Microsoft

Microsoft®

ADO.NET 4

Tim Patrick

online book + practice files

Step by Step

Microsoft®

Microsoft® ADO.NET 4 Step by Step

Tim Patrick

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 WCT 5 4 3 2 1 0

Acquisitions and Development Editor: Russell Jones
Production Editor: Kristen Borg
Production Services: Octal Publishing, Inc.
Technical Reviewer: Sahil Malik
Indexing: Potomac Indexing, LLC
Cover: Karen Montgomery
Compositor: Susan Brown
Illustrator: Robert Romano

978-0-735-63888-4

To Abel Chan, a good friend and a good programmer.

Contents at a Glance

Part I **Getting to Know ADO.NET**

Chapter 1
Introducing ADO.NET 4 3

Chapter 2
Building Tables of Data 17

Chapter 3
Storing Data in Memory 37

Chapter 4
Accessing the Right Data Values 59

Chapter 5
Bringing Related Data Together 73

Chapter 6
Turning Data into Information 89

Chapter 7
Saving and Restoring Data 107

Part II **Connecting to External Data Sources**

Chapter 8
Establishing External Connections 121

Chapter 9
Querying Databases 135

Chapter 10
Adding Standards to Queries 153

Chapter 11
Making External Data Available Locally 169

Chapter 12
Guaranteeing Data Integrity 191

Part III **Entity Framework**

Chapter 13
Introducing the Entity Framework 213

Chapter 14
Visualizing Data Models 225

Chapter 15
Querying Data in the Framework 245

Chapter 16
Understanding Entities Through Objects 267

Part IV **LINQ**

Chapter 17
Introducing LINQ 289

Chapter 18
Using LINQ to DataSet 305

Chapter 19
Using LINQ to Entities 315

Chapter 20
Using LINQ to SQL 331

Part V **Providing RESTful Services with WCF Data Services**

Chapter 21
Binding Data with ADO.NET 347

Chapter 22
Providing RESTful Services with WCF Data Services 369

Table of Contents

Acknowledgments xv

Introduction xvii

Part I Getting to Know ADO.NET

1 Introducing ADO.NET 4 3

- What Is ADO.NET? 3
- Why ADO.NET? 5
- Major Components of ADO.NET 5
- Extensions to ADO.NET 7
- Connecting to External Data 8
- Summary 15
- Chapter 1 Quick Reference 16

2 Building Tables of Data 17

- Implementing Tables 17
  - Logical and Physical Table Implementations 17
  - The *DataTable* Class 18
- Adding Data Columns 21
- Dataset Designer 27
- Summary 34
- Chapter 2 Quick Reference 35

3 Storing Data in Memory 37

- Adding Data 37
  - Creating New Rows 37
  - Defining Row Values 38
  - Storing Rows in a Table 40

Examining and Changing Data 42
Removing Data 45
Batch Processing 46
Row State 47
Row Versions 48
Validating Changes 49
Exception-Based Errors 50
Validation-Based Errors 51
Summary 56
Chapter 3 Quick Reference 57

4 Accessing the Right Data Values 59
Querying and Sorting Data 59
Finding Rows by Primary Key 60
Selecting Rows with a Search Criteria 62
Sorting Search Results 64
Performing Case-Sensitive Lookups 67
Using Expression Columns 67
Summary 71
Chapter 4 Quick Reference 71

5 Bringing Related Data Together 73
Collecting Tables into Sets 73
Establishing Relationships Between Tables 76
Understanding Table Relations 76
Creating Data Relations 78
Locating Parent and Child Records 79
Defining Table Constraints 81
Summary 87
Chapter 5 Quick Reference 88

6 Turning Data into Information 89
Aggregating Data 89
Generating a Single Aggregate 91
Adding an Aggregate Column 94
Aggregating Data Across Related Tables 95
Referencing Parent Fields in Expressions 98

Setting Up Indexed Views 98
Creating a *DataView* *99*
Using a *DataView* 101
Summary 106
Chapter 6 Quick Reference 106

7 Saving and Restoring Data 107
Serializing *DataSet* and *DataTable* Objects 107
Writing XML 108
Reading XML 110
Guiding XML Generation 111
Identifying Namespaces 111
Nesting Child Tables 113
Managing and Positioning Columns 113
Summary 117
Chapter 7 Quick Reference 118

Part II Connecting to External Data Sources

8 Establishing External Connections 121
Using Connection Strings 121
SQL Server Connection Strings 122
OLE DB and ODBC Connection Strings 124
Connection String Builders 124
Storing Connection Strings 126
Understanding Data Providers 126
Connecting to SQL Server via a Data Provider 127
Creating and Opening Connections 128
Connection Pooling 132
Summary 133
Chapter 8 Quick Reference 133

9 Querying Databases 135
Processing SQL Queries 135
Creating Command Objects 136
Processing Queries 137
Processing Asynchronously 139

Returning Query Results 140
Returning a Single Value. 141
Returning Data Rows. 142
Accessing Field Values. 144
Processing More Complicated Results 146
Summary 150
Chapter 9 Quick Reference. 151

10 Adding Standards to Queries 153

Developing Parameterized Queries 153
Understanding the Need for Parameters 154
Implementing Standard Queries. 155
Using Parameters with Other Providers. 160
Using Parameters in Stored Procedures. 161
Summary 166
Chapter 10 Quick Reference. 167

11 Making External Data Available Locally 169

Understanding Data Adapters. 169
Moving Data from Source to Memory. 171
Moving Data into a *DataTable*. 171
Moving Data into a *DataSet*. 173
Moving Data from Memory to Source. 175
Configuring the Update Commands 175
Performing the Update. 179
Generating Update Commands Automatically. 180
Table and Column Mapping. 186
Summary 188
Chapter 11 Quick Reference. 189

12 Guaranteeing Data Integrity 191

Transactions and Concurrency. 191
Using Local Transactions. 195
Employing Savepoints. 202
Using Distributed Transactions 204
Summary 208
Chapter 12 Quick Reference. 209

Part III **Entity Framework**

13 Introducing the Entity Framework 213
Understanding the Entity Framework. 213
Defining the Entity Framework's Terms 215
Understanding the Entity Framework's Layers 216
Understanding the Conceptual Model 217
Understanding the Storage Model. 217
Understanding the Model Mappings. 218
Using the Entity Framework 218
Building the Model 218
Generating the Objects. 220
Instantiating the Context 221
Running Framework Queries 222
Summary 223
Chapter 13 Quick Reference. 223

14 Visualizing Data Models. 225
Designing an Entity Framework Model 225
Using the Entity Data Model Wizard 225
Entity Data Model Designer. 230
Working with the Mapping Details Panel 235
Using the Model Browser. 240
Managing the Object Layer 241
Summary 242
Chapter 14 Quick Reference. 243

15 Querying Data in the Framework. 245
Getting to Know Entity SQL 246
Writing Basic Queries 246
Using Literals, Operators, and Expressions 249
Grouping and Aggregating Entity Data. 252
Using Features Unique to Entity SQL 254
Running Entity SQL Queries 256
Running Queries Using an *ObjectQuery* 256
Running Queries Using a Provider 260
Summary 264
Chapter 15 Quick Reference. 265

16 Understanding Entities Through Objects 267

Managing Entity Data Through Objects 267

Accessing Entity Data Through Objects. 268

Modifying Entity Data Through Objects 271

Using Query Builder Methods 278

Queryable Extension Methods 283

Summary 285

Chapter 16 Quick Reference. 285

Part IV **LINQ**

17 Introducing LINQ 289

Getting to Know LINQ. 289

Using LINQ with .NET Objects 291

Starting a Query with the *From* Clause 293

Projecting Results with the *Select* Clause 293

Filtering Results with the *Where* Clause. 295

Sorting Results with the *Order By* Clause 296

Selecting Linked Results with the *Join* Keyword. 297

Limiting the Queried Content 299

Summarizing Data Using Aggregates 301

Applying Set Operations 302

Summary 303

Chapter 17 Quick Reference. 304

18 Using LINQ to DataSet. 305

Understanding the LINQ to DataSet Provider. 305

Writing Queries with LINQ to DataSet 306

Summary 312

Chapter 18 Quick Reference. 313

19 Using LINQ to Entities 315

Understanding the LINQ to Entities Provider 315

Writing Queries with LINQ to Entities. 316

Working with Entity and Database Functions. 321

Working with Custom Database Functions 324

Summary 329

Chapter 19 Quick Reference. 329

20 Using LINQ to SQL 331
Understanding the LINQ to SQL Provider 332
Comparing LINQ to SQL with LINQ to Entities 332
Understanding the Components of LINQ to SQL. 333
Using the Object Relational Designer. 335
Using Custom Database Functions in Queries 339
Summary 343
Chapter 20 Quick Reference. 343

Part V Presenting Data to the World

21 Binding Data with ADO.NET 347
Binding Data in Windows Forms 347
Creating Complex-Bound Applications 348
Creating Simple-Bound Applications 351
Understanding Windows Forms Data Binding 352
Binding Data in WPF 354
Creating Data-Bound WPF Applications 354
Understanding WPF Data Binding 360
Binding Data in ASP.NET. 362
Creating Data-Bound ASP.NET Applications. 362
Understanding ASP.NET Data Binding. 364
Summary 367
Chapter 21 Quick Reference. 367

22 Providing RESTful Services with WCF Data Services 369
Getting to Know the Service Layers 369
Introducing WCF Data Services. 369
Introducing REST 370
Setting Up a Data Service. 371
Defining Service Rights. 375
Accessing a Data Service using REST 377
Querying Entities with REST. 377
Updating Entities with REST. 381
Summary 384
Chapter 22 Quick Reference. 384

Index 385

Acknowledgments

An open-ended thank you goes to Microsoft, not only for developing some of the products that have kept me gainfully employed for nearly three decades, but for welcoming me into their book-writing fold. It was also a pleasure to work again with the team at O'Reilly Media, Microsoft's publishing partner. Editors Russell Jones and Kristen Borg kept all the trains running on time, which wasn't easy given the busy technical and publishing traffic. Rounding out the group were Meghan Blanchette, Sumita Mukherji, and Adam Witwer. Thank you all.

Sahil Malik, ADO.NET expert and fellow author, has the distinction of having read through every single word of this book looking for technical concerns. Nancy Sixsmith did the same for the mechanics of language, grammar, and consistency. The book is richer for their involvement.

Claudette Moore once again worked her agenting magic, somehow always managing to make everyone on both sides of a contract happy. This book would be nothing more than a series of discarded emails were it not for her hard work and dedication. Thank you, Claudette, for yet another adventure.

Thanks to all my friends at Harvest, especially fellow food and movie lovers Alice, Brenda, Andy, Suzy, Matt, Tiffany, Jeff, and Monica. Love and appreciation in heaps to my wife Maki and my son Spencer, both of whom exude patience and care. And thanks once again to God for making all these other acknowledgments possible in the first place.

Tim Patrick
October 2010

Introduction

ADO.NET is Microsoft's core data access library for .NET developers, and is the heart of many data-centric technologies on the Windows development platform. It works with C#, Visual Basic, and other .NET-enabled languages. If you are a .NET developer looking to interact with database content or other external data sources, then ADO.NET is the right tool for you.

Microsoft ADO.NET 4 Step by Step provides an organized walkthrough of the ADO.NET library and its associated technologies. The text is decidedly introductory; it discusses the basics of each covered system, with examples that provide a great head start on adding data features to your applications. While the book does not provide exhaustive coverage of every ADO.NET feature, it does offer essential guidance in using the key ADO.NET components.

In addition to its coverage of core ADO.NET library features, the book discusses the Entity Framework, the LINQ query system, and WCF Data Services. Beyond the explanatory content, each chapter includes step by step examples and downloadable sample projects that you can explore for yourself.

Who Is This Book For?

As part of Microsoft Press's "Developer Step By Step" series of training resources, *Microsoft ADO.NET 4 Step by Step* makes it easy to learn about ADO.NET and the advanced data tools used with it.

This book exists to help existing Visual Basic and C# developers understand the core concepts of ADO.NET and related technologies. It is especially useful for programmers looking to manage database-hosted information in their new or existing .NET applications. Although most readers will have no prior experience with ADO.NET, the book is also useful for those familiar with earlier versions of either ADO or ADO.NET, and who are interested in getting filled in on the newest features.

Assumptions

As a reader, the book expects that you have at least a minimal understanding of .NET development and object-oriented programming concepts. Although ADO.NET is available to most, if not all, .NET language platforms, this book includes examples in C# and Visual Basic only. If you have not yet picked up one of those languages, you might consider reading John Sharp's *Microsoft Visual C# 2010 Step by Step* (Microsoft Press 2010) or Michael Halvorson's *Microsoft Visual Basic 2010 Step by Step* (Microsoft Press 2010).

With a heavy focus on database concepts, this book assumes that you have a basic understanding of relational database systems such as Microsoft SQL Server, and have had brief exposure to one of the many flavors of the query tool known as SQL. To go beyond this book and expand your knowledge of SQL and Microsoft's SQL Server database platform, other Microsoft Press books such as Mike Hotek's *Microsoft® SQL Server® 2008 Step by Step* (Microsoft Press, 2008) or Itzik Ben-gan's *Microsoft® SQL Server® 2008 T-SQL Fundamentals* (Microsoft Press, 2008) offer both complete introductions and comprehensive information on T-SQL and SQL Server.

Organization of This Book

This book is divided into five sections, each of which focuses on a different aspect or technology within the ADO.NET family. Part I, "Getting to Know ADO.NET," provides a quick overview of ADO.NET and its fundamental role in .NET applications, then delves into the details of the main ADO.NET library, focusing on using the technology without yet being concerned with external database connections. Part II, "Connecting to External Data Sources," continues that core library focus, adding in the connectivity features. Part III, "Entity Framework," introduces the Entity Framework, Microsoft's model-based data service. Another service layer, LINQ, takes center stage in Part IV, "LINQ." Finally, Part V, "Presenting Data to the World," covers some miscellaneous topics that round out the full discussion of ADO.NET.

Finding Your Best Starting Point in This Book

The different sections of *Microsoft ADO.NET 4 Step by Step* cover a wide range of technologies associated with the data library. Depending on your needs and your existing understanding of Microsoft data tools, you may wish to focus on specific areas of the book. Use the following table to determine how best to proceed through the book.

If you are	Follow these steps
New to ADO.NET development, or an existing ADO developer	Focus on Parts I and II and on Chapter 21 in Part V, or read through the entire book in order.
Familiar with earlier releases of ADO.NET	Briefly skim Parts I and II if you need a refresher on the core concepts. Read up on the new technologies in Parts III and IV and be sure to read Chapter 22 in Part V.
Interested in the Entity Framework	Read Part III. Chapter 22 in Part V discusses data services built on top of Entity Framework models.
Interested in LINQ data providers	Read through the chapters in Part IV.

Most of the book's chapters include hands-on samples that let you try out the concepts just learned. No matter which sections you choose to focus on, be sure to download and install the sample applications on your system.

Conventions and Features in This Book

This book presents information using conventions designed to make the information readable and easy to follow.

- In most cases, the book includes separate exercises for Visual Basic programmers and Visual C# programmers. You can skip the exercises that do not apply to your selected language.
- Each exercise consists of a series of tasks, presented as numbered steps (1, 2, and so on) listing each action you must take to complete the exercise.
- Boxed elements with labels such as "**Note**" provide additional information or alternative methods for completing a step successfully.
- Text that you type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key while you press the Tab key.
- A vertical bar between two or more menu items (e.g. File | Close), means that you should select the first menu or menu item, then the next, and so on.

System Requirements

You will need the following hardware and software to complete the practice exercises in this book:

- One of Windows XP with Service Pack 3 (except Starter Edition), Windows Vista with Service Pack 2 (except Starter Edition), Windows 7, Windows Server 2003 with Service Pack 2, Windows Server 2003 R2, Windows Server 2008 with Service Pack 2, or Windows Server 2008 R2
- Visual Studio 2010, any edition (multiple downloads may be required if using Express Edition products)
- SQL Server 2008 Express Edition or higher (2008 or R2 release), with SQL Server Management Studio 2008 Express or higher (included with Visual Studio, Express Editions require separate download)
- Computer that has a 1.6GHz or faster processor (2GHz recommended)
- 1 GB (32 Bit) or 2 GB (64 Bit) RAM (add 512 MB if running in a virtual machine or SQL Server Express Editions; more for advanced SQL Server editions)
- 3.5GB of available hard disk space
- 5400 RPM hard disk drive

- DirectX 9 capable video card running at 1024 x 768 or higher-resolution display
- DVD-ROM drive (if installing Visual Studio from DVD)
- Internet connection to download software or chapter examples

Depending on your Windows configuration, you might need Local Administrator rights to install or configure Visual Studio 2010 and SQL Server 2008 products.

Code Samples

Most of the chapters in this book include exercises that let you interactively try out new material learned in the main text. All sample projects, in both their pre-exercise and post-exercise formats, are available for download from the book's catalog page online:

http://aka.ms/638884/files

Note In addition to the code samples, your system should have Visual Studio 2010 and SQL Server 2008 installed. The instructions below use SQL Server Management Studio 2008 to set up the sample database used with the practice examples. If available, install the latest service packs for each product.

Installing the Code Samples

Follow these steps to install the code samples on your computer so that you can use them with the exercises in this book.

1. Open the *file that you downloaded from the book's web site.*

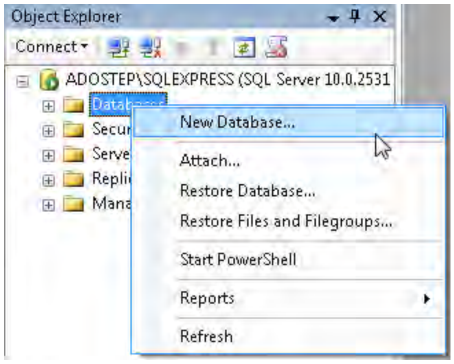
2. Copy the entire contents of the opened .zip file to a convenient location on your hard disk.

