

HINWEIS: Sie könnten natürlich für jede Tabelle auch ein eigenes DataSet erstellen, doch weder ist es nötig, diesen Aufwand zu betreiben, noch würde es Sinn ergeben. Wenn Sie später nämlich Relationen zwischen den verschiedenen DataTable-Objekten herstellen wollen, *sollten* Sie die verschiedenen Tabellen (oder auch Ansichten, bzw. *Resultsets*, die aus gespeicherten Prozeduren hervorgehen) sogar unter einem DataSet zusammenfassen.

- Klicken Sie schließlich auf *Fertigstellen*.

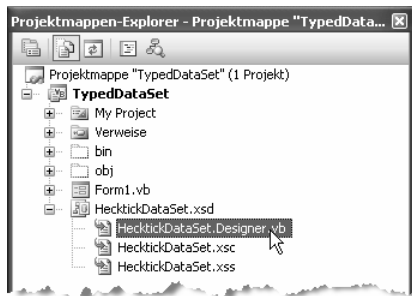


Abbildung 32.26: Das typisierte DataSet befindet sich anschließend im Projektmappen-Explorer

Grundsätzliches zu typisierten DataSets

Im Projektmappen-Explorer finden Sie anschließend eine neue Datei namens *HecktickDataSet.xsd*. Wenn Sie sich alle Dateien des Projektes durch Mausklick auf das entsprechende Symbol im Projektmappen-Explorer anzeigen lassen, sehen Sie, dass es – ähnlich wie bei Formularen – weitere Dateien zu dieser DataSet-Definition gibt. Eine davon nennt sich *HeckTickDataSet.Designer.vb*, und diese Visual Basic-Codedatei enthält den Code, der vom Designer erstellt wurde und eine spezielle Klasse darstellt, der die *HeckTickDataSet*-Klasse auf Basis der DataSet-Klasse typisiert.

Lassen Sie uns an einem Beispiel anschauen, was Ihnen dieser Code im Detail beim Entwickeln für Vorteile bringt.

In den vergangenen Abschnitten haben Sie schon kennen gelernt, welche Schritte notwendig sind, um eine Verbindung zu Datenbank aufzubauen, einer Tabelle dieser Datenbank zwei Datenzeilen hinzuzufügen und anschließend eine dieser neu hinzugefügten Zeilen zu verändern.

- Fügen Sie eine Schaltfläche in das Formular ein. Definieren Sie *Test* als Schaltflächenbeschriftung und nennen Sie die Schaltfläche *btnTest*.
- Doppelklicken Sie auf die Schaltfläche, um den Codeeditor zu öffnen und die Ereignisbehandlungsprozedur für das *Click*-Ereignis auszuformulieren

BEGLEITDATEIEN: Der Einfachheit halber können Sie eine vorbereitete Textdatei mit den benötigten Codezeilen verwenden, die Sie unter dem Namen *TypedDataSet.txt* im Verzeichnis *.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32* finden.

Der Code, für diese Prozedur, sieht folgendermaßen aus:

```
Public Class Form1
    Private Sub btnTest_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnTest.Click
```

```

'Die Definitionen für typisierte Tabellen befinden sich in der DataSet-Klasse
Dim locProjekteTable As New HecktickDataSet.ProjekteDataTable

'Die Definitionen für typisierte Adapter für Tabellen befinden sich
'in einem separaten Namespace
Dim locProjekteAdapter As New HecktickDataSetTableAdapters.ProjekteTableAdapter

'Eine neue Datenzeile hinzufügen. IntelliSense hilft Ihnen hier beim
'Finden der richtigen Parameter, weil diese Datenzeile jetzt typisiert ist.
locProjekteTable.AddProjektRow("Orcas", " (VS 2007) Schreiben des Entwicklerbuchs", _
    #2/1/2007#, #12/31/2007#, "Büro")

'Eine typisierte Datenzeile deklarieren.
Dim locProjektRow As HecktickDataSet.ProjekteRow

'Vorteil: AddProjektRow liefert die hinzugefügte Zeile als typisierte Row zurück.
locProjektRow = locProjekteTable.AddProjektRow("Katmai", _
    "(SQL Server 2005) Schreiben des Crash-Kurs", _
    #3/1/2007#, #1/31/2008#, "Büro")

'Aktualisieren funktioniert mit dem Projekte-TableAdapter.
locProjekteAdapter.Update(locProjekteTable)

'Bei AutoIncrement-Feldern wird automatisch dafür gesorgt,
'dass das Feld aktualisiert wird.
Dim locIDProjekte As Integer = locProjektRow.IDProjekte

MessageBox.Show("Datensätze wurden hinzugefügt. Sie können Sie im Server-Explorer betrachten!")

'Daten komplett neu einlesen
locProjekteTable.Clear()

'Funktioniert wie beim Adapter.
locProjekteAdapter.Fill(locProjekteTable)

'Die Zeile mit der entsprechenden ID finden
locProjektRow = locProjekteTable.FindByIDProjekte(locIDProjekte)

'Jetzt können wir die Datenfelder direkt per Eigenschaftennamen ändern,
'die genau so heißen, wie die Datenfelder in der Datenbank.
locProjektRow.Projektbeschreibung = "(SQL Server 2007) Schreiben des Crash-Kurses"
locProjektRow.Startzeitpunkt = #2/1/2007#
locProjektRow.Endzeitpunkt = #10/31/2007#
locProjektRow.Ausführungsort = "Ruprechts Büro"

'Tabelle aktualisieren.
locProjekteAdapter.Update(locProjekteTable)
End Sub
End Class

```

Zusammenspiel zwischen TableAdapter und typisierter Table-Klasse

Im vorherigen Beispiellisting können Sie gut das Zusammenspiel zwischen der typisierten DataTable einer typisierten DataSet-Klasse und dem so genannten TableAdapter-Objekt erkennen. Ein TableAdapter ist eine Klasse, nach deren Basistyp Sie vergeblich in der .NET-Framework-Bibliothek suchen. Diese Klasse wird nämlich komplett vom DataSet-Designer erstellt, sie wird nicht von *DataAdapter* abgeleitet, ersetzt aber, neben anderen Funktionen, die sie implementiert, die Funktionalität der *DataAdapter*-Klasse, die Sie in den vorherigen Beispielen kennen gelernt haben, vollständig.

So bleibt die prinzipielle Vorgehensweise hinter den Kulissen für typisierte DataTable-Objekte gleich, denn:

- Für jede Zeile, die Sie einer typisierten DataTable (ProjekteDataTable im Beispiel) hinzufügen, kapselt die entsprechende TableAdapter-Klasse (ProjekteTableAdapter im Beispiel) eine SQL-INSERT-Abfrage. Für das Aktualisieren von Zeilen wird eine entsprechende UPDATE-Anweisung und für das Löschen eine INSERT-Anweisung jeweils mit den entsprechenden Parameters-Auflistungen generiert.
- Um eine typisierte DataTable mit Daten zu füllen, verwenden Sie die FillBy-Methode des entsprechenden TableAdapter.
- Änderungen führen Sie, wie im vorherigen Beispiel zu sehen, an der typisierten DataTable durch. Sie können mit *AddTabellenameRow* eine neue Zeile einer Tabelle hinzufügen, Zeilen mit *RemoveTabellenameRow* aus der Tabelle entfernen oder direkt durch typischere Eigenschaften auf die Datenfelder eines Datensatzes zugreifen, der durch ein entsprechendes *TabellenameRow*-Objekt repräsentiert wird (*Tabellename* steht dabei jeweils für den Namensausschnitt im typisierten Element, zum Beispiel *ProjekteRow* für eine Datenzeile der *Projekte*-Tabelle im Beispiel).
- Um die Änderungen an die Datenbank zu übermitteln, verwenden Sie die Update-Methode des jeweiligen TableAdapter-Objektes.
- Verfügte die Ausgangstabelle über einen primären Schlüssel, befindet sich im jeweiligen *TabellenameTableAdapter* eine Methode namens *FindByIdPrimärSchlüsselname*, die ein *TabellenameRow*-Objekt zurückliefert. Im Beispiel gibt es die Methode *FindByIdProjekte*, die versucht eine Zeile mit der entsprechenden ID innerhalb einer *ProjekteDataTable*-Instanz zu finden, und diese Zeile als *ProjekteRow*-Objekt zurückliefert.

TIPP: Sie können den jeweiligen TableAdapter um weitere Abfrage-Methode erweitern, die jeweils auf neuen SQL-SELECT-Abfragen basieren. Das Beste: Diese Abfragen lassen sich interaktiv mit dem Designer erstellen. Wie das geht, zeigt der folgende Abschnitt.

Erweitern des TableAdapters um zusätzliche Abfragefunktionen, denen Parameter übergeben werden können

Standardmäßig implementiert ein TableAdapter, der lediglich aus einer Tabellendefinition hervorgegangen ist, eine simple SELECT-Abfrage an die Datenbank, die *alle* Felder und Daten einer Tabelle zurückliefert. Für komplexere Business-Anwendungen reicht das natürlich nicht aus; hier müssen Tabellenselektionen oftmals viel individueller und flexibler von statten gehen. Bedenken Sie einmal, was passieren würde, wenn Sie mit in einem großen Konzern alle Daten der Buchungstabelle mit SELECT abrufen würden: Mehrere Millionen Datenzeilen würden übertragen. In der Regel benötigt

man hier auch Parameter, die einer Abfragefunktion übergeben werden müssen, um entsprechende Datensätze aus der Datenbank zu ermitteln, die gemäß der übergebenen Parameter in irgendeiner Form gefiltert wurden.

Anstatt nun selbst Hand anzulegen, und komplizierte SELECT-Abfragen zu formulieren, verwenden Sie abermals den Designer. Wie das geht, zeigt die folgende Schritt-für-Schritt-Anleitung, die auf dem bereits erstellten Projekt des vorherigen Abschnitts aufsetzt.

- Zu diesem Zweck schalten Sie in die Designer-Ansicht des DataSet-Objektes, indem Sie im Projektmappen-Explorer einen Doppelklick auf den Wurzelzweig des DataSets ausführen.

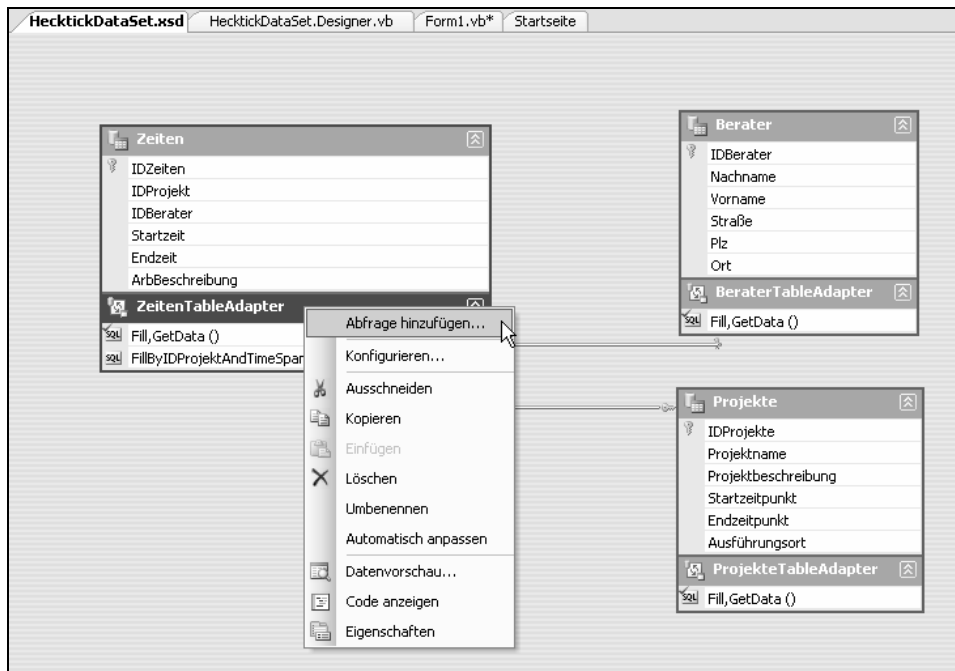


Abbildung 32.27: Sie fügen einem *TableAdapter* eine neue Abfragefunktion hinzu, indem Sie das Kontextmenü über der *TableAdapter*-Titelzeile öffnen und den Menübefehl *Abfrage hinzufügen* wählen