Installing the Sample Database

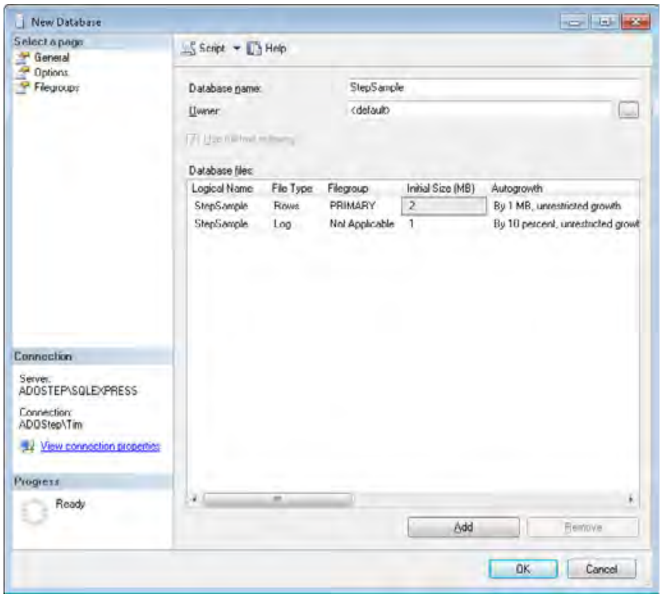
Follow these steps to install the sample database used by many of the book's practice examples.

Note You must first download and install the Code Samples using the instructions listed above. Also, you must have both SQL Server 2008 and SQL Server Management Studio 2008 installed, any edition.

1. Start SQL Server Management Studio 2008 and open a new Object Explorer connection to the target database instance using the File | Connect Object Explorer menu command.
2. In the Object Explorer panel, right-click on the Databases branch of the connection tree, and select New Database from the shortcut menu.



3. When the New Database dialog box appears, enter **StepSample** in the Database Name field. Click OK to create the database.



Database name: StepSample

Owner: <default>

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth
StepSample	Rows	PRIMARY	2	By 1 MB, unrestricted growth
StepSample	Log	Not Applicable	1	By 10 percent, unrestricted growt

Connection

Server:
ADOSTEP\SQLEXPRESS

Connection:
ADOStep\Tim

View connection properties

Progress

Ready

4. Select File | Open | File from the main SQL Server Management Studio menu, and locate the *DB Script.sql* file installed with the book's sample projects. This file appears in the *Sample Database* folder within the main installation folder.
5. Click the Execute button on the SQL Editor toolbar to run the script. This will create the necessary tables and objects needed by the practice examples.
6. Close SQL Server Management Studio 2008.

Using the Code Samples

The main installation folder extracted from the *ADO.NET 4 SBS Examples.zip* file contains three subfolders.

- *Sample Database* This folder contains the SQL script used to build the sample database. The instructions for creating this database appear earlier in this Introduction.
- *Exercises* The main example projects referenced in each chapter appear in this folder. Many of these projects are incomplete, and will not run without following the steps indicated in the associated chapter. Separate folders indicate each chapter's sample code, and there are distinct folders for the C# and Visual Basic versions of each example.
- *Completed Exercises* This folder contains all content from the Exercises folder, but with chapter-specific instructions applied.

To complete an exercise, access the appropriate chapter-and-language folder in the *Exercises* folder, and open the project file. If your system is configured to display file extensions, Visual Basic project files use a .vbproj extension, while C# project files use .csproj as the file extension.

Uninstalling the Code Samples

To remove the code samples from your system, simply delete the installation folder that you extracted from the .zip file.

Software Release

This book was written for use with Visual Studio 2010, including the Express Editions products. Much of the content will apply to other versions of Visual Studio, but the code samples may be not be fully compatible with earlier or later versions of Visual Studio.

The practice examples in the book use SQL Server 2008, including the Express Edition products. Many of the examples may work with SQL Server 2005 or earlier versions, but neither the installation script nor the sample projects have been tested with those earlier releases.

Errata and Book Support

We've made every effort to ensure the accuracy of this book and its companion content. If you do find an error, please report it on our Microsoft Press site:

1. Go to *www.microsoftpressstore.com*.
2. In the Search box, enter the book's ISBN or title.
3. Select your book from the search results.
4. On your book's catalog page, find the Errata & Updates tab
5. Click View/Submit Errata.

You'll find additional information and services for your book on its catalog page. If you need additional support, please e-mail Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

Part I
Getting to Know ADO.NET

Chapter 1: Introducing ADO.NET 4

Chapter 2: Building Tables of Data

Chapter 3: Storing Data in Memory

Chapter 4: Accessing the Right Data Values

Chapter 5: Bringing Related Data Together

Chapter 6: Turning Data into Information

Chapter 7: Saving and Restoring Data

Chapter 1
Introducing ADO.NET 4

After completing this chapter, you will be able to:

- Identify what ADO.NET is
- Explain ADO.NET's role in an application
- Identify the major components that make up ADO.NET
- Create an ADO.NET link between a database and a .NET application

This chapter introduces you to ADO.NET and its purpose in the world of Microsoft .NET application development. ADO.NET has been included with the .NET Framework since its initial release in 2002, playing a central role in the development of both desktop and Internet-targeted applications for programmers using C#, Visual Basic, and other Framework languages.

What Is ADO.NET?

ADO.NET is a family of technologies that allows .NET developers to interact with data in standard, structured, and primarily disconnected ways. If that sounds confusing, don't worry. This book exists to remove the confusion and anxiety that many developers experience when they first learn of ADO.NET's multiple object layers, its dozens of general and platform-specific classes, and its myriad options for interacting with actual data.

Applications written using the .NET Framework depend on *.NET class libraries*, which exist in special DLL files that encapsulate common programming functionality in an easy-to-access format. Most of the libraries supplied with the .NET Framework appear within the *System* namespace. *System.IO*, for instance, includes classes that let you interact with standard disk files and related data streams. The *System.Security* library provides access to, among other things, data encryption features. ADO.NET, expressed through the *System.Data* namespace, implements a small set of libraries that makes consuming and manipulating large amounts of data simple and straightforward.

ADO.NET manages both internal data—data created in memory and used solely within an application—and external data—data housed in a storage area apart from the application, such as in a relational database or text file. Regardless of the source, ADO.NET generalizes the relevant data and presents it to your code in spreadsheet–style rows and columns.

Note Although ADO.NET manipulates data in tabular form, you can also use ADO.NET to access nontabular data. For instance, an ADO.NET provider (discussed later in the chapter, on page 7) could supply access to hierarchical data such as that found in the Windows Registry, as long as that provider expressed the data in a tabular structure for ADO.NET's use. Accessing such nontabular data is beyond the scope of this book.

If you are already familiar with relational databases such as Microsoft SQL Server, you will encounter many familiar terms in ADO.NET. Tables, rows, columns, relations, views; these ADO.NET concepts are based loosely on their relational database counterparts. Despite these similarities, ADO.NET is not a relational database because it doesn't include key "relational algebra" features typically found in robust database systems. It also lacks many of the common support features of such databases, including indexes, stored procedures, and triggers. Still, if you limit yourself to basic create, read, update, and delete (CRUD) operations, ADO.NET can act like a miniature yet powerful in-memory database.

As an acronym, "ADO.NET" stands for—nothing. Just like the words "scuba," "laser," and "NT" in Windows NT, the capital letters in ADO.NET used to mean something, but now it is just a standalone term. Before Microsoft released the .NET Framework, one of the primary data access tools Windows developers used in their programs was known as *ADO*, which did stand for something: ActiveX Data Objects. After .NET arrived on the scene, ADO.NET became the natural successor to ADO. Although conceptual parallels exist between ADO.NET and ADO, the technologies are distinct and incompatible.

Note ADO is based on Microsoft's older COM technology. The .NET Framework provides support for COM components, and therefore enables .NET programs to use ADO. This is especially useful for development teams transitioning legacy applications to .NET. Although ADO and ADO.NET components can appear in the same application, they can interact only indirectly because their object libraries are unrelated.

When communicating with external data stores, ADO.NET presents a disconnected data experience. In earlier data platforms, including ADO, software developers would typically establish a persistent connection with a database and use various forms of record locking to manage safe and accurate data updates. But then along came the Internet and its browser-centric view of information. Maintaining a long-standing data connection through bursts of HTTP text content was no longer a realistic expectation. ADO.NET's preference toward on-again, off-again database connections reflects this reality. Although this paradigm change brought with it difficulties for traditional client-server application developers, it also helped usher in the era of massive scalability and n-tier development that is now common to both desktop and Web-based systems.

Why ADO.NET?

In the early days of computer programming, the need for a data library like ADO.NET didn't exist. Programmers had only a single method of accessing data: direct interaction with the values in memory. Permanently stored data existed on tape reels in fire-resistant, climate-controlled, raised-floor rooms. Data queries could take hours, especially if someone with more clout had a higher-priority processing need.

Over time, computers increased in complexity, and (as if to fill some eternal maxim) data processing needs also expanded to consume all available computing resources. Businesses sought easier ways to manage entire records of numeric, text, and date-time values on their mainframe systems. Flat-file and relational database systems sprang up to establish proprietary management of millions of data values. As personal computers arrived and matured, developers soon had several database systems at their disposal.

This was great news for data consumers. Businesses and individuals now had powerful tools to transform data bits into usable information, to endow seemingly unrelated values with meaning and purpose. But it was bad news for developers. As technology marched on, companies purchased one proprietary system after another. Programming against such systems meant a reinvention of the proverbial wheel each time a middle manager asked for yet another one-time report. Even the standard SQL language brought little relief because each database vendor provided its own spin on the meaning of "standard."

What programmers needed was a way to generalize different data systems in a standard, consistent, and powerful way. In the world of .NET application development, Microsoft ADO.NET meets that need. Instead of worrying about the minutiae associated with the different database systems, programmers using ADO.NET focus on the data content itself.

Major Components of ADO.NET

The *System.Data* namespace includes many distinct ADO.NET classes that work together to provide access to tabular data. The library includes two major groups of classes: those that manage the actual data within the software and those that communicate with external data systems. Figure 1-1 shows the major parts that make up an ADO.NET instance.

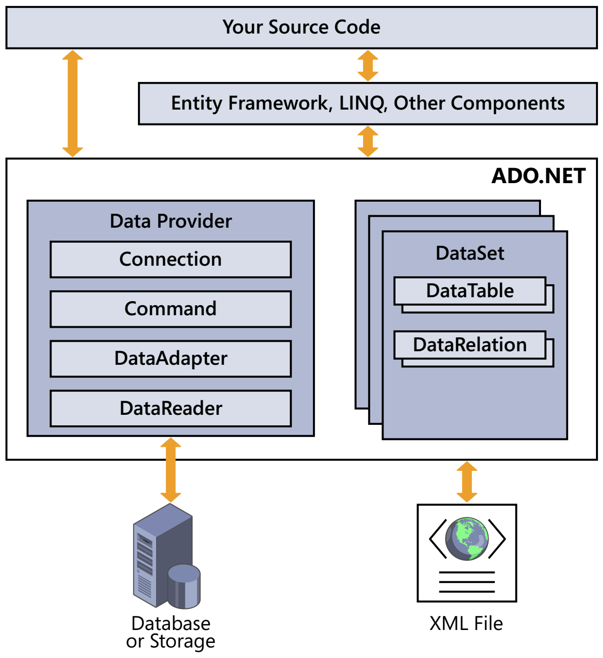


FIGURE 1-1 Key ADO.NET elements.

At the data-shaped heart of the library is the *DataTable*. Similar in purpose to tables in a database, the *DataTable* manages all the actual data values that you and your source code ultimately care about. Each *DataTable* contains zero or more rows of data, with the individual data values of each row identified by the table's column definitions.

- Each table defines *DataColumn* items, each representing the individual data values that appear in the table's records. *DataColumn* definitions include a data type declaration based on the kind of data destined for each column. For instance, a *CustomerLastName* column might be defined to use data of type *System.String*, whereas an *OrderSalesTax* column could be crafted for use with *System.Decimal* content.
- One *DataRow* entry exists for each record of data stored within a table, providing access to the distinct columnar data values. ADO.NET includes methods that let you add to, delete from, modify, and query each *DataTable* object's rows. For tables connected to an external data storage area, any changes made can be propagated back to the source.
- You can optionally establish links between the tables of data using *DataRelation* entries.

- Programmatic limitations can be placed on tables and their data values using *Constraint* instances.
- *DataView* instances provide a limited or modified view of the rows in a *DataTable*.
- Tables can be grouped together into a *DataSet*. Some tools that interact with ADO.NET data require that any tables be bound within a *DataSet*, but if you plan to do some limited work with only a single table, it's fine to work with just the *DataTable* instance.

DataTable instances and their associated objects are sufficient for working with internal data. To connect with external data from a database, ADO.NET features multiple *data providers*, including a custom provider for Microsoft SQL Server. Database platforms without a specific provider use the more generic ODBC and OLE DB providers, both included with ADO.NET. Several third-party providers can be purchased or obtained free of charge, which target specific platforms, including Oracle.

- All communication with the external data source occurs through a *Connection* object. ADO.NET supports connection pooling for increased efficiency between queries.
- SQL queries and data management statements get wrapped in a *Command* object before being sent to the data source. Commands can include optional *Parameter* instances that let you call stored procedures or create fill-in-the-blank queries.
- The *DataAdapter* object stores standard query definitions for interacting with a database, removing the tedium of constantly needing to build SQL statements for each record you want to read or write, and helping to automate some ADO.NET-related tasks.
- The *DataReader* object provides fast, read-only access to the results of a query for those times when you just need to get your data quickly.

ADO.NET also includes features that let you save an entire *DataSet* as an XML file and load it back in later. And that's just the start. You'll learn how to use all these elements—and more—throughout the upcoming chapters.

Extensions to ADO.NET

Generalizing access to data is a key benefit of using ADO.NET. But an even greater advantage for .NET developers is that all values managed through ADO.NET appear as objects, first-class members of the .NET data world. Each data field in a table is a strongly typed data member, fully compliant with .NET's Common Type System. Individual fields can be used just like any other local variable. Data rows and other sets of objects are standard .NET collections and can be processed using standard iteration methods.

Because ADO.NET values exist as true .NET objects and collections, Microsoft has enhanced the core ADO.NET feature set with new tools. Two of these technologies, the Entity Framework and LINQ, are not formally part of ADO.NET. But their capability to interact with and enhance the ADO.NET experience makes them essential topics for study.

The *Entity Framework*, the focus of Part III of this book, emphasizes the conceptual view of your data. Although the data classes in ADO.NET are programmer-friendly, you still need to keep track of primary keys and relationships between tables and fields. The Entity Framework attempts to hide that messiness, and restores the promise of what object-oriented programming was supposed to be all about. In the Entity Framework, a customer object includes its orders; each order includes line item details. Instead of working with the raw table data, you interact with logically designed entities that mimic their real-world counterparts, and let the Framework worry about translating it all into SQL statements.

LINQ, introduced in Part IV, brings the joy of English-like queries to your favorite programming language. Microsoft enhanced both Visual Basic and C# with new LINQ-specific language features. Now, instead of building string-based SQL statements to query data, the syntax of each programming language becomes the query language. LINQ is a generic data tool, enabling you to easily mix ADO.NET data and other content sources together into a single set of results.

Connecting to External Data

Chapter 8, "Establishing External Connections," introduces the code elements that support communications between ADO.NET and external sources of data. Although using only code to establish these connections is quite common, Visual Studio also includes the *Data Source Connection Wizard*, a mouse-friendly tool that guides you through the creation of a ready-to-use *DataSet*. Here's an example of using the Connection Wizard.

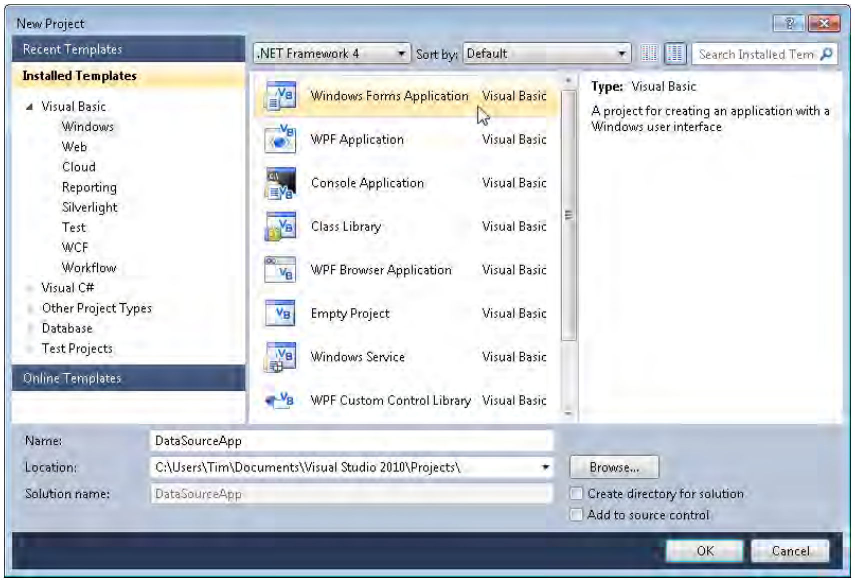
Creating a Data Source Using the Connection Wizard

1. Start Visual Studio 2010. Select File | New | Project from the main menu.

   ADO.NET is supported in most common project types. To keep things simple for now, create a Windows Forms application using either C# or Visual Basic as the language. The following figures show the process using a Visual Basic Windows Forms application, although the steps are identical in C#.