- Fahren Sie mit der Maus auf die Titelzeile eines Tabellen-Adapters (nicht der Tabelle!), öffnen Sie das Kontextmenü per Rechtsklick, und wählen Sie den Menübefehl *Abfrage hinzufügen*. Für dieses Beispiel wählen Sie bitte den *ZeitenTableAdapter*.
- Sie befinden sich anschließend im *Konfigurationsassistenten für TableAdapter-Abfragen*, dessen Einstellungen Sie auf den ersten beiden Seiten belassen, wie sie sind, und mit zweimaligem *Weiter* zur übernächsten Seite wechseln.
- Auf der Seite, die jetzt erscheint, klicken Sie auf *Abfragegenerator*. Sie sehen anschließend einen Dialog, etwa wie in *Abbildung 32.28*.

Das Ziel ist es jetzt, mit dem Abfragegenerator die Basis für eine neue *TableAdapter*-Funktion zu schaffen, die Datensätze der Zeitentabelle der Beispieldatenbank zurückliefert, die nur für ein be-

stimmtes Projekt und innerhalb eines bestimmten Zeitbereichs getätigt wurden. Die Tabelle soll ferner nach den Beraternachnamen und der Anfangszeit sortiert werden.

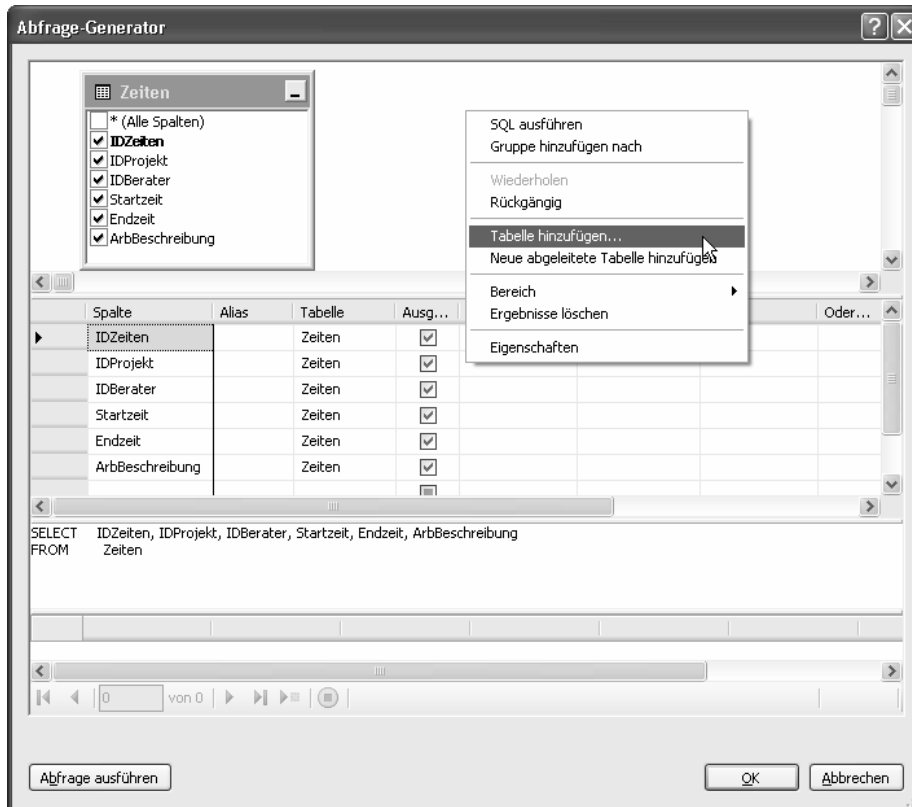


Abbildung 32.28: Mit dem Abfragegenerator können Sie *SELECT*-Abfragen unter anderem für neue Abfragefunktionen des *TableAdapter* erstellen

WICHTIG: Wenn Sie eine Abfrage für eine neue *TableAdapter*-Funktion erstellen, achten Sie darauf, das Schema des vorhandenen *TableAdapters* nicht zu erweitern. Sie dürfen also – wie es beispielsweise bei *JOIN*-Abfragen passiert – keine zusätzlichen Felder hinzufügen. Das bedeutet aber nicht, dass Sie keine *JOIN*-Abfragen verwenden dürfen, um beispielsweise nach Feldern in verknüpften Tabellen zu selektieren oder sortieren, wie das folgende Beispiel zeigt.

- Da wir für dieses Beispiel das Resultset nach dem Berater-Nachnamen sortieren lassen wollen, dieser aber über die Zeitentabelle selbst nicht verfügbar ist, müssen wir einen *INNER JOIN* erstellen, bei dem *IDBerater* der Zeitentabelle als Indikator auf den eigentlichen Datensatz des Beraters in der Beratertabelle herhält. Neben der standardmäßig schon angezeigten Zeitentabelle benötigen wir also die Beratertabelle, um dort das benötigte Feld auszuwählen und die *JOIN*-Verknüpfung herzustellen. Dazu wählen Sie aus dem Kontextmenü, das Sie über dem freien Tabellenbereich öffnen, den Menübefehl *Tabelle hinzufügen*.
- Wählen Sie aus dem Dialog die Beratertabelle aus.

- Setzen Sie nun in der Beratertabellendefinition im oberen Bereich des Dialogs für das Feld Nachname ein Häkchen. In diesem Moment erscheint das entsprechende Datenfeld auch in der Datenfeldliste darunter. In der Spalte Ausgabe dieser Datenfeldliste entfernen Sie das Häkchen anschließend wieder, sodass sich anschließend ein Bild ergibt, das Sie auch in Abbildung 32.29 sehen.

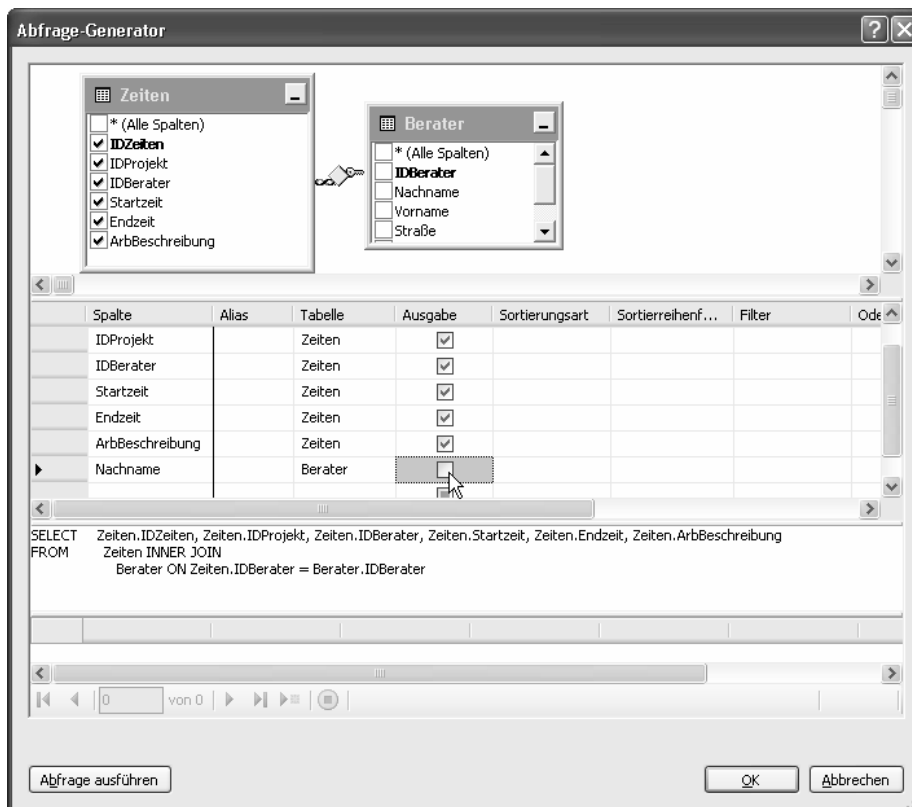


Abbildung 32.29: Um ein Feld einer anderen Tabelle nur für Selektionszwecke verfügbar zu machen, ist es am einfachsten, es in der Tabellendefinition kurz anzuklicken, um es der Datenfeldliste hinzuzufügen, seine Ausgabe in der Datenfeldliste aber anschließend zu unterdrücken

- Nun bestimmen Sie die entsprechenden Filter und Sortierkriterien. Da nach dem Feld *Nachname* sortiert werden soll, klicken Sie in der entsprechenden Zeile in die Zelle der Spalte *Sortierungsart*.

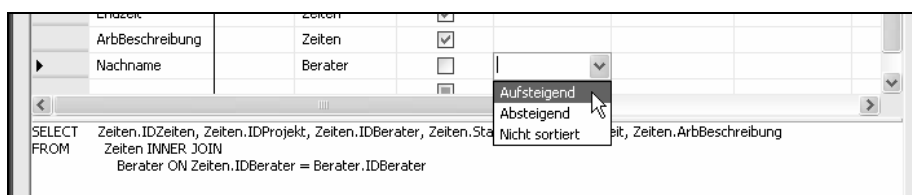


Abbildung 32.30: Bestimmen Sie die Sortierungsart eines Feldes mit der Aufklappliste in der jeweiligen Datenfeldlistenzeile

- Wählen Sie aus der Aufklappliste, die jetzt erscheint und sich öffnet, den Eintrag *Aufsteigend*.
- Bestimmen Sie als nächstes die zu übergebenden Parameter, indem Sie mit @ beginnende Platzhalter für die Selektionskriterien verwenden. Möchten Sie beispielsweise, dass Sie der zu generierenden Funktion später einen Parameter IDProjekt zur Selektion der Zeitdaten nur nach bestimmten Projekten übergeben können, definieren Sie in der Datenfeldlistenzeile IDProjekt den Ausdruck =@IDProjekt als Filterkriterium.

HINWEIS: Diese Syntax gilt übrigens für den SQL Server. In anderen Namespaces wie OleDb oder Oracle werden Parameter teilweise durch »?« oder andere Sonderzeichen notiert. Schauen Sie dann einfach in Dokumentation, das Prinzip ist das Gleiche.

- Für die Parameter Startzeit und Endzeit, die den auszuwertenden Zeitbereich definieren sollen, verfahren Sie äquivalent. Orientieren Sie sich dabei am besten an Abbildung 32.31.