2. In the New Project dialog box, provide a name for the project.
3. Click OK.

   Visual Studio will create a project.

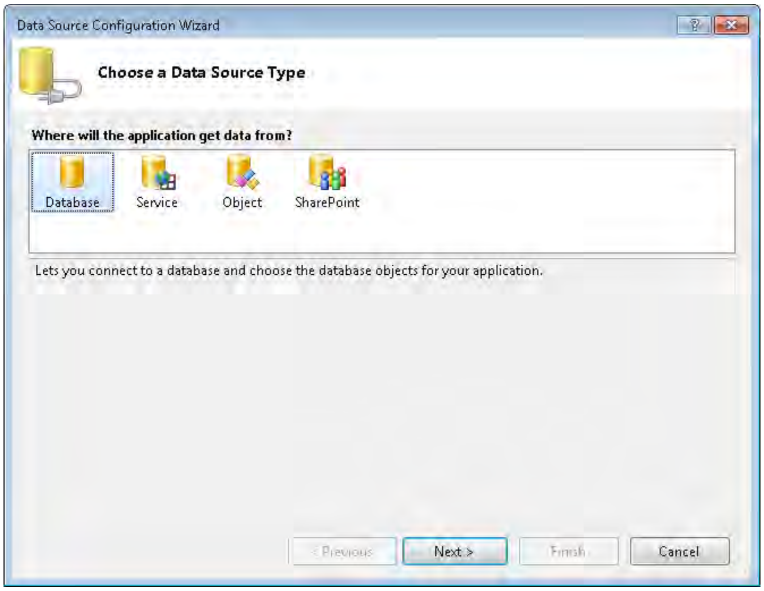

New Project
Recent Templates
Installed Templates
Visual Basic
Windows
Web
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Visual C#
Other Project Types
Database
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Tem
Windows Forms Application Visual Basic
WPF Application Visual Basic
Console Application Visual Basic
Class Library Visual Basic
WPF Browser Application Visual Basic
Empty Project Visual Basic
Windows Service Visual Basic
WPF Custom Control Library Visual Basic
Browse...
Create directory for solution
Add to source control
OK
Cancel

Type: Visual Basic
A project for creating an application with a Windows user interface

Name: DataSourceApp
Location: C:\Users\Tim\Documents\Visual Studio 2010\Projects\
Solution name: DataSourceApp

4. Select Data | Add New Data Source from the menu.

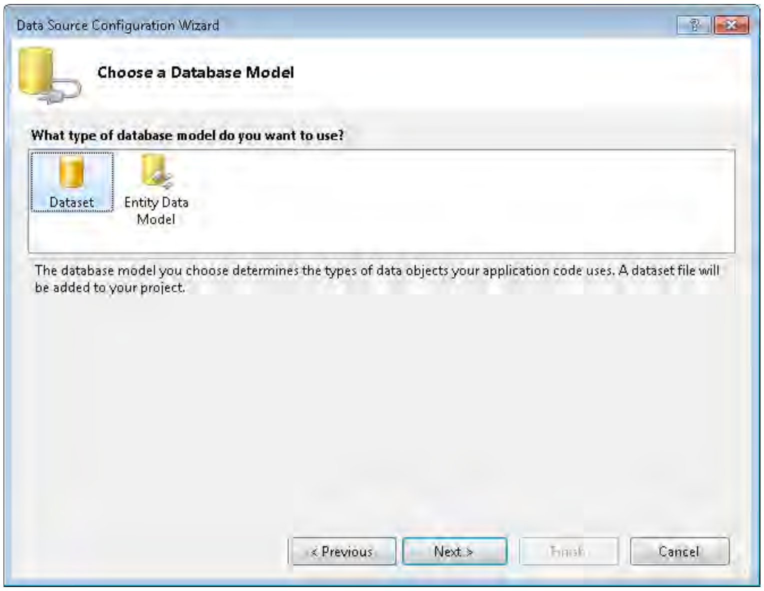
   Visual Studio displays the Data Source Configuration Wizard.


Data Source Configuration Wizard
Choose a Data Source Type
Where will the application get data from?
Database
Service
Object
SharePoint
< Previous
Next >
Finish
Cancel

Lets you connect to a database and choose the database objects for your application.

The Database choice should already be selected in the Choose A Data Source Type panel.

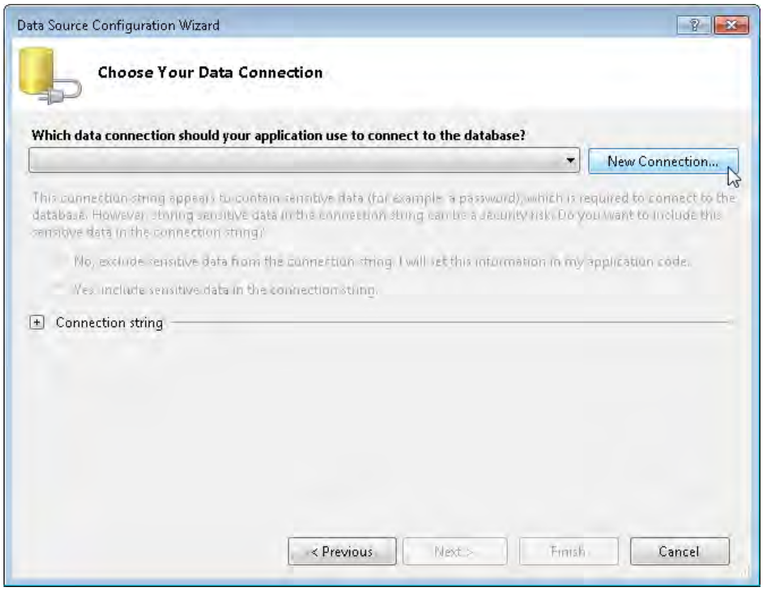
5. Click Next.
6. In the Choose a Database Model panel, choose Dataset.


Data Source Configuration Wizard
Choose a Database Model
What type of database model do you want to use?
Dataset
Entity Data Model
< Previous
Next >
Finish
Cancel

The database model you choose determines the types of data objects your application code uses. A dataset file will be added to your project.

7. Click Next.

   The Wizard displays the Choose Your Data Connection panel. If you previously configured data sources, they will appear in the Which Data Connection Should Your Application Use To Connect To The Database? list.
8. Because you are setting up a connection to the test database for the first time, click the New Connection button.



Which data connection should your application use to connect to the database?

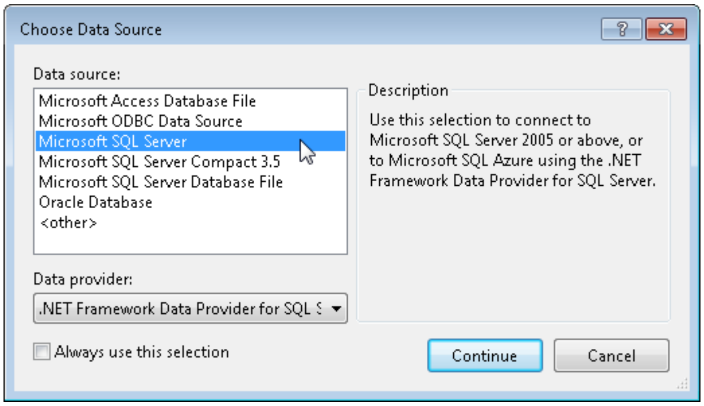
This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

No, exclude sensitive data from the connection string. I will set this information in my application code.

Yes, include sensitive data in the connection string.

9. When the Choose Data Source dialog box appears, select Microsoft SQL Server from the Data Source list.

   The Data Provider field will automatically choose the SQL Server data provider. For maximum flexibility, clear the Always Use This Selection field.


Choose Data Source
Data source:
Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Compact 3.5
Microsoft SQL Server Database File
Oracle Database
<other>
Description
Data provider:
.NET Framework Data Provider for SQL S
Continue
Cancel

Use this selection to connect to Microsoft SQL Server 2005 or above, or to Microsoft SQL Azure using the .NET Framework Data Provider for SQL Server.

Always use this selection

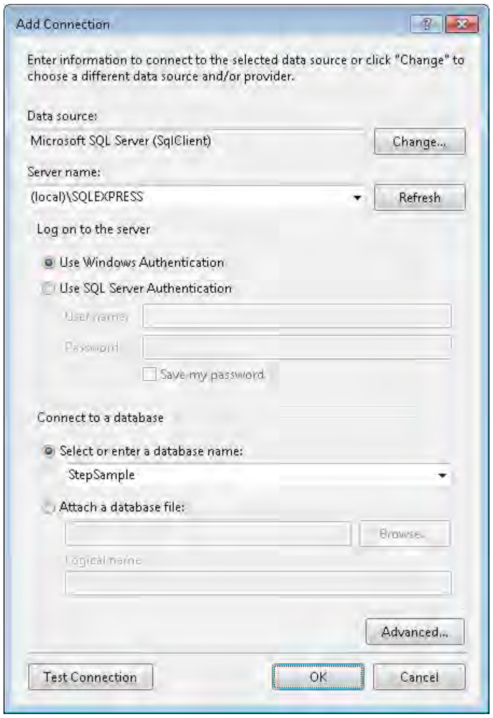
Note Choosing Microsoft SQL Server will access a database that has previously been attached to a SQL Server database instance. To create a data source that directly references a database file not necessarily attached to the engine instance, select Microsoft SQL Server Database File from the Data Source list instead. The wizard will then prompt you for the disk location of the file.

10. Click Continue to accept the data source.

11. In the Add Connection dialog box, select the server from the Server Name field.

    For SQL Server 2008 Express Edition instances, this is typically the name of the local computer with **\SQLEXPRESS** appended to the name. If you are using the full SQL Server product, leave off the **\SQLEXPRESS** suffix. For SQL Server instances hosted on the same system as your Visual Studio installation, you can use **(local)** as the server name.

    For SQL Server instances configured with database-managed authentication, select Use SQL Server Authentication and supply the appropriate user name and password. For databases managed with Windows authentication (the likely choice for the test database), select Use Windows Authentication instead.



Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: (local)\SQLEXPRESS

Log on to the server

Use Windows Authentication (selected)

Use SQL Server Authentication

Connect to a database

Select or enter a database name: (selected) StepSample

Attach a database file:

    The Select Or Enter a Database Name field should now include the available databases within the test database file. (If not, confirm that you have supplied the right server name and authentication values and that SQL Server is running on your system.)

12. Select StepSample (or the name of your primary test database) from the list. Then click OK to complete the connection configuration.

    Control returns to the wizard with the new data connection selected in the list on the Choose Your Data Connection panel.

Note ADO.NET uses *connection strings*, short, semicolon-delimited definition strings, to identify the data source. As you develop new applications, you will probably forgo the Data Source Configuration Wizard as a means of building connection strings. If you are curious about what appears in a connection string, expand the Connection String field in the Choose Your Data Connection panel.

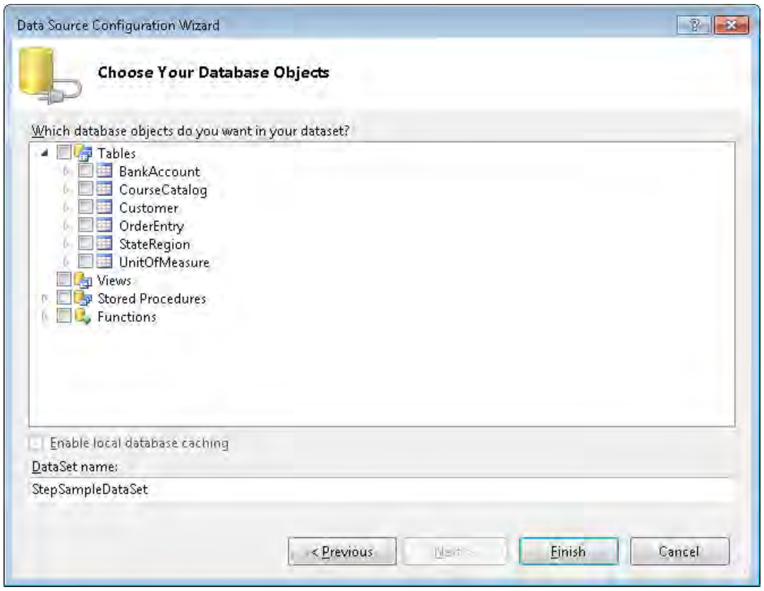
13. Click the Next button to continue.

The next wizard panel asks if the connection string should be stored in the application's configuration file. The field should already be selected, which is good, although you might want to give it a more programmer-friendly name.

Note .NET applications use two types of configuration files (although it varies by project type): *application configuration files* and *user configuration files*. Although your application has access to the settings in both files, if you plan to include a feature in your program that modifies these saved settings, make sure that you place such settings in the user configuration file. Application configuration files can't be modified from within the associated application.

14. Click the Next button once more to continue.

SQL Server will perform a quick analysis of your database, compiling a list of all available data-exposing items, including tables, views, stored procedures, and functions. The Choose Your Database Objects panel displays all items found during this discovery process.


Data Source Configuration Wizard

Choose Your Database Objects

Which database objects do you want in your dataset?

- Tables
  - BankAccount
  - CourseCatalog
  - Customer
  - OrderEntry
  - StateRegion
  - UnitOfMeasure
- Views
- Stored Procedures
- Functions

Enable local database caching

DataSet name:

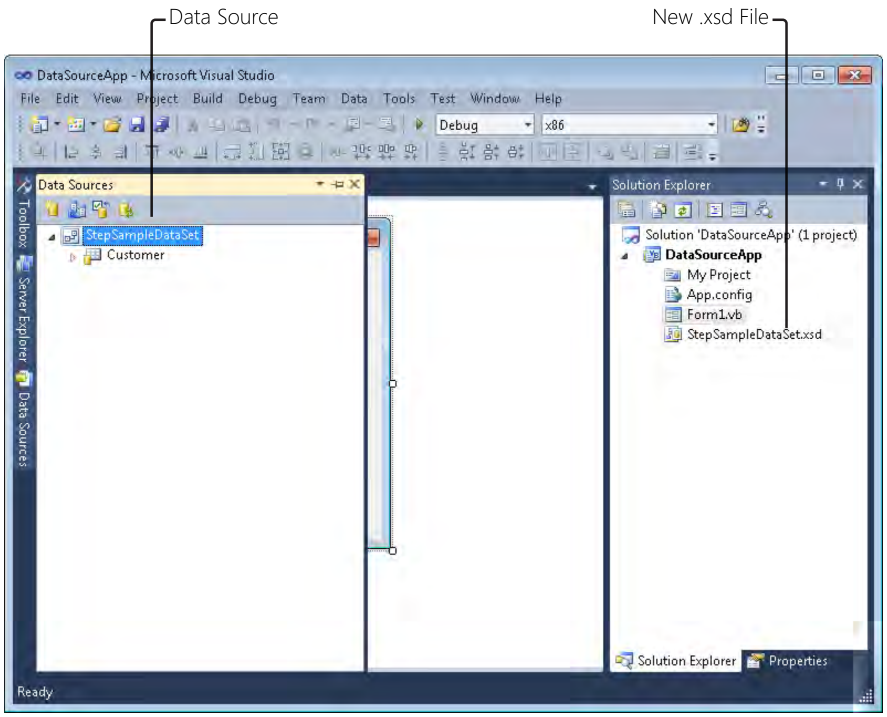
StepSampleDataSet

15. For this test, include the Customer table in the DataSet by expanding the Tables section and marking the Customer table with a check mark.

    You can optionally modify the DataSet Name field to something that will be easier to repeatedly type in your source code. Click Finish to exit the wizard and create the data source. The data source is now available for use in your application.

16. Select Data | Show Data Sources from the Visual Studio menu to see the data source.

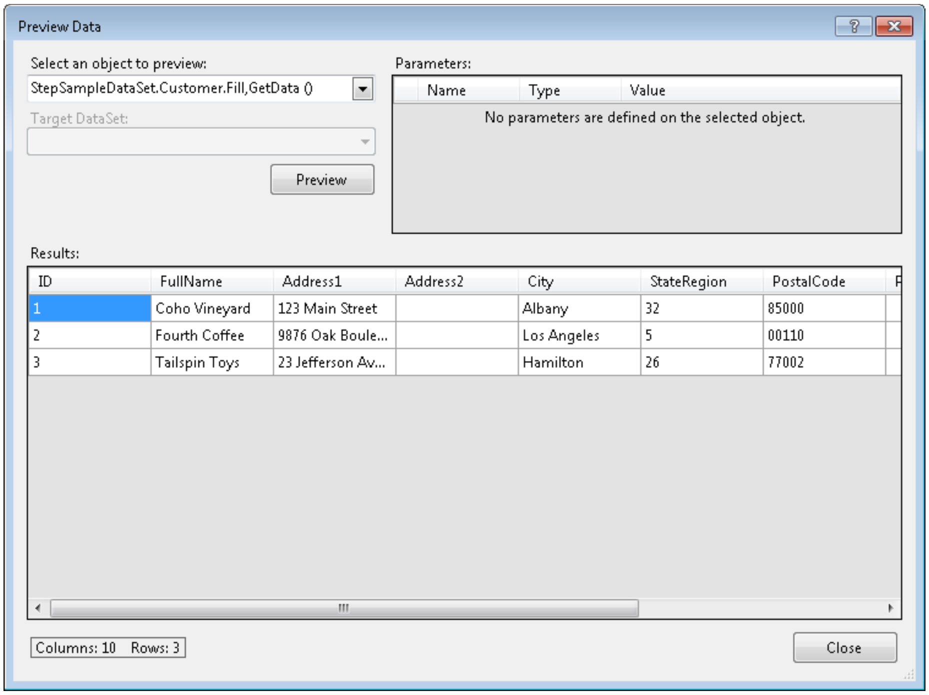
    The wizard also added a new .xsd file to your project; it appears in the Solution Explorer with your other project files. This XML file contains the actual definition of the data source. Removing this file from the project removes the Wizard-created data source.



Visual Studio also lets you preview the data records within the data source.

17. Select Data | Preview Data from the Visual Studio menu to open the Preview Data dialog box.

    The menu choice might be hidden depending on what is currently active in the Visual Studio IDE. If that menu choice does not appear, click the form in the design window and then try to select the menu item again.


Preview Data
Select an object to preview:
StepSampleDataSet.Customer.Fill,GetData ()
Target DataSet:
Preview
Parameters:
Results:
Close

Name	Type	Value
No parameters are defined on the selected object.		

ID	FullName	Address1	Address2	City	StateRegion	PostalCode
1	Coho Vineyard	123 Main Street		Albany	32	85000
2	Fourth Coffee	9876 Oak Boule...		Los Angeles	5	00110
3	Tailspin Toys	23 Jefferson Av...		Hamilton	26	77002

Columns: 10 Rows: 3

Summary

This chapter provided an overview of Microsoft's ADO.NET technology and its major data management components. At its heart, computer programming is all about data manipulation, whether the data values represent customer records, characters and objects in a 3D interactive video game, or the bits in a compressed audio file. With this inherent focus on data, it makes sense that Microsoft would provide a great tool for interacting with tabular data, one of the most useful ways of organizing data, especially in a business setting.

As you will see in upcoming chapters, the concepts included in this opening chapter have direct ties to specific ADO.NET classes and class members. As a .NET developer, you already have a core understanding of how ADO.NET can be used in an application because everything in the library is expressed as standard .NET objects. The only things you still need to learn are some of the details that are specific to ADO.NET—the very subjects covered in the rest of this book.

Chapter 1 Quick Reference

To	Do This
Create a new data source	Create or open a project in Visual Studio. Select Data | Add New Data Source. Follow the steps in the Connection Wizard.
Preview data in an existing data source	Select Data | Preview Data. Select the target data source from the Select An Object To Preview list. Click the Preview button.
Remove a data source from a project	Select the .xsd file in the Solution Explorer. Press the Delete key or right-click on the file and select Delete from the shortcut menu.

Chapter 3

Storing Data in Memory

After completing this chapter, you will be able to:

- Explain how a *DataTable* stores data
- Add new data rows to a table
- Examine, update, and remove existing values in a table row
- Explain how ADO.NET differentiates between pending and final data values
- Integrate data verification code into your *DataTable* object

Adding columns to a *DataTable* is an essential step in managing data in ADO.NET, but the columns themselves contain no data. To store actual data values in an ADO.NET table, you must use the *DataRow* class. After you place one or more data rows in a table, the real work of managing application-specific information begins. This chapter introduces the *DataRow* class and its role in data storage within each data table.

Note The exercises in this chapter all use the same sample project, a simple editor of *DataRow* records within a single *DataTable*. Although you will be able to run the application after each exercise, the expected results for the full application might not appear until you complete all exercises in the chapter.

Adding Data

Adding new data rows to a table is a three-step process:

1. Create a new row object.
2. Store the actual data values in the row object.
3. Add the row object to the table.

Creating New Rows

The *DataColumn* objects you add to a *DataTable* let you define an unlimited number of column combinations. One table might manage information on individuals, with textual name fields and dates for birthdays and driver-license expirations. Another table might exist to track the score in a baseball game, and contain no names or dates at all. The type of information you store in a table depends on the columns included in that table, along with the name, data type, and field constraints for each column.

The *DataRow* class lets you store a single row of data in a table. However, a row of data that tracks customers or medical patients is not the same as a row that tracks baseball scores. The columns differ in number, data types, and even their names and positions. Therefore, each ADO.NET *DataRow* must be configured to work with a specific *DataTable* and its collection of *DataColumn* instances.

The *DataTable* class includes the *NewRow* method to generate table-specific data rows. Whenever you want to add a new row of data to a table, the first step always involves generating a new *DataRow* with the *NewRow* method.

C#

```
DataRow oneRow = someTable.NewRow();
```

Visual Basic

```
Dim oneRow As DataRow = someTable.NewRow()
```

The generated row includes information about each data column defined for the table. Typically, the data associated with each column in the new row is initially NULL, the database state for an unassigned field. However, if a *DataColumn* definition includes a *DefaultValue* setting, that initial value will appear immediately in the generated row for the named column. Also, any column that has its *AutoIncrement* and related fields set (typically a primary key field) will include generated sequential values for that column.

Defining Row Values

The *DataRow* class includes an *Item* property that provides access to each defined column, by name, zero-based index number, or reference to the physical *DataColumn* instance. When writing code with a specific table format in mind, programmers generally use the column-name method because it makes clear which field is being referenced in a code statement.

C#

```
oneRow.Item["ID"] = 123;         // by column name
oneRow.Item[0] = 123;            // by column position
DataColumn whichColumn = someTable.Columns[0];
oneRow.Item[whichColumn] = 123;  // by column instance
```

Visual Basic

```
oneRow.Item("ID") = 123         ' by column name
oneRow.Item(0) = 123            ' by column position
Dim whichColumn As DataColumn = someTable.Columns(0)
oneRow.Item(whichColumn) = 123  ' by column instance
```

Because *Item* is the default member for the *DataRow* class, you can omit the name when referencing row values, as shown here:

C#

```
oneRow["ID"] = 123;
```

Visual Basic

```
oneRow("ID") = 123
```

Visual Basic includes a special "exclamation point" syntax that condenses the statement even more, but you can use it only with column names, not with column indexes.

Visual Basic

```
oneRow!ID = 123
```

Note Members of the *Item* class are defined as the generic *Object* type; they are not strongly typed to the data type defined for the columns. This means that you can store data of an incorrect type in any field during this assignment phase. Errors will not be reported until you attempt to add the *DataRow* object to the table's *Rows* collection, as described in the "Storing Rows in a Table" section of this chapter on page 40.

As you assign values to a row, they become available immediately for use in other expressions.

C#

```
orderData["Subtotal"] = orderRecord.PreTaxTotal;
orderData["SalesTax"] = orderRecord.PreTaxTotal * orderRecord.TaxRate;
orderData["Total"] = orderData["Subtotal"] + orderData["SalesTax"];
```

Visual Basic

```
orderData!Subtotal = orderRecord.PreTaxTotal
orderData!SalesTax = orderRecord.PreTaxTotal * orderRecord.TaxRate
orderData!Total = orderData!Subtotal + orderData!SalesTax
```

Fields with no default or auto-increment value are automatically set to NULL. If for any reason you need to set a field to NULL from a non-NULL state, assign it with the value of .NET's *DBNull* class.

C#

```
oneRow["Comments"] = System.DBNull.Value;
```

Visual Basic

```
oneRow!Comments = System.DBNull.Value
```

As mentioned in Chapter 2, "Building Tables of Data," you can test field values in C# using the *DBNull.Value.Equals* method or in Visual Basic with the *IsDBNull* function. The *DataRow* class includes its own *IsNull* method; it is functionally equivalent to the methods from Chapter 2. Instead of passing the *IsNull* method a field value to test, you pass it the column's name, the column's position, or an instance of the column.

C#

```
if (oneRow.IsNull("Comments"))...
```

Visual Basic

```
If (oneRow.IsNull("Comments") = True)...
```

Note *System.DBNull* is not the same as *null* in C#, or *Nothing* in Visual Basic. Those keywords indicate the absence of an object's value. *System.DBNull.Value* is an object that presents a value.

Storing Rows in a Table

After you have assigned all required data values to the columns in a new row, add that row to the *DataTable* using the table's *Rows.Add* method.

C#

```
someTable.Rows.Add(oneRow);
```

Visual Basic

```
someTable.Rows.Add(oneRow)
```

An overload of the *Add* method lets you skip the formal row-object creation process; instead, you supply the final field values directly as arguments. All provided values must appear in the same order and position as the table's *DataColumn* instances.

C#

```
// ----- Assumes column 0 is numeric, 1 is string.
someTable.Rows.Add(new Object[] {123, "Fred"});
```

Visual Basic

```
' ----- Assumes column 0 is numeric, 1 is string.
someTable.Rows.Add(123, "Fred");
```

Whichever method you employ, the *Add* process tests all data values to be added to the table for data type compliance before adding the row. If the new row contains any values that can't be stored in the target column-specific data type, the *Add* method throws an exception.

Adding Rows to a *DataTable*: C#

1. Open the "Chapter 3 CSharp" project from the installed samples folder. The project includes two *Windows.Forms* classes: *AccountManager* and *AccountDetail*.
2. Open the source code view for the *AccountManager* form. Locate the *AccountManager_Load* event handler. This routine creates a custom *DataTable* instance with five columns: *ID* (a read-only, auto-generated long integer), *FullName* (a required 30-character unique string), *Active* (a Boolean), *AnnualFee* (an optional decimal), and *StartDate* (an optional date).
3. Add the following statements just after the "Build some sample data rows" comment. These rows add new *DataRow* objects to the table using the *Rows.Add* alternative syntax:

```
CustomerAccounts.Rows.Add(new Object[] {1L, "Blue Yonder Airlines", true,
    500m, DateTime.Parse("1/1/2007")});
CustomerAccounts.Rows.Add(new Object[] {2L, "Fourth Coffee", true, 350m,
    DateTime.Parse("7/25/2009")});
CustomerAccounts.Rows.Add(new Object[] {3L, "Wingtip Toys", false});
```

Adding Rows to a *DataTable*: Visual Basic

1. Open the "Chapter 3 VB" project from the installed samples folder. The project includes two *Windows.Forms* classes: *AccountManager* and *AccountDetail*.
2. Open the source code view for the *AccountManager* form. Locate the *AccountManager_Load* event handler. This routine creates a custom *DataTable* instance with five columns: *ID* (a read-only, auto-generated long integer), *FullName* (a required 30-character unique string), *Active* (a Boolean), *AnnualFee* (an optional decimal), and *StartDate* (an optional date).

3. Add the following statements just after the "Build some sample data rows" comment. These rows add new *DataRow* objects to the table using the *Rows.Add* alternative syntax:

```
CustomerAccounts.Rows.Add({1&, "Blue Yonder Airlines", True, 500@, #1/1/2007#})
CustomerAccounts.Rows.Add({2&, "Fourth Coffee", True, 350@, #7/25/2009#})
CustomerAccounts.Rows.Add({3&, "Wingtip Toys", False})
```

Examining and Changing Data

After adding a data row to a table, you can process it as a table member. For instance, you can iterate through the table's *Rows* collection, examining the stored column values as you pass through each record. The following code adds up the sales tax for all records in the *allSales* table:

C#

```
decimal totalTax = 0m;
foreach (DataRow scanRow in someTable.Rows)
    if (!DBNull.Value.Equals(scanRow["SalesTax"]))
        totalTax += (decimal)scanRow["SalesTax"];
```

Visual Basic

```
Dim totalTax As Decimal = 0@
For Each scanRow As DataRow In someTable.Rows
    If (IsDBNull(scanRow!SalesTax) = False) Then _
        totalTax += CDec(scanRow!SalesTax)
Next scanRow
```

Because each row's collection of items is not strongly typed, you might need to cast or convert each field to the target data type before using it.

Note ADO.NET does include extension methods that provide strongly typed access to each row's members. These methods were added to the system to support LINQ and its method of querying data within the context of the Visual Basic or C# language. Part IV of this book introduces LINQ and its use with ADO.NET data.

Because of this lack of strong typing, be careful when assigning new values to any row already included in a table. For example, code that assigns a string value to an integer column will compile without error, but will generate a runtime error.

Modifying Existing Rows in a *DataTable*: C#

Note This exercise uses the "Chapter 3 CSharp" sample project and continues the preceding exercise in this chapter.

1. Open the source code view for the *AccountDetail* form. Locate the *AccountDetail_Load* routine.
2. Add the following code, which fills in the form's display fields with content from an existing *DataRow* instance:

```
if (AccountEntry != null)
{
    AccountID.Text = string.Format("{0:0}", AccountEntry["ID"]);
    ActiveAccount.Checked = (bool)AccountEntry["Active"];
    if (DBNull.Value.Equals(AccountEntry["FullName"]) == false)
        AccountName.Text = (string)AccountEntry["FullName"];
    if (DBNull.Value.Equals(AccountEntry["AnnualFee"]) == false)
        AnnualFee.Text = string.Format("{0:0.00}",
            (decimal)AccountEntry["AnnualFee"]);
    if (DBNull.Value.Equals(AccountEntry["StartDate"]) == false)
        StartDate.Text = string.Format("{0:d}",
            (DateTime)AccountEntry["StartDate"]);
}
```

3. Locate the *ActOK_Click* routine. In the *Try* block, just after the "Save the changes in the record" comment, you'll find the following code line:

```
workArea.BeginEdit();
```

Just after that line, add the following code, which updates an existing *DataRow* instance with the user's input:

```
workArea["Active"] = ActiveAccount.Checked;
if (AccountName.Text.Trim().Length == 0)
    workArea["FullName"] = DBNull.Value;
else
    workArea["FullName"] = AccountName.Text.Trim();
if (AnnualFee.Text.Trim().Length == 0)
    workArea["AnnualFee"] = DBNull.Value;
else
    workArea["AnnualFee"] = decimal.Parse(AnnualFee.Text);
if (StartDate.Text.Trim().Length == 0)
    workArea["StartDate"] = DBNull.Value;
else
    workArea["StartDate"] = DateTime.Parse(StartDate.Text);
```

Modifying Existing Rows in a *DataTable*: Visual Basic

Note This exercise uses the "Chapter 3 VB" sample project and continues the preceding exercise in this chapter.

1. Open the source code view for the *AccountDetail* form. Locate the *AccountDetail_Load* routine.
2. Add the following code, which fills in the form's display fields with content from an existing *DataRow* instance:

```
If (AccountEntry IsNot Nothing) Then
    AccountID.Text = CStr(AccountEntry!ID)
    ActiveAccount.Checked = CBool(AccountEntry!Active)
    If (IsDBNull(AccountEntry!FullName) = False) Then _
        AccountName.Text = CStr(AccountEntry!FullName)
    If (IsDBNull(AccountEntry!AnnualFee) = False) Then _
        AnnualFee.Text = Format(CDec(AccountEntry!AnnualFee), "0.00")
    If (IsDBNull(AccountEntry!StartDate) = False) Then _
        StartDate.Text = Format(CDate(AccountEntry!StartDate), "Short Date")
End If
```

3. Locate the *ActOK_Click* routine. In the *Try* block, just after the "Save the changes in the record" comment, you'll find the following code line:

```
workArea.BeginEdit()
```

Just after that line, add the following code, which updates an existing *DataRow* instance with the user's input:

```
workArea!Active = ActiveAccount.Checked
If (AccountName.Text.Trim.Length = 0) _
    Then workArea!FullName = DBNull.Value _
    Else workArea!FullName = AccountName.Text.Trim
If (AnnualFee.Text.Trim.Length = 0) _
    Then workArea!AnnualFee = DBNull.Value _
    Else workArea!AnnualFee = CDec(AnnualFee.Text)
If (StartDate.Text.Trim.Length = 0)
    Then workArea!StartDate = DBNull.Value _
    Else workArea!StartDate = CDate(StartDate.Text)
```

Removing Data

You remove *DataRow* objects from a table via the *DataTable.Rows* collection's *Remove* and *RemoveAt* methods. The *Remove* method accepts an instance of a row that is currently in the table.

C#

```
DataRow oneRow = someTable.Rows[0];
someTable.Rows.Remove(oneRow);
```

Visual Basic

```
Dim oneRow As DataRow = someTable.Rows(0)
someTable.Rows.Remove(oneRow)
```

The *RemoveAt* method also removes a row, but you pass it the index position of the row to delete.

C#

```
someTable.Rows.RemoveAt(0);
```

Visual Basic

```
someTable.Rows.RemoveAt(0)
```

If you have an instance of a data row available, but you want to call the *RemoveAt* method, you can obtain the index of the row from the *Rows* collection's *IndexOf* method.

C#

```
int rowPosition = someTable.Rows.IndexOf(oneRow);
```

Visual Basic

```
Dim rowPosition As Integer = someTable.Rows.IndexOf(oneRow)
```

You can put any row you remove from a table right back into the *Rows* collection by using the standard *DataTable.Rows.Add* method. Another *Rows* method, *InsertAt*, adds a *DataRow* object to a table, but lets you indicate the zero-based position of the newly added row. (The *Add* method always puts new rows at the end of the collection.) The following code inserts a row as the first item in the collection:

C#

```
someTable.Rows.InsertAt(oneRow, 0);
```

Visual Basic

```
someTable.Rows.InsertAt(oneRow, 0)
```

To remove all rows from a table at once, use the *DataTable.Rows* object's *Clear* method.

C#

```
someTable.Rows.Clear();
```

Visual Basic

```
someTable.Rows.Clear()
```

As convenient as *Remove*, *RemoveAt*, and *Clear* are, they come with some negative side effects. Because they fully remove a row and all evidence that it ever existed, these methods prevent ADO.NET from performing certain actions, including managing record removes within an external database. The next section, "Batch Processing," discusses a better method of removing data records from a *DataTable* instance.