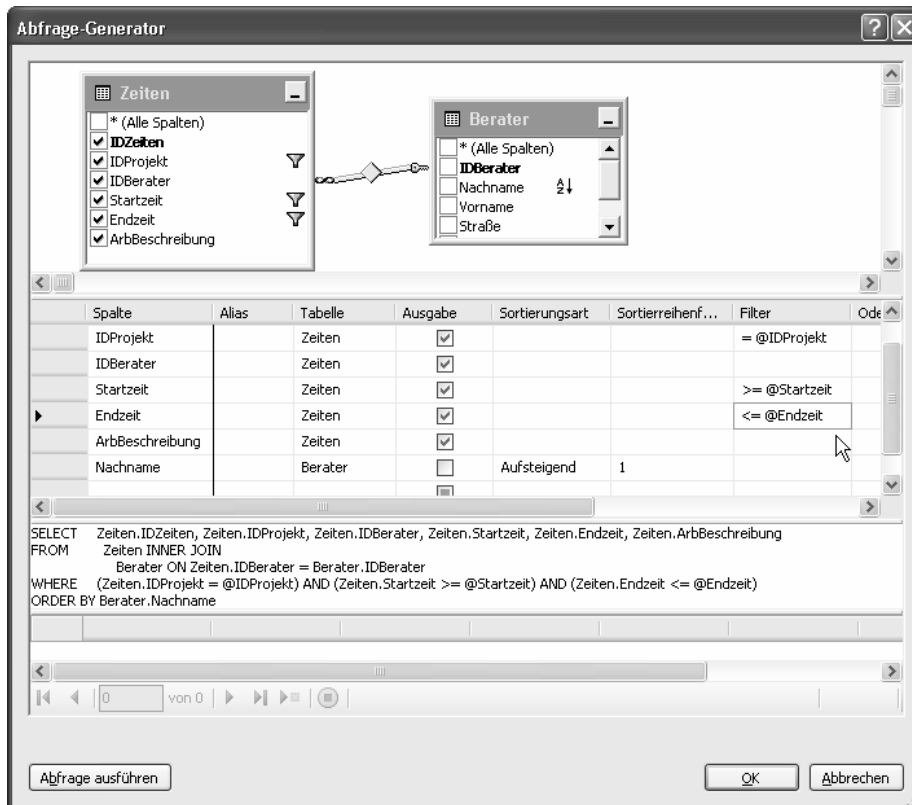


Abbildung 32.31: So definieren Sie die Filterkriterien mit der Angabe von Variablen, die später in Funktionsparameter umgewandelt werden

- Und das war es auch schon! Klicken Sie auf *OK*, um den Abfrage-Generator zu verlassen.
- Klicken Sie auf *Weiter*, um zum letzten Assistentenschritt zu gelangen.

- Schließlich bestimmen Sie noch, so wie in Abbildung 32.32 zu sehen, die Funktionsnamen, die der DataSet-Designer für den *TableAdapter* generieren soll. Für unser Beispiel verwenden Sie die Namen *FillByIDProjektUndZeitbereich* sowie *GetDataByIDProjektUndZeitbereich*.

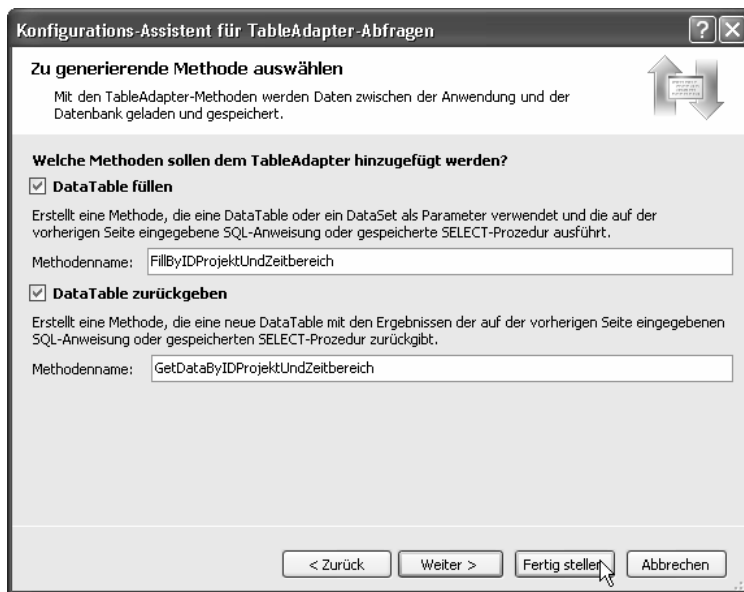


Abbildung 32.32: Bestimmen Sie schließlich noch den Namen der *TableAdapter*-Funktionen

- Klicken Sie anschließend auf *Fertig stellen*, um den Assistenten abzuschließen.
- Wechseln Sie anschließend zur Design-Ansicht von *Form1.vb*.
- Fügen Sie im Formular eine *Test2*-Schaltfläche namens *btnTest2* ein.
- Doppelklicken Sie auf die Schaltfläche, um das Codefenster zu öffnen, und den Rumpf der entsprechenden Ereignisbehandlungsprozedur einzufügen.

BEGLEITDATEIEN: Der Einfachheit halber können Sie eine vorbereitete Textdatei mit den benötigten Codezeilen verwenden, die Sie unter dem Namen *TypedDataSet2.txt* im Verzeichnis *.\VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32* finden.

```
Private Sub btnTest2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnTest2.Click

    'Wir benötigen alle Adapter, um an die Daten heranzukommen.
    Dim locZeitenAdapter As New HecktickDataSetTableAdapters.ZeitenTableAdapter
    Dim locProjekteAdapter As New HecktickDataSetTableAdapters.ProjekteTableAdapter
    Dim locBeraterAdapter As New HecktickDataSetTableAdapters.BeraterTableAdapter

    'Dieses Mal verwenden wir wegen der Relation ein DataSet
    Dim locHeckTickDataSet As New HecktickDataSet()
    locProjekteAdapter.Fill(locHeckTickDataSet.Projekte)
    locBeraterAdapter.Fill(locHeckTickDataSet.Berater)
```

```
'TODO: Passen Sie den Zeitbereich Ihrer Beispieldatenbank an.
locZeitenAdapter.FillByIDProjektUndZeitbereich(locHeckTickDataSet.Zeiten, _
    locHeckTickDataSet.Projekte(0).IDProjekte, _
    #3/15/2006#, #3/15/2006 11:59:59 PM#)

'Alle Datensätze mit Nachnamen ausgeben; die Relation zwischen Zeiten und
'Berater wurde einerseits durch die Verwendung eines typisierten DataSets
'und andererseits durch die Erstellung der Relationen der Tabellen im SQL Server
'automatisch hergestellt.
For Each locZeitenRow As HecktickDataSet.ZeitenRow In locHeckTickDataSet.Zeiten.Rows
    Debug.Print(locZeitenRow.BeraterRow.Nachname & ": " & _
        locZeitenRow.Startzeit.ToShortDateString & " - " & _
        locZeitenRow.Endzeit.ToShortDateString)
Next

End Sub
```

Sie sehen im Codelisting, dass die *TableAdapter*-Funktion, die wir gerade erstellt haben, nun am *ZeitenTableAdapter* zur Verfügung steht.