Batch Processing

The features shown previously for adding, modifying, and removing data records within a *DataTable* all take immediate action on the content of the table. When you use the *Add* method to add a new row, it's included immediately. Any field-level changes made within rows are stored and considered part of the record—assuming that no data-specific exceptions get thrown during the updates. After you remove a row from a table, the table acts as if it never existed.

Although this type of instant data gratification is nice when using a *DataTable* as a simple data store, sometimes it is preferable to postpone data changes or make several changes at once, especially when you need to verify that changes occurring across multiple rows are collectively valid.

ADO.NET includes table and row-level features that let you set up "proposed" changes to be accepted or rejected en masse. When you connect data tables to their external database counterparts in later chapters, ADO.NET uses these features to ensure that updates to both the local copy of the data and the remote database copy retain their integrity. You can also use them for your own purposes, however, to monitor changes to independent *DataTable* instances.

Note In reality, ADO.NET always uses these batch processing monitoring tools when changes are made to any rows in a table, even changes such as those in this chapter's simple code samples. Fortunately, the Framework is designed so that you can safely ignore these monitoring features if you don't need them.

To use the batch system, simply start making changes. When you are ready to save or reject all changes made within a *DataTable*, call the table's *AcceptChanges* method to commit and approve all pending changes, or call the *RejectChanges* method to discard all unsaved changes. Each *DataRow* in the table also includes these methods. You can call the row-level methods directly, but the table-level methods automatically trigger the identically named methods in each modified row.

C#

```
someTable.AcceptChanges();  // Commit all row changes
someTable.RejectChanges();  // Reject changes since last commit
```

Visual Basic

```
someTable.AcceptChanges()  ' Commit all row changes
someTable.RejectChanges()  ' Reject changes since last commit
```

Row State

While making your row-level edits, ADO.NET keeps track of the original and proposed versions of all fields. It also monitors which rows have been added to or deleted from the table, and can revert to the original row values if necessary. The Framework accomplishes this by managing various state fields for each row. The main tracking field is the *DataRow.RowState* property, which uses the following enumerated values:

- ***DataRowState.Detached*** The default state for any row that has not yet been added to a *DataTable*.
- ***DataRowState.Added*** This is the state for rows added to a table when changes to the table have not yet been confirmed. If you use the *RejectChanges* method on the table, any added rows will be removed immediately.
- ***DataRowState.Unchanged*** The default state for any row that already appears in a table, but has not been changed since the last call to *AcceptChanges*. New rows created with the *NewRow* method use this state.
- ***DataRowState.Deleted*** Deleted rows aren't actually removed from the table until you call *AcceptChanges*. Instead, they are marked for deletion with this state setting. See the following discussion for the difference between "deleted" and "removed" rows.
- ***DataRowState.Modified*** Any row that has had its fields changed in any way is marked as modified.

Every time you add or modify a record, the data table updates the row state accordingly. However, removing records from a data table with the *Rows.Remove* and *Rows.RemoveAt* methods circumvents the row state tracking system, at least from the table's point of view.

To enable ADO.NET batch processing support on deleted rows, use the *DataRow* object's *Delete* method. This does not remove the row from the *DataTable.Rows* collection. Instead, it marks the row's state as deleted. The next time you use the table or row *AcceptChanges* method to confirm all updates, the row will be removed permanently from the table.

If you want to use the batch processing features, or if your *DataTable* instances are associated with a database table, even if that table is temporarily disconnected, you need to use the row-specific *Delete* method instead of *Remove* and *RemoveAt*.

C#

```
someTable.Rows.Remove(oneRow); // Removes row immediately
oneRow.Delete();               // Marks row for removal during approval
```

Visual Basic

```
someTable.Rows.Remove(oneRow) ' Removes row immediately
oneRow.Delete()               ' Marks row for removal during approval
```

If you retain a reference to a deleted row once it has been removed from the table, its *RowState* property will be set to *DataRowState.Detached*, just like a new row that has not yet been added to a table.

Note When working with *DataTable* instances that are connected to true database tables, ADO.NET will still allow you to use the *Remove* and *RemoveAt* methods. However, these methods will not remove the row from the database-side copy. They remove only the local *DataTable.Rows* copy of the row. You must use the row's *Delete* method to ensure that the row gets removed from the database upon acceptance.

Row Versions

When you make changes to data within a data table's rows, ADO.NET keeps multiple copies of each changed value. Row version information applies to an entire row, even if only a single data column in the row has changed.

- ***DataRowVersion.Original*** The starting value of the field before it was changed (the value that was in effect after the most recent use of *AcceptChanges* occurred).
- ***DataRowVersion.Proposed*** The changed but unconfirmed field value. The *Proposed* version of a field doesn't exist until you begin to make edits to the field or row. When changes are accepted, the proposed value becomes the actual (*Original*) value of the field.

- ***DataRowVersion.Current*** For fields with pending edits, this version is the same as *Proposed*. For fields with confirmed changes, the *Current* version is the same as the *Original* version.
- ***DataRowVersion.Default*** For rows attached to a *DataTable*, this version is the same as *Current*. For detached rows, *Default* is the same as *Proposed*. The *Default* version of a row is not necessarily the same as the default values that might appear in newly created rows.

Depending on the current state of a row, some of these row versions might not exist. To determine whether a row version does exist, use the *DataRow.HasVersion* method. The following code block uses the *HasVersion* method and a special overload of the *Item* method to access various row versions:

C#

```
if (oneRow.HasVersion(DataRowVersion.Proposed))
{
    if (oneRow.Item["Salary", DataRowVersion.Original] !=
            oneRow.Item["Salary", DataRowVersion.Proposed])
        MessageBox.Show("Proposed salary change.");
}
```

Visual Basic

```
If (oneRow.HasVersion(DataRowVersion.Proposed) = True) Then
    If (oneRow.Item("Salary", DataRowVersion.Original) <>
            oneRow.Item("Salary", DataRowVersion.Proposed)) Then _
        MessageBox.Show("Proposed salary change.")
End If
```

The default row version returned by the *Item* property is the *Current* version.

Validating Changes

As mentioned earlier in the "Storing Rows in a Table" section on page 40, attempting to store data of the incorrect data type in a column will throw an exception. However, this will not prevent a user from entering invalid values, such as entering 7,437 into a *System.Int32* column that stores a person's age. Instead, you must add validation code to your data table to prevent such data range errors.

Exception-Based Errors

ADO.NET monitors some data errors on your behalf. These errors include the following:

- Assigning data of the wrong data type to a column value.
- Supplying string data to a column that exceeds the maximum length defined in the column's *DataColumn.MaxLength* property.
- Attempting to store a NULL value in a primary key column.
- Adding a duplicate value in a column that has its *Unique* property set.
- Taking data actions that violate one of the table's custom constraints. Constraints are discussed in Chapter 5, "Bringing Related Data Together."

When one of these data violations occurs, ADO.NET throws an exception in your code at the moment when the invalid assignment or data use happens. The following code block generates an exception because it attempts to assign a company name that is too long for a 20-character column:

C#

```
DataColumn withRule = someTable.Columns.Add("FullName", typeof(string));
withRule.MaxLength = 20;
// ...later...
DataRow rowToUpdate = someTable.Rows[0];
rowToUpdate["FullName"] = "Graphic Design Institute"; // 24 characters
  // ----- Exception occurs with this assignment
```

Visual Basic

```
Dim withRule As DataColumn = someTable.Columns.Add("FullName", GetType(String))
withRule.MaxLength = 20
' ...later...
Dim rowToUpdate As DataRow = someTable.Rows(0)
rowToUpdate!FullName = "Graphic Design Institute"  ' 24 characters
  ' ----- Exception occurs with this assignment
```

Naturally, you would want to enclose such assignments within exception handling blocks. However, there are times when adding this level of error monitoring around every field change is neither convenient nor feasible—not to mention the impact it has on performance when adding or updating large numbers of records. To simplify exception monitoring, the *DataRow* class includes the *BeginEdit* method. When used, any data checks normally done at assignment are postponed until you issue a matching *DataRow.EndEdit* call.

C#

```
rowToUpdate.BeginEdit();
rowToUpdate["FullName"] = "Graphic Design Institute"; // 24 characters
  // ----- No exception occurs
rowToUpdate["RevisionDate"] = DateTime.Today; // Other field changes as needed
rowToUpdate.EndEdit();
  // ----- Exception for FullName field occurs here
```

Visual Basic

```
rowToUpdate.BeginEdit()
rowToUpdate!FullName = "Graphic Design Institute"  ' 24 characters
  ' ----- No exception occurs
rowToUpdate!RevisionDate = Today  ' Other field changes as needed
rowToUpdate.EndEdit()
  ' ----- Exception for FullName field occurs here
```

To roll back any changes made to a row since using *BeginEdit*, use the *CancelEdit* method. Even when you complete a row's changes with the *EndEdit* method, the changes are not yet committed. It is still necessary to call the *AcceptChanges* method, either at the row or the table level.

The *DataTable.AcceptChanges* and *DataRow.AcceptChanges* methods, when used, automatically call *DataRow.EndEdit* on all rows with pending edits. Similarly, *DataTable.RejectChanges* and *DataRow.RejectChanges* automatically issue calls to *DataRow.CancelEdit* on all pending rows changes.

Validation-Based Errors

For data issues not monitored by ADO.NET, including business rule violations, you must set up the validation code yourself. Also, you must manually monitor the data table for such errors and refuse to confirm changes that would violate the custom validation rules.

Validation occurs in the various event handlers for the *DataTable* instance that contains the rows to monitor. Validation of single-column changes typically occurs in the *ColumnChanging* event. For validation rules based on the interaction between multiple fields in a data row, use the *RowChanging* event. The *RowDeleting* and *TableNewRow* events are useful for checking data across multiple rows or the entire table. You can also use any of the other table-level events (*ColumnChanged*, *RowChanged*, *RowDeleted*, *TableClearing*, *TableCleared*) to execute validation code that meets your specific needs.

Inside the validation event handlers, use the *Proposed* version of a row value to assess its preconfirmed value. When errors occur, use the row's *SetColumnError* method (with the name, position, or instance of the relevant column) to indicate the problem. For row-level errors, assign a description of the problem to the row's *RowError* property. The following code applies a column-level business rule to a numeric field, setting a column error if there is a problem:

C#

```
private void Applicant_ColumnChanging(Object sender,
    System.Data.DataColumnChangeEventArgs e)
{
    // ----- Check the Age column for a valid range.
    if (e.Column.ColumnName == "Age")
    {
        if ((int)e.ProposedValue < 18 || (int)e.ProposedValue > 29)
            e.Row.SetColumnError(e.Column,
                "Applicant's age falls outside the allowed range.");
    }
}
```

Visual Basic

```
Private Sub Applicant_ColumnChanging(ByVal sender As Object,
        ByVal e As System.Data.DataColumnChangeEventArgs)
    ' ----- Check the Age column for a valid range.
    If (e.Column.ColumnName = "Age") Then
        If (CInt(e.ProposedValue) < 18) Or (CInt(e.ProposedValue) > 29) Then
            e.Row.SetColumnError(e.Column,
                "Applicant's age falls outside the allowed range.")
        End If
    End If
End Sub
```

Adding column or row-level errors sets both the *DataRow.HasErrors* and the *DataTable.HasErrors* properties to *True*, but that's not enough to trigger an exception. Instead, you need to monitor the *HasErrors* properties before confirming data to ensure that validation rules are properly applied. Another essential method, *ClearErrors*, removes any previous error notices from a row.

C#

```
// ----- Row-level monitoring.
oneRow.ClearErrors();
oneRow.BeginEdit();
oneRow.FullName = "Tailspin Toys";  // Among other changes
if (oneRow.HasErrors)
{
    ShowFirstRowError(oneRow);
    oneRow.CancelEdit();
}
else
    oneRow.EndEdit();

// ----- Table-level monitoring. Perform row edits, then...
if (someTable.HasErrors)
{
    DataRow[] errorRows = someTable.GetErrors();
    ShowFirstRowError(errorRows[0]);
    someTable.RejectChanges();  // Or, let user make additional corrections
}
```

```
else
    someTable.AcceptChanges();

// ...later...

public void ShowFirstRowError(DataRow whichRow)
{
    // ----- Show first row-level or column-level error.
    string errorText = "No error";
    DataColumn[] errorColumns = whichRow.GetColumnsInError();

    if (errorColumns.Count > 0)
        errorText = whichRow.GetColumnError(errorColumns[0]);
    else if (whichRow.RowError.Length > 0)
        errorText = whichRow.RowError;
    if (errorText.Length == 0) errorText = "No error";
    MessageBox.Show(errorText);
}
```

Visual Basic

```
' ----- Row-level monitoring.
oneRow.ClearErrors()
oneRow.BeginEdit()
oneRow.FullName = "Tailspin Toys"  ' Among other changes
If (oneRow.HasErrors = True) Then
    ShowFirstRowError(oneRow)
    oneRow.CancelEdit()
Else
    oneRow.EndEdit()
End If

' ----- Table-level monitoring. Perform row edits, then...
If (someTable.HasErrors = True) Then
    Dim errorRows() As DataRow = someTable.GetErrors()
    ShowFirstRowError(errorRows(0))
    someTable.RejectChanges()  ' Or, let user make additional corrections
Else
    someTable.AcceptChanges()
End If

' ...later...

Public Sub ShowFirstRowError(ByVal whichRow As DataRow)
    ' ----- Show first column-level or row-level error.
    Dim errorText As String = ""
    Dim errorColumns() As DataColumn = whichRow.GetColumnsInError()

    If (errorColumns.Count > 0) Then
        errorText = whichRow.GetColumnError(errorColumns(0))
    ElseIf (whichRow.RowError.Length > 0) Then
        errorText = whichRow.RowError
    End If
    If (errorText.Length = 0) Then errorText = "No error"
    MessageBox.Show(errorText)
End Sub
```

Validating Content in a *DataRow*: C#

Note This exercise uses the “Chapter 3 CSharp” sample project and continues the preceding exercise in this chapter.

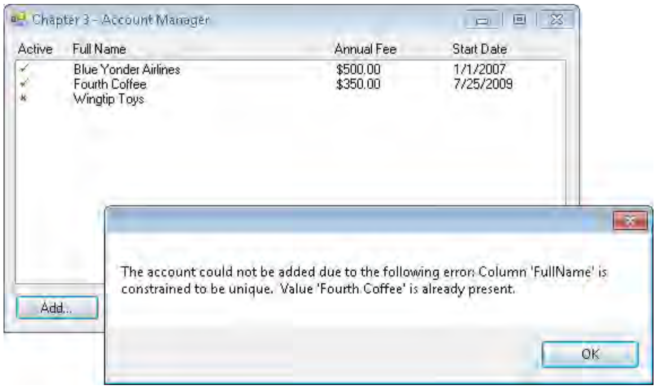
1. Open the source code view for the *AccountManager* form.
2. Locate the *CustomerAccounts_ColumnChanging* event handler, which is called whenever a column value in a *CustomerAccounts* table row changes. Add the following code, which checks for valid data in two of the columns:

```
if (e.Column.ColumnName == "AnnualFee")
{
    // ----- Annual fee may not be negative.
    if (DBNull.Value.Equals(e.ProposedValue) == false)
    {
        if ((decimal)e.ProposedValue < 0m)
            e.Row.SetColumnError(e.Column,
                "Annual fee may not be negative.");
    }
}
else if (e.Column.ColumnName == "StartDate")
{
    // ----- Start date must be on or before today.
    if (DBNull.Value.Equals(e.ProposedValue) == false)
    {
        if (((DateTime)e.ProposedValue).Date > DateTime.Today)
            e.Row.SetColumnError(e.Column,
                "Start date must occur on or before today.");
    }
}
```

3. Locate the *CustomerAccounts_RowChanging* event handler, which is called whenever any value in a row changes within the *CustomerAccounts* table. Add the following code, which checks for valid data involving multiple columns:

```
if (e.Row.HasVersion(DataRowVersion.Proposed) == true)
{
    if (((bool)e.Row["Active", DataRowVersion.Proposed] == true) &&
            (DBNull.Value.Equals(e.Row["StartDate",
            DataRowVersion.Proposed]) == true))
        e.Row.RowError = "Active accounts must have a valid start date.";
}
```

4. Run the program. Use its features to create, edit, and delete data rows. When you attempt to provide invalid data—incorrect data types, violations of business rules, duplicate account names—the program provides the appropriate error messages.



Active	Full Name	Annual Fee	Start Date
✓	Blue Yonder Airlines	$500.00	1/1/2007
✓	Fourth Coffee	$350.00	7/25/2009
*	Wingtip Toys		

The account could not be added due to the following error: Column 'FullName' is constrained to be unique. Value 'Fourth Coffee' is already present.

Validating Content in a *DataRow*: Visual Basic

Note This exercise uses the "Chapter 3 VB" sample project and continues the preceding exercise in this chapter.

1. Open the source code view for the *AccountManager* form.

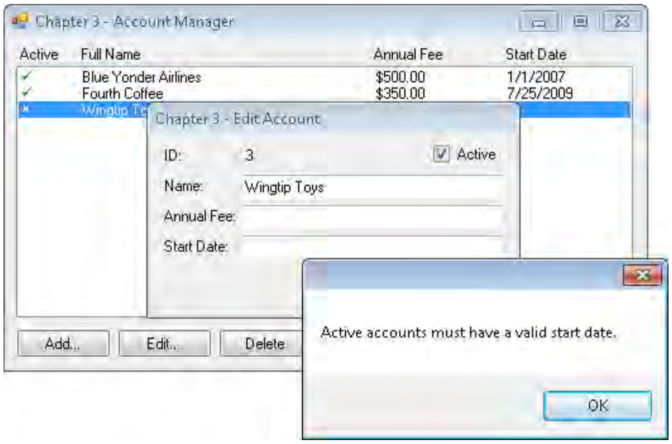
2. Locate the *CustomerAccounts_ColumnChanging* event handler, which is called whenever a column value in a *CustomerAccounts* table row changes. Add the following code, which checks for valid data in two of the columns:

```
If (e.Column.ColumnName = "AnnualFee") Then
    ' ----- Annual fee may not be negative.
    If (IsDBNull(e.ProposedValue) = False) Then
        If (CDec(e.ProposedValue) < 0@) Then _
            e.Row.SetColumnError(e.Column,
            "Annual fee may not be negative.")
    End If
ElseIf (e.Column.ColumnName = "StartDate") Then
    ' ----- Start date must be on or before today.
    If (IsDBNull(e.ProposedValue) = False) Then
        If (CDate(e.ProposedValue).Date > Today) Then _
            e.Row.SetColumnError(e.Column,
            "Start date must occur on or before today.")
    End If
End If
```

3. Locate the *CustomerAccounts_RowChanging* event handler, which is called whenever any value in a row changes within the *CustomerAccounts* table. Add the following code, which checks for valid data involving multiple columns:

```
If (e.Row.HasVersion(DataRowVersion.Proposed) = True) Then
    If (CBool(e.Row("Active", DataRowVersion.Proposed)) = True) And
            (IsDBNull(e.Row("StartDate",
            DataRowVersion.Proposed)) = True) Then
        e.Row.RowError = "Active accounts must have a valid start date."
    End If
End If
```

4. Run the program. Use its features to create, edit, and delete data rows. When you attempt to provide invalid data—incorrect data types, violations of business rules, duplicate account names—the program provides the appropriate error messages.



Active	Full Name	Annual Fee	Start Date
✓	Blue Yonder Airlines	$500.00	1/1/2007
✓	Fourth Coffee	$350.00	7/25/2009
×	Wingtip To		

ID: 3 ☑ Active
Name: Wingtip Toys
Annual Fee:
Start Date:

Active accounts must have a valid start date.

Summary

This chapter discussed the *DataRow* class, the final destination of all data in ADO.NET. With one instance created per data record, the *DataRow* class manages each individual columnar field. When used alone, its stored values use the generic *Object* data type, but when inserted into a *DataTable* object's *Rows* collection, all data type limitations and other constraints established for the table's columns act together to verify the row's content.

Beyond these column settings, you can add event handlers to the *DataTable* that apply custom business rules to the column and row data, providing an additional layer of validation—and ultimately, integrity—for the table's data.

Chapter 3 Quick Reference

To	Do This
Add a row to a *DataTable*	Use the *DataTable* object's *NewRow* method to obtain a *DataRow* instance. Update the *Item* values in the *DataRow* as needed. Add the row using the table's *Rows.Add* method.
Delete a row from a *DataTable*	Call the *DataRow* object's *Delete* method. Call the *DataTable* object's *AcceptChanges* method.
Check for data issues in new and modified *DataRow* objects	Create a *DataTable*. Add *DataColumn* definitions as needed. Add event handlers for the *DataTable* object's *ColumnChanging*, *RowChanging*, or other events. In the handlers, call the *DataRow* object's *SetColumnError* method or update its *RowError* property.
Temporarily suspend data validation while modifying a data row	Call the *DataRow* object's *BeginEdit* method. Update the *Item* values in the *DataRow* as needed. Call the *DataRow* object's *EndEdit* method.

Index

Symbols

$filter option 380
$orderby option 380
$select option 380
$value option 383
* (asterisk) symbol, using as entity name 376
.csdl file extensions 217
{ } (curly braces), building custom collections using 254
.dbml file extensions 336
.edmx file extensions 217, 218, 325
= (equal sign), comparing columns to literal values using 63
>= (greater than or equal to sign)
  comparing columns to literal values using 63
  in Where clause (LINQ) 296
> (greater than sign)
  comparing columns to literal values using 63
<> (inequality sign)
  comparing columns to literal values using 63
<= (less than or equal to sign)
  comparing columns to literal values using 63
< (less than sign)
  comparing columns to literal values using 63
  in Where clause (LINQ) 296
.NET applications, types of configuration files 13
.NET developers, ADO.NET and 3
.NET Framework. *See also* Entity Framework (EF)
  ADO.NET in 213
  connection string builders in 126
  data providers 126–127
  strongly typed DataSets in 214
.NET objects, tools 8
( ) parentheses
  in expression evaluation 63
  using in Where clause (LINQ) 296
@-prefixed placeholders 155, 157, 161, 167, 178
' ' (single quotes)
  using BINARY keyword with 250
  using GUID keyword with 250
  using strings with 249
\SQLEXPRESS
  appended with SQL Server 2008 Express Edition instances 12
.ssdl file extensions 217, 325
- (subtraction) operators, in Entity SQL language 250
.svc service files 370
.tt (text templates) file extensions 241
.xsd file extensions
  created from Connection Wizard 14
  creating tables with mouse 28

A

ABS (absolute value) 251
AcceptChanges method 48, 57, 99
Access, provider class libraries for 126
ACID
  rules 192–193
  with transactions 204
Acos function 323
acronym, ADO.NET 4
Add Connection dialog box 12
Add Entity dialog box 231–232
Add Function Import dialog box 233–234
Add... functions
  in Entity SQL language 251
  in LINQ to Entities 322
adding
  aggregate columns 94–95
  BindingNavigator control to Windows forms 353
  calculated columns 71
  connections to databases 32
  data columns 21–28
  data rows to tables 37–41
  Entity Framework model to projects 243

expression columns 68–70
mapping condition to entity 237–239
navigation buttons to WPF window 357–360
new entities through objects 271–272
relationships between two tables 79
tables to DataSets 75
Add New Item dialog box 28
AddObject method 272
add operator in REST 380
AddWithValue method 157
ADO.NET
about 3–5
components of 5–7
extensions to 7–8
prior versions of, Oracle providers in 127
ADO.NET Data Services 369
ADO.NET Entity Data Model Designer
about 218
generating objects using 220
mapping details panel, working with 235–240
using 230–236
ADO.NET Entity Data Model Wizard 218, 225–229, 325, 372
ADO.NET EntityObject Generator 241
ADO.NET Self-Tracking EntityObject Generator 241
ADO vs. ADO.NET 4
Aggregate clauses 301, 302
aggregate functions 252–254, 301–302
aggregating data 89–98
adding aggregate columns 94–95
functions for 89–90
generating
single aggregates 91–94
summaries 95–98
referencing parent fields in expressions 98
aliases, using in Entity SQL language 247
All
as EntitySetRights member 376
function 301
AllowDBNull, DataColumn class property 23
AllowDelete Boolean properties 101
AllowEdit Boolean properties 101
AllowNew Boolean properties 101
AllRead, as EntitySetRights member 376
AllWrite, as EntitySetRights member 376
AND operator 250, 296
and operator in REST 380
anonymous type definition (new {}) 294
anonymous types 290
Any function 301
application configuration files, modifying settings in 13
Application Name key 123
"applies" as keyword in Entity SQL language 248–249
arguments, XmlWriteMode 109
arrays
of DataRow instances
in Select method 98
of DataRowView instances
in FindRows method 102
ASC (ascending sorts) 65
ascending sorts (ASC) 65
Ascii function 323
ASC modifier 248
AsEnumerable extension method 306
Asin function 323
AsNonUnicode function 322
ASP.NET
applications 371
data binding in 362–366
services 370
association ends, Entity Framework definition of 215
associations
editing 232–233
edit mappings 237
Entity Framework definition of 215, 216
sets of, Entity Framework definition 216
asterisk (*) symbol, using as entity name 376
AsUnicode function 322
Atan2 function 323
Atan function 323
Atomicity rule 192
AtomPub (Atom Publishing Protocol) 370, 377–378, 382
AttachDBFilename key 123
AutoIncrement, DataColumn class property 23, 38
AutoIncrementSeed, DataColumn class property 23

AutoIncrementStep, DataColumn class property 23
auto-update EF model 233
Average function 301
Avg function 89, 252

B

batch processing 46–49
BeginEdit method 50–53, 57
BeginExecuteNonQuery method 139–141
BeginExecuteXmlReader method 146
BeginTransaction method 196
BINARY keyword 250
BindingContext instances 353
binding data
  about 347
  in ASP.NET 362–366
  in Windows forms 347–354
    creating complex forms 348
    creating simple forms 351–352
    understanding 352–354
  in Windows Presentation Foundation (WPF) 352–362
    creating data-bound applications 355–361
BindingNavigator control 353
BindingSource class 353
Bitwise functions 251
BLOBs 281
Boolean functions
  All 301
  Any 301
  Contains 251
  EndsWith 251
  StartsWith 251
Boolean operators
  AND 63, 296, 380
  NOT 63, 296, 380
  OR 63, 296, 380
Boolean properties
  AllowDelete 101
  AllowEdit 101
  AllowNew 101
Boolean values
  true and false keywords as 249
BuildConnection function 129

C

C#
  accessing data through model-generated objects 268–270
  accessing field values 144
  adding aggregate columns 95
  adding BindingNavigator control to Windows forms 353–354
  adding columns to DataTables 25–26
  adding constraints manually 83
  adding database tables, using DataSet Designer 32–33
  adding DelimSubstring method to System.String data 278
  adding expression columns 68–69
  adding new entities through objects 271–272
  adding parameters to a command 156
  adding relationships between two tables 79
  adding rows to DataTables 41
  adding tables to Data Sets 75
  AddWithValue method in setting parameters 157
  building connection strings 125
  calling AddLocation stored procedure 162–163
  calling BeginExecuteNonQuery method 139
  calling Complete method of TransactionScope 205–206
  calling DeleteObject method 273
  calling EndExecuteNonQuery method 139
  calling stored procedures with parameters 163–165
  computing aggregate values 91–93
  configuring update commands 175–176
  creating command objects 136
  creating custom tables, in DataSet Designer 28–32
  creating data bound WPF applications 355–361
  creating data service from EF model 372–373
  creating DataSet objects 73–74
  creating DataTables 19
  creating data views 99–101
  creating instances of TransactionScope 204
  creating new instance of command builder 180
  creating SQL Server connections 128–129

defining update and delete rules in DataRelations 85–86
employing savepoints 203
finding rows by primary key 60–61
generating DataTables from DataView 103–104
generating EF objects 220–221
generating summaries 96
generating XML from DataSet 114–115
GetOrdinal method in 144
implementing many-to-many relationships based on primary keys 83–84
LINQ- specific language features 8
locating parent-child records 80
modifying databases through entity objects 274–276
modifying existing properties 271
modifying rows in DataTable 43–44
moving data into DataSets 173–174
moving data into DataTables 171–173
nesting child tables 113
null values in 290
opening SQL Server connections 129–130
processing database queries 146–147
processing with distributed transactions 206–207
processing with local transactions 198–200
ReadXml in 110
referencing parent fields in expressions 98
removing DataRow objects 45–46
retrieving entity data through ObjectQuery 257–258
retrieving entity data through provider 261–263
returning data rows 142–143
running nonqueries 138–139
selecting and sorting DataRow objects 65–66
SELECT queries, returning single values 141–142
syncing data with SqlDataAdapter 181–183
System.DBNull in 40
this keyword and 280
transactions on open database connections 196
updating Data Tables 179
updating data with parameters 158–159
using batch processing 47
using BeginEdit 51
using ColumnMapping property 114
using Commit and Rollback methods 197
using CreateObjectSet method 222
using DataRow class types 38
using DataRow.HasVersion method 49
using Dataset Designer with 27
using DataTableMapping objects 186–187
using ExecuteReader method to generate data reader 260
using LINQ in
  applying set operations 302–303, 303
  calling custom database functions 326–327
  creating implicit joins 306–307
  filtering results with Where clause 296
  limiting data returned 299–300
  projecting results with Select clause 294
  querying LINQ to SQL 340–341
  querying the data set 309–310
  querying to Entities 317–319
  selecting linked results with Join keyword 298–299
  sorting results with OrderBy clause 297
  starting queries with From clause 293
  summarizing data using aggregates 301–302
  to SQL provider structure 334–336
  using database functions 323
  using EntityFunctions class 323

Where clause not supported 321–322
using namespace elements 19
using namespace properties in 112–113
using query builder methods 283
using Rollback method 203
validating data 51–55
viewing source code 233
WriteXML method in 108–109
calculated columns, adding 71
calling
custom database functions 326–329
DataRow.EndEdit 50, 51
Dispose 205
EndExecuteNonQuery method 139
SaveChanges 273
stored procedures 138, 161
Cancel method 140
canonical functions, using 323
Caption, DataColumn class property 24
cardinality, relational database modeling term 76
cascade deletes in parent-child relationships, enforcing 88
CASE block, in Entity SQL language 252
case-sensitive lookups, performing 67
CAST function 251
Ceiling function 251
Char function 323
CharIndex functions 323
Checksum function 323
Child keywords to table relationships 95–96
child-parent data, generating hierarchal 118
child-parent topics. *See* parent-child
Child prefixes 94
child records, locating 79–81
child tables
nesting 113
Choose Data Source dialog box 11
classes
BindingSource 353
ComplexObject 268
DataContext 334
DataRowExtensions 305
DataTableExtensions 305
DbTransaction 195
EdmFunctionAttribute 325
EntityFunctions 322, 323
EntityObject 220, 268
ObjectContext 268
ObjectQuery 256–259
ObjectQuery(Of T) 268, 283, 315
ObjectSet(Of TEntity) 268, 315
OdbcTransaction 195
OleDbTransaction 195
SqlCommand 136–137
SqlDataAdapter 172
SqlFunctions 323
SqlParameter 161
SQLParameter 157
SqlTransaction 195
System.Data
Constraint 81–87
DataRelation 78
TransactionScope 204
within data providers 127
classes in DataTables 18–21
DataColumn
about 21–22
ColumnMapping property 114
Namespace property in 111–113
properties in 23–24
DataRow
about 18–20, 37
configuring 38
entries 6
HasVersion method 49
methods in retrieving current child rows of data 79–81
removing objects 45–46
validating content in 54–56
DataSet
Namespace property in 111–113
DataTable
about 19, 24–25
in LINQ queries 305, 306
Namespace property 111–113
supporting expression columns 67
DataView
about 99–100
creating 99–101
using 101–105
class libraries, providers 126–127
clearing data in Entity Framework
using functions in Entity SQL language 250–252
CLR (Common Language Runtime) 324
code generation items 241
Code Generation Strategy property 242
collections, entity-based 254

ColumnAttribute 334
ColumnChanged event 51
ColumnChanging event 51
column errors, setting 51
ColumnMapping property 114
ColumnName, DataColumn class property 24
column names
  as filter expression element 63
columns
  data types in bound relationship 78
  expression, using 67–70
  managing and positioning 113–117
  mapping with external database tables 186–188
  rowversion 195
command builders, using 181
Command classes, SQL Server 127
command objects, creating 136–137
commands
  CREATE FUNCTION DDL 324
commands, Update Model From Database shortcut 233
CommandText field, types of string data accepted 137
command trees 280
CommandType property 137
Commit method 196–197
commit/rollback support
  MSDTC distributed transactions and 204
Common Language Runtime (CLR) 324
Common Type System, of NET
  ADO.NET and 7
Complete method 205–206
ComplexObject class 268
complex properties, crafting 233, 234–235
components of ADO.NET 5–7
Compute method
  calculating aggregate of his single table columns 91
  Parent and Child prefixes and 94
Concat function 251
conceptual models (conceptual layers)
  focus of Entity Data Model Designer 235
  in Entity Framework 215, 217–218
  linking with storage models 226
  using 218–219
Conceptual Schema Definition Language (CSDL) 217, 218–219, 226, 235, 242, 268
concurrency
  data transactions and 191–195
  definition of 194
configuration files for .NET Applications 13
connecting to SQL Server via data providers 127–132
Connection classes, SQL Server 127
Connection objects 7
connection pooling 7, 129, 132
connection strings
  building 226
  content for building 124–126
  identifying data sources using 13
  using 121–126
Connection Timeout key 124
Connection Wizard, creating a data source using 8–14
Consistency rule 193
Constraint class, UniqueConstraint 81
Constraint instances, using 7
constraints, defining table 81–87
Contains function 251
Control keys, selecting properties using 234
CONVERT function 64
Cos function 323
Cot function 323
COUNT and BIGCOUNT functions 252
Count function 90, 301
Count method 283
count of records, getting 304
COUNT(*) syntax 252
CreateDateTime function 322
CreateDateTimeOffset function 322
CREATE FUNCTION DDL command 324
CreateObjectSet method 222
CreateTime function 322
creating
  C# data tables 19
  command objects 136–137
  complex types for use in entity 243
  custom tables

with DataSet Designer 28–32
database connections 128–129
data-bound ASP.NET applications 362–364
data-bound WPF applications 355–361
data reader 142–143
data relations 78–79
data service from EF model 372–375
DataSet objects 73–74
data sources
using Connection Wizard 8–14
DataTable objects 18
with Dataset Designer 28
DataViews 99–101
instances of command builder 180
properties within an entity 232
row objects 37–38
SqlDataReader object 149
Visual Basic data tables 20
CROSS JOIN keyword 248
cross joins 297
CRUD operations 4
CSDL (Conceptual Schema Definition Language) 217, 218–219, 226, 235, 242, 268
curly braces ({ }), building custom collections using 254
CurrentTimestamp function 323
CurrentUser function 323
custom DataTables, creating 28–34