Wenn Sie das Beispiel starten und auf die neue Schaltfläche klicken, sehen Sie folgende Ausgabe im Ausgabefenster.¹⁰

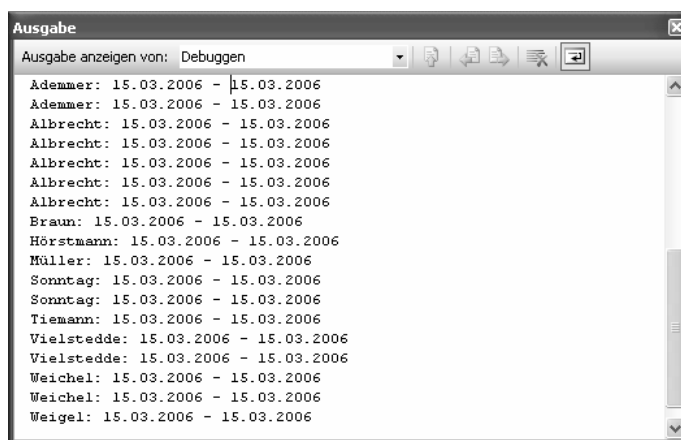


Abbildung 32.33: Das Programm verwendet die neue Funktion des *TableAdapter*

Erstellen von datenbankgebundenen Formularen in Windows Forms-Anwendungen

Das Repertoire des Designers in Sachen interaktives Anwendungsdesign ist damit allerdings noch lange nicht erschöpft.

¹⁰ Um eine Anzeige im Ausgabefenster zu bekommen, muss im Optionsdialog von Visual Studio unter Umständen im Bereich *Debuggen/Allgemein* die Option *Gesamten Text aus Ausgabefenster an Direktfenster umleiten* ausgeschaltet sein.

Die folgende Anleitung zeigt, wie Sie auf Basis der Anwendung, die wir bislang entwickelt haben, unser Formular, das sich ja bislang noch nicht mit übermäßigem Funktionsreichtum rühmen kann, in Nullkommanichts zu einem halbwegs brauchbaren Abfragewerkzeug von Projektzeiten ausbauen können.

- Zu diesem Zweck wechseln Sie in die Designer-Ansicht des Formulars.
- Wählen Sie aus dem Menü *Daten* den Menübefehl *Datenquellen anzeigen* aus.
- Öffnen Sie den Zweig vor *HeckTickDataSet*, den vor *Projekte* und schließlich den *Projekte* untergeordneten Zweig *Zeiten*. Orientieren Sie sich dabei am besten an Abbildung 32.34. Achten Sie darauf, dass es zwei *Zeiten*-Elemente gibt. Verwenden Sie im Folgenden den *Projekte* untergeordneten Zweig und nicht den am Ende der Liste stehenden.
- Öffnen Sie die Aufklappliste von *Projekte*.

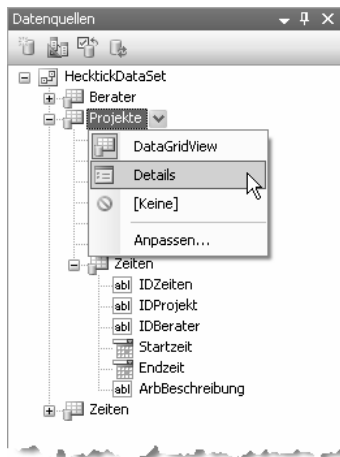


Abbildung 32.34: Erweitern Sie die entsprechenden Zweige im Datenquellenfenster und öffnen Sie die Aufklappliste im Projekte-Element

- Wählen Sie aus dem Menü den Menüpunkt *Details*.

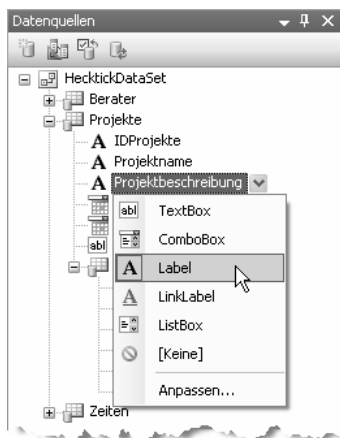


Abbildung 32.35: Wählen Sie anschließend die Steuerelemententsprechungen jedes Datenfeldes aus den Aufklapplisten aus

- Öffnen Sie jede Aufklappliste, die sich hinter *IDProjekte*, *Projektname* und *Projektbeschreibung* befindet, und stellen Sie alle zugeordneten Steuerelemente für die Datenwiedergabe im Formular auf *Label*. Orientieren Sie sich an Abbildung 32.35.
- Nun ziehen das Feld *IDProjekte* vom Datenquellen-Fenster in das Formular, und beobachten Sie, was passiert.

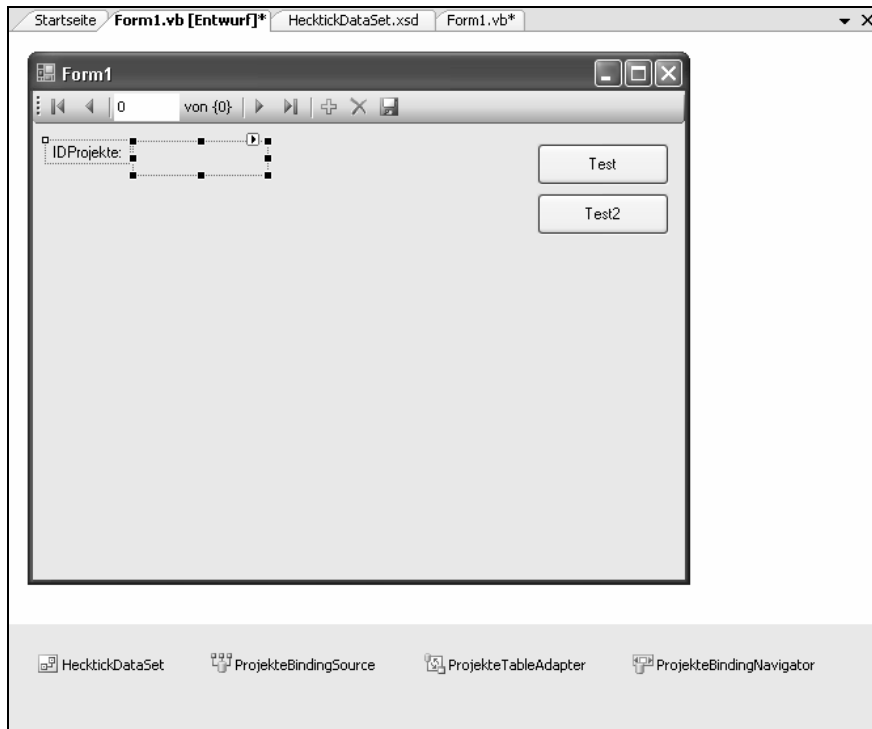


Abbildung 32.36: Nach dem Einfügen des ersten Feldes aus dem Datenquellen-Fenster hat der Designer automatisch eine Navigationsleiste und die entsprechenden Komponenten eingefügt, um die Datenbindung funktionsfähig zu machen

Mit dem letzten Schritt hat der Designer das *HeckTickDataSet* dem Komponentenfach hinzugefügt. Ferner hat er eine *ProjekteBindingSource*, einen entsprechenden *ProjekteTableAdapter* sowie einen *ProjekteBindingNavigator* (das ist das Steuerelement mit den Schaltflächen im Stile eines Videorekorders) dem Komponentenfach bzw. Formular hinzugefügt, und damit die komplette Funktionalität zum Browsen von Daten einer bestimmten Datenquelle implementiert. Sie können das Projekt zu diesem Zeitpunkt schon starten und werden feststellen, dass Sie bereits Zugriff auf die *Projekte*-Datentabelle nehmen können – auch wenn sich das Blättern in den Daten mit dem *BindingNavigator* zur Zeit nur im Aktualisieren der *Projekte-ID* bemerkbar macht.

Die *ProjekteBindingSource* wird hier verwendet, um die Bindung der Datenquelle (die *Projekte*-Tabelle) an bestimmte Eigenschaften von Steuerelementen im Formular zu ermöglichen. So wurde mit der letzten *Drag&Drop*-Aktion im Beispiel die *Text*-Eigenschaft des *Label*-Steuerelements an das Datenfeld *IDProjekte* gebunden. (Mehr zum Thema *BindingSource* erfahren Sie übrigens in ► Kapitel 27.)

- Fügen Sie auf diese Weise die Felder *Projektname* und *Projektbeschreibung* ebenfalls im Formular ein.
- Vergrößern oder verkleinern Sie die Felder bzw. Beschriftungen ein wenig, sodass sie übersichtlich im Formular angeordnet sind.
- Öffnen Sie anschließend die Aufklappliste des untergeordneten *Zeiten*-Zweigs.

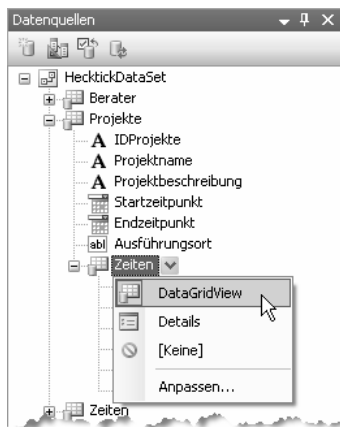


Abbildung 32.37: Öffnen Sie die Aufklappliste des *Zeiten*-Elements

- Wählen Sie *DataGridView*.

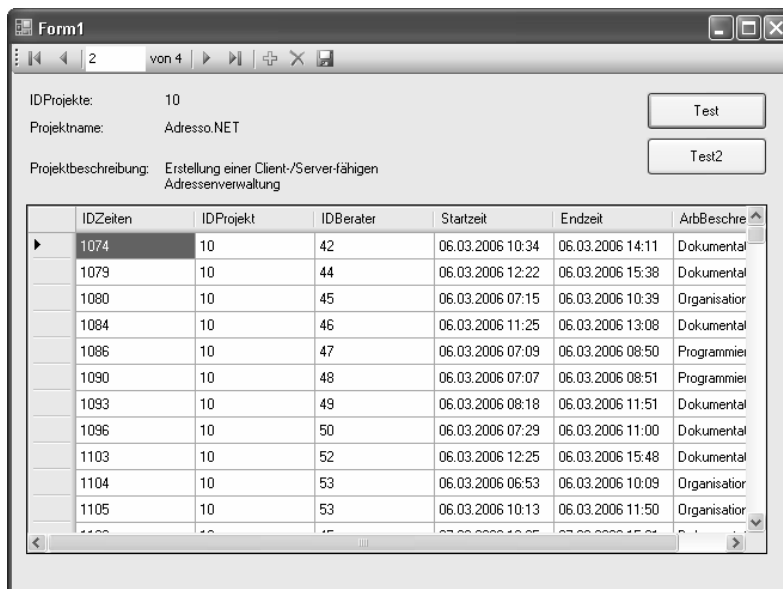


Abbildung 32.38: Das Ergebnis nach ein paar Minuten im Designer kann sich sehen lassen. Und das Beste ist: Sie haben dazu nicht eine einzige Zeile Code programmieren müssen!