D

data. *See also* external data
aggregating 89–98
adding aggregate columns 94–95
functions to 89–90
generating single aggregates 91–94
generating summaries 95–98
referencing parent fields in expressions 98
examining and changing 42–44
integrity of 191–208
employing savepoints 202–203
transactions and concurrency 191–195
using distributed transactions 204–208
using local transactions 195–203
managing 3
removing 45–46
tabular form
manipulating 4
DataAdapter
class
Entity Framework and 214
SQL Server 127
objects 7
understanding 169–171
data adapters
role in forms data binding 352
understanding 169–171
Database Explorer
adding connections to databases 32
O/R Designer and 336
databases
connecting to SQL Server 127–132
processing queries 146–149
querying 135–150
processing SQL queries 135–140
returning results 140–149
resolution of conflicts in 192–194
databases, normalization of 21–28
database tables
adding to DataSet Designer 32–34
data binding
about 347
in ASP.NET 362–366
in Windows forms 347–354
creating complex forms 348–350
creating simple forms 351–352
understanding 352–354
in Windows Presentation Foundation (WTF) 352–362
creating data-bound applications 355–361
DataBindings collection 353
DataColumn
adding expression columns by creating 68–69
definitions
data type declarations 6
including a DefaultValue setting 38
items, defining 6
objects

about 37–38
DataColumn class
about 21–22
ColumnMapping property 114
Namespace property in 111–113
properties in 23–24
Data Connections, in Server Explorer
adding connections to databases 32
DataContext class 334
data definition language (DDLs) 246
Data definition statements 136
data encryption, providing System.Security library 3
DataGridView control 350
DataLength function 323
data manipulation language (DML) 246
Data manipulation statements 136
data providers
connecting to SQL Server via 127–132
understanding 126–127
data range errors 49–56
DataReader classes, SQL Server 127
DataReader instance, retrieving records using SQLDataAdapter 170
DataReader objects 7
DataRelation class 78–79
DataRelation entries, linking 6
DataRelations
defining update and delete rules in 85–87
Entity Framework definition of 215
data relationships. *See* relational databases
DataRow class
about 18–20, 37
configuring 38
entries 6
HasVersion method 49
methods in retrieving parent-child rows of data 79–81
removing objects 45–46
validating content in 54–56
DataRow.EndEdit call 50, 51
DataRowExtensions class 305
DataRow.HasErrors property 52
DataRow instances
in LINQ 306
using Select method with 98
DataRow objects
Delete method 48
removing 45–46
selecting and sorting 65–68
data rows
adding process 37–44
creating rows 37–38
defining row values 38–40
storing rows in tables 40–41
roll back changes 51
DataRowState
.Added 47
.Deleted 47
.Detached 47, 48
.Modified 47
.Unchanged 47
DataRowVersion
.Current 49
.Default 49
.Original 48
.Proposed 48
DataRowView instances, generated from DataView 99
DataService(Of T) 372
data services
accessing using REST 377–384
setting up 371–376
types of 369–370
DataSet class
Namespace property 111–113
DataSet Designer
adding database tables 32–33
creating custom tables 28–32
using in code 33–34
using Toolbox with 29, 35
DataSet instance 353, 360
DataSet objects, creating 73–74
DataSets
adding tables to 75
column mapping with the external database tables 186–188
Entity Framework and 214
linking to external data 169–171
LINQ to 291
moving data into 173–175
moving data to source from 178
parallel for EF entities in 215
preventing incoming data from modifying schema of 189
tables grouped as 7, 73–76
DataSet tables

creating relationships between 78–79
Data Source Configuration Wizard 13, 355, 360
Data Source Connection Wizard 27, 122–124, 126
Data Source key 123
data sources
creating 8–14
removing 16
DataTable classes
DataColumn 21–24
about 21–22
ColumnMapping property 114
Namespace property in 111–113
properties in 23–24
DataRow
about 18–20, 37
configuring 38
entries 7
HasVersion method 49
methods in retrieving parent-child rows of data 79–81
removing objects 45–46
validating content in 54–56
DataSet
Namespace property 111–113
DataTable
about 19, 24–25
in LINQ queries 305, 306
Namespace property 111–113
supporting expression columns 67
DataView
about 99–100
creating 100–102
using 102–106
DataTable class type
about 18, 19
data-related events in 19
generating table-specific data rows 38
PrimaryKey property 24–25
DataTableExtensions class 305
DataTable.HasErrors property 52
DataTableMapping objects 186
DataTable objects
Compute method 91
populating Columns collection of 21–28
using Dataset Designer to create 28
DataTable.Rows.Find method 71
DataTables
about 17
adding to DataSets 75
as Key ADO.NET element 6–7
building 17–36
adding data columns 21–28
classes in. *See* classes in DataTables
implementing tables 17–21
using Dataset Designer 27–33
collecting into sets 73–76
column mapping with external database tables 186–188
DataView instances and 7
defining constraints 81–87
establishing relationships between 76–81
creating data relations 78–79
locating parent-child records 79–81
types of relationships 76–78
generating from DataView 103–105
logical implementation of objects 17–18
modifying names of 29
modifying TableName property 19
moving data into 171–173
moving data to source from 178–179
parallel for EF entities in 215
physical implementation of objects 18
searching rows
by primary key 60–62
with search criteria 62–64
storing data in 37–57
adding data process 37–41
batch processing 46–48
examining in changing data 42–44
removing data 45–46
validating changes 49–56
DataType, DataColumn class property 24
data types
columns bound in relationships and 78
user-defined, for entities 215
DataView class
about 99–100
creating 99–101
using 101–105
DataView instances
setting RowStateFilter properties 100

view of rows 7
DataView methods
Find 102
FindRows 101
ToTable 103, 105
DataViewRowsState, enumerated values of 100
DateAdd function 323
date and time functions 251, 322, 323
DateDiff function 323
DateName function 323
DatePart function 323
DATETIME keyword 249
DateTimeMod, DataColumn class property 24
DBNull
returning for nondata results 141
DbTransaction class 195
DbType property 157
DDLs (data definition language) 246
DefaultValue
automatic 40
setting in DataColumn definition 38
DefaultValue, DataColumn class property 24
defining
constraints for relationships 81–87
updated delete rules In DataRelations 85–87
Degrees function 323
Delete method 48, 57, 101
DeleteObject method 273
DELETE queries, adding in SqlDataAdapter 185
DeleteRule properties
setting 82
DELETE statements
building 274
generating automatically 180
update operations and 169–170, 171, 180
DelimSubstring method 278–279
DEREF function 255
DESC modifier 65, 248
dialog boxes
Add Connection 12
Add Entity 231
Add Function Import 233–234
Add New Item 28
Choose Data Source 11
Feed And Web Slice Setting 379
Unique Constraint 30–31
Difference function 323
Diff... functions 251, 322
Direction property in SQLParameter class 161
Dispose, calling 205
Dispose method 128
Distinct clause 299
DISTINCT keyword 254
Distinct method 281
distributed transactions, using 204–208, 209
div operator in REST 380
DML (data manipulation language) 246
double-precision floating-point value literal 249
drag-and-drop visual designer. *See* Object Relational (O/R) Designer
Durability rule 193

E

EDM (Entity Data Model)
adding 225
conceptual model in EF and 216
EdmFunction attribute 326, 328
EdmFunctionAttribute class 325
edmgen.exe 220, 223
EF (Entity Framework). *See* Entity Framework (EF)
ELSE clause
in Entity SQL language 252
embedded XML expressions 291
Encrypt and TrustServerCertificate keys 124
EndEdit, DataRow call 50, 51
EndEdit method 51, 57
EndExecuteNonQuery method 139–140
EndExecuteXmlReader method 146
EndsWith function 251
EnlistTransaction method 205
entities
editing 231–232
Entity Framework definition of 215, 216
importing database tables as 227–230
LINQ to. *See* LINQ (providers)
managing through objects 267–278
accessing entity data 268–270
modifying entity data 271–278
using Query Builder methods 278–284
entity
adding mapping conditions to 237–239

creating complex types for 243
Entity Framework definition of 215
using stored procedures to manage data 239–240
entity-based collections 254
Entity Client 127
EntityClient Provider 260–265
entity container
Entity Framework definition of 216
instantiated version of 221
ObjectContext class as embodiment of 268
Entity Data Model Designer, ADO.NET
about 218
generating objects using 220
Map Entity To Functions button 274
mapping details panel, working with 236–241
using 230–236
Entity Data Model (EDM)
adding 225
conceptual model in EF and 217
Entity Data Model Wizard 218, 225–229, 325, 372
Entity Framework (EF)
about 8, 213
adding to projects 243
as Object Services layer 267
building models 218–219
designing models 225–240
creating a data service 372–375
refreshing model after making structural changes 243
using Entity Data Model Designer 230–236
using Entity Data Model Wizard 225–229
using Model Browser 240
working with mapping details panel 235–240
generating objects 220–221
instantiating context 221
layers 216–218
LINQ provider. *See* LINQ (providers)
namespaces, grouping of related entities 217
OrderBy clause, as query builder method in 222
provider 370
running framework queries 222
storage model 325
understanding 213–216
Where clause, as query builder method in 222
EntityFunctions class 322, 323
entity keys 215
EntityObject base class 220
EntityObject class 268
EntityObject Generator, ADO.NET 241
entity set, definition of 216
EntitySetRights members 376
Entity SQL language
about 222, 246
building custom collections 254
grouping and aggregating entity data 252–254
query builder method equivalents 281–282
user-defined functions 255
using functions 250–252
using literals 249–251
using operators 250
writing basic queries 246–248
Entity SQL queries 256–264
about 278
converting into command tree 280
using EntityClientProvider 260–265
using ObjectQuery class 256–259
Entity Types, definition of 215
envelope transactions, parameterized update queries and 199, 201
eq operator in REST 380
equal sign (=)
comparing columns to literal values using 63
Equals keyword 299
errors
data range 49–56
row-level 51–52
setting column 51
event handlers, validation occurring in 51
events
ColumnChanged 51
ColumnChanging 51
RowChanged 51
RowChanging 51
RowDeleted 51
RowDeleting 51

TableCleared 51
TableClearing 51
TableNewRow 51
Excel
provider class libraries for 126
Except function 302
exception-based errors 49–51
exception handling blocks 50
EXCEPT keyword 252
Except method 282
ExecuteNonQuery method 138, 260
ExecuteReader method 142, 260
ExecuteScalar method 141
ExecuteSQL method 147, 148
ExecuteSQLReturn method 149
ExecuteXmlReader method 146
EXISTS keyword, in Entity SQL language 250
Exp function 323
exporting, DataSet as file to XML 118
expression columns, using 67–70
expressions, filter elements 63–64
extension methods 280, 290, 302–303
external connections 121–132
connection pooling 129, 132
connection strings
builders 124–126
using 121–126
data providers
connecting to SQL Server via 127–132
understanding 126–127
keys for establishing 123–124
using connection strings 121–125
external data
connecting to 8–14
linking to local data set 169–171
managing 3
moving data into DataSets 173–175
moving data into DataTables 171–173
moving from memory to 175–185
configuring update commands 175–179
generating update commands 180–185
performing update 179–180

F

false keyword 249
Feed And Web Slice Setting dialog box 379
Field extension method 307–308
Field (Of T) method 306
Fill method 171, 172, 174, 186
“fill” operations
moving data from databases into DataSets 169, 171
filter expression elements 63–64
filtering, results with the Where clause 295
Find method
FindRows tasks and 102
locating single row based on primary key 60
FindRows method 101–102
First method 283, 284
floating-point value literals 249
Floor function 251
Foreign Key Columns 229
ForeignKeyConstraint class 81
formats, aggregation 90
FOR UPDATE clause 195
FROM clause, in Entity SQL language 246–248
From clause in LINQ 293, 298
FullName value, modifying 158
FULL OUTER JOIN keyword 248
full outer joins 297
function imports, editing 233–234
functions
Acos 323
Add...
in Entity SQL language 251
in LINQ to Entity 322
aggregate 252–254, 301–302
All 301
Any 301
Ascii 323
Asin 323
AsNonUnicode 322
AsUnicode 322
Atan 323
Atan2 323
Average 301
Avg 89, 252
Bitwise 251
BuildConnection 129
canonical, using 323
CAST 251
Ceiling 251
Char 323
CharIndex 323
Checksum 323
Concat 251

Contains 251
CONVERT 64
Cos 323
Cot 323
Count 90, 301
COUNT and BIGCOUNT 252
CreateDateTime 322
CreateDateTimeOffset 322
CreateTime 322
CurrentTimestamp 323
CurrentUser 323
DataLength 323
DateAdd 323
date and time 251, 322, 323
DateDiff 323
DateName 323
DatePart 323
Degrees 323
DEREF 255
Diff... 251, 322
Difference 323
EndsWith 251
Except 302
Exp 323
Floor 251
GetColumnTable 25
GetConnectionString 146, 159, 183, 198, 200, 274, 276, 318, 319, 340, 341
GetDate 323
GetDesignerTable 34
GetUtcDate 323
HostName 323
IIF 64
IndexOf 251
Intersect 302
IsDate 323
IsDBNull 23, 40, 42, 44, 56
ISNULL 64
IsNumeric 323
Left 251, 322
LEN 64
Length 251
Log 323
Log10 323
LongCount 301
LTrim 251
math 251, 323
Max 90, 252, 301
Min 90, 252, 301
NChar 323
NewGuid 251
PatIndex 323
Pi 323
Power 251
QuoteName 323
Radians 323
Rand 323
REF 255
Replace 251
Replicate 323
Reverse 251
Right 251, 322
Round 251, 322
ROW 255
RTrim 251
SaveFormData 274–275, 276–277
SET 252
Sign 323
SoundCode 323
Space 323
“SqlServer” in Entity SQL language 251
SQRT 322
Square 323
StandardDeviation 323
StartsWith 251
statistical 323
StDev 90, 253
STDEVP 253
string 251, 322, 323
StringConvert 323
Stuff 323
Substring 64, 251
Sum 89, 252, 301
Tan 323
ToLower 251
ToUpper 251
TransferDistributed 206, 207
Trim 64, 251
Truncate 323
Unicode 323
Union 302
UnionAll 302
user-defined 255
user-defined (LINQ) 324–329, 339–342
UserName 323
Var 90, 253, 323
VarP 253, 323

G

Generate Database Wizard 217
ge operator in REST 380
GetBytes method 145
GetChar method 145
GetChars method 145
GetChildRows method 79–81
GetColumnTable function 25
GetConnectionString function 146, 159, 183, 198, 200, 274, 276, 318, 319, 340, 341
GetDate function 323
GetDateTime method 145
GetDateTimeOffset method 145
GetDecimal method 144
GetDesignerTable function 34
GetDouble method 145
GetFloat method 145
GetGuid method 145
GetInt16 method 145
GetInt32 method 145
GetInt64 method 145
GetOrdinal method 144

GetParentRows method 79–81, 80
GetSchemaTable method 146
GetSqlMoney method 145
GetString method 145
GetTimeSpan method 145
GetUtcDate function 323
GET verb, HTTP 377
greater than or equal to sign (>=)
  comparing columns to literal values using 63
  in Where clause (LINQ) 296
greater than sign (>), comparing columns to literal values using 63
Group By clause
  collecting aggregate summaries with 302
GROUP BY clause
  in Entity SQL language 253
GroupBy method 281
GROUP BY (SQL language) in ADO.NET 95
group joins 299
Group keyword 299
gt operator in REST 380
GUID keyword 250

H

HasErrors, DataRow property 52
HasErrors, DataTable property 52
HasRows property in returning data rows 142
HasVersion method, DataRow 49
HAVING clause, in Entity SQL language 253
HostName function 323
HTTP
  GET verb 377
  inserting new records 382
  issuing DELETE request 383
  requests, as querying method in EF 222
  transport 371

I

IEnumerable(Of T) interface 289, 315
IIF function 64
IIS (Microsoft Internet Information Service) 370
implementing tables 17–21
importing
  database tables into DataTables 189
  DataSet as file to XML 118
indexed views 98–105
IndexOf function 251
inequality sign (<>)
  comparing columns to literal values using 63
inference, local type 290
Initial Catalog key 123
InitializeService method 375
INNER JOIN keyword 248
inner joins 297
IN operator
  in Entity SQL language 250
  using in filter expressions 63
INSERT queries, adding in SqlDataAdapter 184
INSERT statements
  building 274
  generating automatically 180
  returning specified fields 141
  update operations and 169–170, 171, 180
integer literals 249
Integrated Security key 123
integrity of data 191–208
  employing savepoints 202–203
  transactions and concurrency 191–195
  using distributed transactions 204–208
  using local transactions 195–203
IntelliSense in Visual Studio 326, 328
Interact method 281
internal data, managing 3
Internet Explorer 8, accessing XML 379
Intersect function 302
INTERSECT keyword 252
Intersect method 282
invalid data
  preventing 49–56
IQueryable interface 370
IQueryable(Of T) interface 283, 289, 315
IsDate function 323
IsDBNull function 23, 40, 42, 44, 56
IS NOT operator 250
IsNullable property 157
ISNULL function 64
IsNumeric function 323
Isolation rule 193
IS operator 250
Item properties, in defining row values 38
“it” keyword 280
IUpdatable interface 370

J

JavaScript Object Notation (JSON) 370, 379
joined-table queries and SQLCommandBuilder 181
Join keyword 247, 297
JSON (JavaScript Object Notation) 370, 379, 382

K

key ADO.NET elements 6–7
keyboard shortcuts, adding new columns to DataTables 30, 35
keys for server connections 123–124