- Verschieben Sie nun per Drag&Drop den kompletten Zeitzweig in das Formular. Wieder wird der Designer aktiv, und fügt zusätzliche Komponenten in das Komponentenfach ein – dieses Mal um eine so genannte Master/Details-Ansicht zwischen Label-Feldern und DataGridView zu realisieren. Wenn Sie das Programm anschließend starten, dann verstehen Sie, was damit gemeint ist: Sie blättern mit dem BindingNavigator durch die Datensätze der *Projekte*-Tabelle (*Master*). Gleichzeitig zeigen Sie in der DataGridView alle Zeitendatensätze an, die zum gerade »aufgeschlagenen« Projekte-Datensatz gehören (*Details* – siehe Abbildung 32.38)

Und so geht es weiter

Im Rahmen dieses Buches kann ein Kapitel, das ein Thema wie ADO.NET behandelt, allenfalls die Grundlagen klären. Das Thema ADO.NET liefert alleine Stoff für mehrere Kapitel Bücher, ja sogar ganze Bücher und Buchfortsetzungen. Das liegt nicht nur daran, dass das Framework mit ADO.NET eine extrem umfangreiche Klassenbibliothek zur Verfügung stellt, sondern dass ein Entwickler sich auch intensiv mit der SQL-Abfragesprache und dem SQL Server 2005 selbst auseinander setzen muss, um professionelle Datenbankanwendungen erstellen zu können.

Ihr Wissensstand nach der Lektüre dieses Kapitels sollte aber in jedem Fall ausreichend sein, um erste Datenbankapplikationen sicher entwickeln zu können. In vielen Fällen wissen Sie schon jetzt mehr, als viele VB6-Entwickler jemals zu »alten« ADO-Zeiten gewusst haben.

Sie haben gelernt, wie Sie »manuell« mit den ADO.NET-Komponenten umgehen müssen, und Sie haben erfahren, wie Sie Formulare auf einfache Weise und ohne Code zu programmieren zur Datenanzeige verwenden können. Das Geheimnis einer Anwendung liegt sicherlich in einer Mischung aus beidem, und es gibt viele weitere Aspekte von ADO.NET, die es zu ergründen gilt, und die Ihnen schließlich helfen, Ihre Meisteranwendung effizient und robust zu entwickeln.

Die Online-Hilfe von Visual Studio .NET stellt für weitere Nachforschungen zu diesem Thema eine großartige Wissensquelle dar, und letzten Endes wird Ihnen auch das Internet einmal mehr helfen, interessante Artikel zu den Themen zu finden.

Und schließlich haben wir auch noch die versprochene dokumentierte Beispielanwendung für Sie parat (siehe Abbildung 32.39), die mithilfe der Techniken entstanden ist, die Sie in diesem Kapitel kennen gelernt haben.

BEGLEITDATEIEN: Sie finden das fertige Projektbeispiel im Verzeichnis \VB 2005 - Entwicklerbuch\G - Smart-Client\Kap32\HeckTick\HeckTick.sln. Und – soviel Kalauer muss sein: Schauen Sie sich es in Ruhe an – lassen Sie dabei keine HeckTick aufkommen!

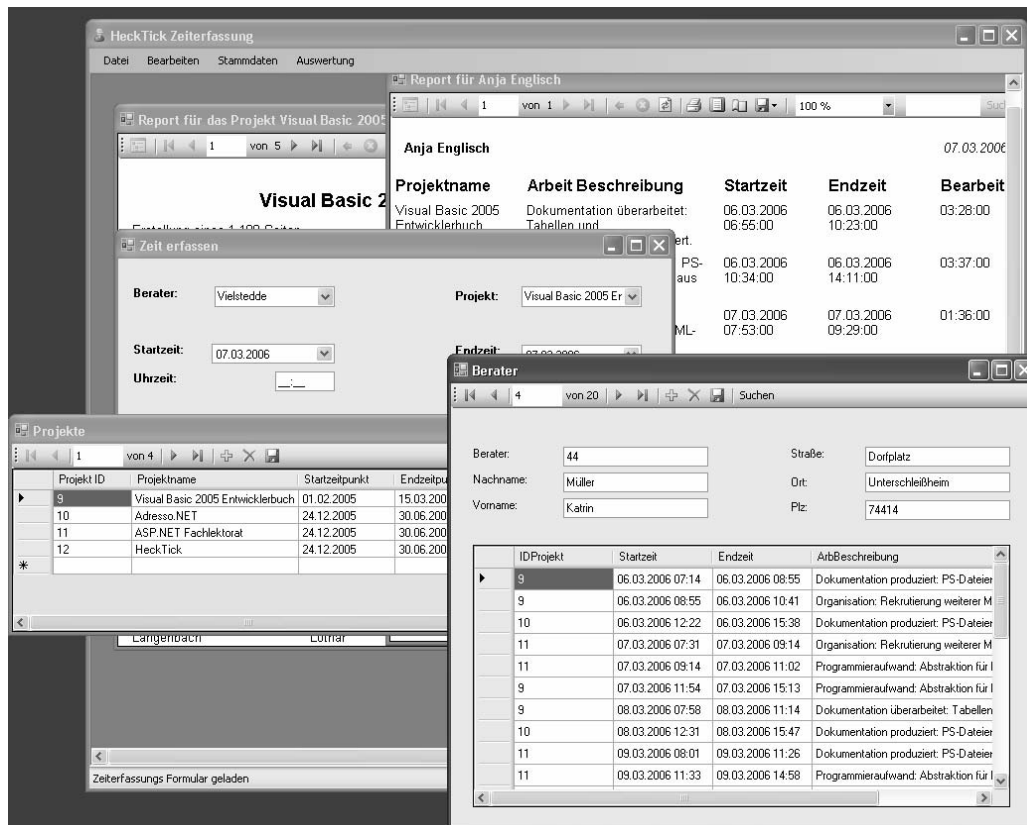


Abbildung 32.39: Hier kommt HeckTick auf!

TIPP: Falls Sie sich dazu entschließen, sich wirklich intensiv mit dem Thema zu beschäftigen, sollten Sie den Erwerb weiterer Bücher zum Thema in Erwägung ziehen. Ich selbst möchte Ihnen an dieser Stelle ein Buch empfehlen, das sich mit dem Thema SQL Server 2005 beschäftigt. Sein Titel: *SQL Server 2005, Konfiguration, Administration, Programmierung, 2. Auflage* (Microsoft Press, ISBN: 3-86063-997-8). Mit dem Autorenteam *Ruprecht Dröge* und *Markus Raatz* haben sich zwei professionelle SQL Server-Experten gefunden und ein rundum gutes und informatives Werk geschaffen.