L

lambda expressions 290, 296
Left function 251, 322
LEFT OUTER JOIN keyword 248
left outer joins 297, 299
LEN function 64
Length function 251
le operator in REST 380
less than or equal to sign (<=)
  comparing columns to literal values using 63
less than sign (<)
  comparing columns to literal values using 63
  in Where clause (LINQ) 296
LIKE operator, matching string patterns using 63
limiting connection time to SQL Server 132
LIMIT keyword 253
linking
  DataTable objects in relationships 88
  results with Join keywords 297–298
  tables 76–78
LINQ
  about 8
  as querying method in EF 222
  data types of columns 307
  enhancements to .Net language 289–291
LINQ (providers)
  to DataSet 291, 305–312
  to Entities
    about 291, 315–316
    downside to 329
    working with custom database functions 324–329
    working with database functions 321–324
    writing queries 316–321
  to Objects
    about 291
    filtering results with Where clause 295
    limiting data returned 299–300
    projecting results with Select clause 293–295
    selecting linked results with Join keyword 297–298
    sorting results with OrderBy clause 296–297
    starting queries with From clause 293
    summarizing data using aggregates 301–302
    using 291–292
  to SQL
    about 291, 331
    building models 337–339
    components of 333–335
    using custom database functions 339
  to XML 291
LINQ queries
  about 278
  converting into command tree 280
  DataTable class in 305
  support of joins 297
  with Objects
    filtering results with Where clause 295
    limiting data returned 299–300
    projecting results with Select clause 293–295
    selecting linked results with Join keyword 297–298
    sorting results with OrderBy clause 296–297
    starting queries with From keyword 293
    summarizing data using aggregates 301–302
ListBox control 347
literals 64, 249–251, 291
local transactions, using 195–203, 209
local type inference 290
locking records 193–194
Log10 function 323
Log function 323
logical implementation of tables 17–18
logical implementation of underlying data source 213

logical models. *See* storage models (storage layers)
logical operators 250, 296, 380
LongCount function 301
lookups, performing case-sensitive 67
lt operator in REST 380
LTrim function 251

M

Many-to-Many table relationships 76–77, 83–84
Map Entity To Functions button 274
Map Entity toolbar buttons 239
Mapping Details panel 226, 235–240, 274
mappings (mapping layers)
  adding condition to entity 237–239
  in Entity Framework 218
  linking storage and conceptual models with 226
  modifying 235–240
  using 219
Mapping Specification Language (MSL) 218, 219, 226, 235
MappingType enumerated values 114
math functions 251, 323
math operators 250, 380
Max function 90, 252, 301
MaxLength, DataColumn class property 24
Max method 283
Me keyword 280
MERGE verb 382
metadata key-value pairs 226
methods
AcceptChanges 48, 57, 99
AddObject 272
AddWithValue 157
AsEnumerable extension 306
BeginExecuteNonQuery 139
BeginExecuteXmlReader 146
BeginTransaction 196
Cancel 140
Commit 196–197
Complete 205–206
Compute
  calculating aggregate of single table columns 91
  Parent and Child prefixes and 94
Count 283
CreateObjectSet 222
DataTable.Rows.Find 71
Delete 101
DeleteObject 273
DelimSubstring 278–279
Dispose 128
Distinct 281
EndExecuteNonQuery 139
EndExecuteXmlReader 146
EnlistTransaction 205
Except 282
ExecuteNonQuery 138, 260
ExecuteReader 142, 260
ExecuteScalar 141–142
ExecuteSQL 147, 148
ExecuteSQLReturn 147, 149
ExecuteXmlReader 146
extension 280, 290, 302–303
Field extension 307–308
Field (Of T) 306
Field(Of T) 306
Fill 171, 172, 174, 186
Find 102
FindRows 101
First 283, 284
GetBoolean method 145
GetByte method 145
GetBytes 145
GetChar 145
GetChars 145
GetChildRows 79–81
GetDateTime 145
GetDateTimeOffset 145
GetDecimal method 144
GetDouble 145
GetFloat 145
GetGuid 145
GetInt16 145
GetInt32 145
GetInt64 145
GetOrdinal 144
GetParentRow 80
GetParrentRow 79–81
GetSchemaTable 146
GetSqlMoney 145
GetString 145
GetTimeSpan 145
GroupBy 281
InitializeService 375
Interact 281
Intersect 282
Max 283
OfType 281
Open 196
OpenReader 147, 149
OrderBy 280, 281, 297
partial 291
query builder 278–284, 281–282
ReadXML 110–111
ReadXmlSchema 110
Refresh 273
RefreshConstraints 85
Rollback 196–197
Save 202–203
Select
  inefficiency of 98–99

sorting rows 100
Select query builder methods 280
SelectValue 281
SetEntitySetAccessRule 376
Skip 281
SqlConnectionStringBuilder 148
SqlDataAdapter.Fill 171, 173–174
SqlDataAdapter.FillSchema 174
ThenBy 297
Top 281
ToTable 103, 105
Union 281, 282
UnionAll 281, 282
Update 180
Where 281, 282, 283, 284
WriteXML 108–110
Microsoft Access, provider class libraries for 126
Microsoft Distributed Transaction Coordinator (MSDTC) 204
Microsoft Excel, provider class libraries for 126
Microsoft Internet Explorer 8, accessing XML 379
Microsoft Internet Information Service (IIS) 370
Microsoft SQL Server. *See* SQL Server
Choosing Data Source dialog box and 11
relationship to ADO.NET 7
sharing terms with ADO.NET 4
Microsoft SQL Server 2005. *See* SQL Server 2005
Microsoft SQL Server 2008 Express Edition
connection keys for 123
naming the server 12
Microsoft Visual Studio. *See* Visual Studio
Min function 90, 252, 301
MissingMappingAction properties 187
MissingSchemaAction properties 188–189
Model Browser 240
modeling terms of relational databases
cardinality 76
models, Entity Framework
building 218–219
definition of 215
modifying
application configuration file settings 13
databases through entity objects 274–277
DataTable names 29
FullName values 158
RowFilter 101
rows in DataTables 43–45
RowStateFilter 101
Sort 101
TableName properties 19
mod operator in REST 380
monitoring errors 50
mouse
selecting properties with 234
.xsd extension for tables created by 28
Move Via a Thumbnail View control in Entity Data Model Designer 229
MSDTC (Microsoft Distributed Transaction Coordinator) distributed transactions 204
MSL (Mapping Specification Language) 218, 219, 226, 235
mul operator in REST 380
MultipleActiveResultSets key 124

N

Name property of the ObjectSet 280
Namespace properties, in DataTable classes 111–113
namespaces
feature in XML 111–113
grouping of related entities 217
SqlClient 127
SqlTypes 145
navigation property, Entity Framework definition of 216
NChar function 323
ne operator in REST 380
Nested property, in child tables 113
new {} (anonymous type definition) 294
NewGuid function 251
NewRow method, generating table-specific data rows 38, 57
nexted results 146
nondata results, returning System.DBNull for 141
None, as EntitySetRights member 376–384
nonqueries, running 137–138
non-Unicode strings 249
Normalization process 21
Nothing values 290
NOT operator 250, 296
not operator in REST 380
nullable types 290

NULL values
  as default value 40
  in aggregate functions 90
  in C# 290
  in Entity SQL language 250
  in SqlDataReader 144
  IsNullable property 157
  ISNULL function and 64

O

ObjectContext
  class 268
  objects 221
object initializers 290
object layer
  generation rules 234
  managing 241
ObjectQuery
  class 256–259
  instance 280
ObjectQuery(Of T) class 268, 283, 315
Object Relational (O/R) Designer 335–339
objects
  DataColumn, adding to DataTable 37
  DataRow
    AcceptChanges method 48
    Delete method 48
    removing 45–46
  DataSet, creating 73–74
  DataTable
    creating 18
    logical implementation of 17–18
    physical implementation of 18
    populating Columns collection of 21–28
  DataTableMapping 186
  in Entity Framework model, generating 220–221
  LINQ to
    about 291
    filtering results with Where clause 295
    limiting data returned 299–300
    projecting results with Select clause 293–295
    selecting linked results with Join keyword 297
    sorting results with OrderBy clause 296–297
    starting queries with From clause 293
    summarizing data using aggregates 301–302
    using 291–292
  ObjectContext 221
  serializing data set and Data Table 107–111
  SqlDataAdapter 171
  TransactionScope 206, 207
  values appearing as 7
Object Services
  about 267–268
  accessing entity data 268–270
  modifying entity data 271–278
  using query builder methods 278–284
ObjectSet(Of TEntity) class 268, 315
object types, generic 39
ODBC
  connection string builders 126
  employing savepoints and 202
  implementation of command-related processing 135
  parameter class and 160
  provider class libraries for 126–127
  supporting nested results 146
OdbcCommand 136
OdbcDataAdapter class 170
ODBC providers 7
OdbcTransaction class 195
OfType method 281
OLE DB
  connection string builders 126
  creating parameterized queries for 167
  employing savepoints and 202
  implementation of command-related processing 135
  parameter class in 154, 160
  provider class libraries for 126–127
  supporting nested results 146
OleDbCommand 136
OleDbConnection 127, 133
OleDbDataAdapter class 170
OLEdbParameter 154
OLE DB providers 7
OleDbTransaction class 195
One-to-Many table relationships 76, 81–82
One-to-One table relationships 76
Open Data Protocol 369–370
Open method 196
OpenReader method 147, 149

operators
  Boolean 63
  comparison 63
  IN 63
  LIKE 63
Optimistic concurrency 194
Oracle
  as target specific platform 7
  provider in prior versions 127
  SELECT statements
    FOR UPDATE clause 195
OrderBy clause
  as query builder method in EF 222
  in applying Skip or Take clauses 300
  in LINQ 296–297
ORDER BY clause 248
OrderByDescending extension method 297
OrderBy extension method 297
OrderBy method 280, 281
O/R (Object Relational) Designer 335–339
OR operator 250, 296
or operator in REST 380
outer joins 297
“out” parameters, creating 167
OUTPUT keywords
  INSERT statements 141
OVERLAPS keyword 252

P

ParameterDirection
  .InputOutput option 161
  options 161
  .Output option 161, 167
  .ReturnValue option 161
Parameter instances, purpose of 7
ParameterName property 157
parameters
  developing queries with 153–161
    implementing standard queries 155–161
    understanding 154–155
    using parameters in stored procedures 161–166, 179
Parameters collection 156, 157
parent-child
  data, generating hierarchal 118
  records, locating 79–81
  relationships
    enforcing cascade deletes 88
    establishing 76–78
parent-column values
  rules for updating or deleting 82
Parent fields
  referencing in expressions 98
parentheses ( )
  in expression evaluation 63
  using in Where clause (LINQ) 296
Parent prefixes 94
partial methods 291
passing
  instance of DataSet 173–174
  SchemaType.Mapped 174
  strings to Add method 74
PatIndex function 323
Pessimistic concurrency 194
physical implementation of DataTable objects 18
Pi function 323
POST verb 382
Power function 251
PrimaryKey property, in DataTable class 24–25, 35
primary keys
  adding 30
  finding rows by 60–62
properties
  creating from scalar properties complex 234–235
  editing 232
  Entity Framework definition of 215
  selecting 234
Properties panel, in DataSet Designer 33
protecting records 193–194
PUT verb 382

Q

queries. *See also* SQL queries; *See also* LINQ queries
  developing parameterized 153–161
    implementing standard queries 155–161
    understanding need for parameters 154–155
    using stored procedures 161–166
Queryable extension methods 283–284
query builder methods 278–284
querying data 59–64
querying databases 135–149
  processing SQL queries 135–140
    asynchronously 139–141
    creating command objects 136–137

processing queries 137–138
returning results 140–149
accessing field values 144–146
processing complicated results 146–151
returning data rows 142–144
returning single values 141–142
running EF queries 222
querying data in Entity Framework 245–264
about 245
running Entity SQL queries 256–264
using an ObjectQuery class 256–259
using EntityClientProvider 260–265
using Entity SQL language 246–255
building custom collections 254
grouping and aggregating entity data 252–254
user-defined functions 255
using functions 250–252
using literals 249–251
using operators 250
writing basic queries 246
QuoteName function 323

R

Radians function 323
Rand function 323
range variable 293
"read locks" 198
ReadMultiple, as EntitySetRights member 376
ReadOnly, DataColumn class property 24
ReadSingle, as EntitySetRights member 376
ReadXML method 110–111
ReadXmlSchema method 110
records, locking and protecting 193–195
Refactor complex types 233
REF function 255
RefreshConstraints method 85
Refresh method 273
relational databases
ADO.NET and 4
definition of concepts 215–216
establishing relationships between 78–79
creating data relations 78–79
locating parent-child records 79–81
types of relationships 76–78
modeling terms
cardinality 76
relationships
establishing, between tables 76–81
understanding table 76–78
relaxed delegates 290
Replace function 251
Replicate function 323
RESTful, as HTTP transport 371
REST (Representational State Transfer)
about 369, 370
accessing data services using 377–384
configuring data rights from 375–376
updating entities with 381–383
using operators in 380
Reverse function 251
Right function 251, 322
RIGHT OUTER JOIN keyword 248
right outer joins 297
rollback changes to rows 51
rollback local transactions 209
Rollback method 196–197
Round function 251, 322
RowChanged event 51
RowChanging event 51
RowDeleted event 51
RowDeleting event 51
RowFilter, modifying 101
ROW function 255
row-level errors 51–52
rows
limiting operations on 101
returning data 142–144
roll back changes 51
searching rows
by primary key 60–62
with search criteria 62–64
tags in DataTable 113
Rows.Add method 40, 41, 42, 57
rows, data
adding process 37–41
creating rows 37–38
defining row values 38–40
storing rows in table 40–41
retrieving sets of 151

Rows.Find method, locating row by primary key 60
RowStateFilter
  modifying 101
  setting properties 100
RowStates 47–48
rowversion columns 195
RowVersions 48–49
RTrim function 251
rules
  defining update and delete 85–87
  for establishing data relationships 78
  for updating or deleting parent-column values 82
rules defining transactions 193–194

S

SaveChanges, calling 273
SaveFormData function 274–275, 276–277
Save method 202–203
savepoints, employing 202–203
scalar properties, creating complex properties from 234–235
schema language, using MSL 226
schemas
  Entity Framework and changes in external 214
  using XML, in Entity Framework model 218
SchemaType.Mapped 174
searching
  criteria 62–64
  sorting results 64–67
Select clause
  in Entity SQL language 246–248
  in LINQ 293–295
SELECT command, adding to SqlDataAdapter 184
SelectCommand property 171
Select method
  entity SQL equivalent of 281
  inefficiency of 98–99
  searching many table columns 62–64
  sorting rows and 100
SELECT queries
  returning data rows 142–144
  returning simple results 141
  using user-defined functions 255
SELECT statements
  Fill operations issuing 169–171
  FOR UPDATE clause (Oracle) 195
  in transactions 198
  of LINQ queries 278
SelectValue method 281
Self-Tracking EntityObject Generator, ADO.NET 241
serializing DataSet and DataTable objects 107–111
Server Explorer
  accessing 32
  Data Connections items in 32–33
  O/R Designer and 336, 337–339
server identity functions 323
service layers 369–371
SetEntitySetAccessRule method 376
SET function 252
setting, RowStateFilter properties 100
Shift keys, selecting properties using 234
Sign function 323
single quotes (' ')
  using BINARY keyword with 250
  using GUID keyword with 250
  using strings with 249
Skip clause 300
SKIP keyword 253
Skip method 281
Solution Explorer
  files added from Connection Wizard 14
  panel 234
sorting
  DataView rows 100
  results with the Order By clause 296–297
  search results 64–67
Sort, modifying 101
SoundCode function 323
source code object layer
  managing 241
  regenerating 233
Space function 323
SqlBulkCopy 127
SqlClient namespace 127
SqlCommandBuilder 180–181, 194, 348
SqlCommand class
  about 127, 136–137, 156
  ExecuteNonQuery method 138, 143
  instances 171
  objects 139–140, 141–142, 156–157
SqlCommand objects 146
SqlConnection 127, 132

SqlConnectionStringBuilder 146, 148, 158, 159, 318, 319, 340, 341
SqlDataAdapter 127
  associating command builder with 181–182
  .Fill method 171, 173–174
  .FillSchema method 174
  importing database tables through 189
  .MissingMappingAction property 187
  .MissingSchemaAction properties 188–189
  objects 171
  .SelectCommand property 175
  support features of 170–171
  syncing data with 181–185
SqlDataAdapter class 172, 188
SqlDataReader class
  about 127
  accessing field values 144–146
  data access methods on 145–146
  DBNull values in 144
  in retrieving multiple record sets 146–150
  scanning data sets with 143
  using with ExecuteNonQuery method 143
SqlDataReader objects, creating 149
SqlDbType property 157
SqlFunctions class 323
SQL language
  GROUP BY clause in ADO.NET 95
  types of statements 136–137
SqlMetal.exe 343
SqlParameter class 157
SQL provider, LINQ to
  about 291, 331
  building models 337–339
  components of 333–335
  Entities provider vs. 332
  using custom database functions 339
SQL queries
  processing 135–140
    asynchronously 139–141
    creating command objects 136–137
    running nonqueries 138–139
  returning results 140–149
    accessing field values 144–146
    processing complicated results 146–151
    returning data rows 142–144
    returning single values 141–142
  risks of building statements with string elements 154
SQL query statements 136
  "@" as placeholders in 155
SQL Server
  ACID requirements and 193
  command builder for 180–181
  connecting via data providers 127–132
  connection strings 122–124, 125
  creating parameterized queries for 167
  databases, communicating with 122–125
  employing savepoints on 202
  Entity Framework support for 215
  Entity Framework vs. 245
  LINQ to Entities support in 332
  ODBC connection strings and 124
  OLE DB connection strings and 124
  processing queries 135–140
    asynchronously 139–141
    creating command objects 136–137
    running nonqueries 137–138
  provider class libraries for 126–127
  provider-specific class names 127
  queries, wrapper for 150
  returning query results 140–149
    accessing field values 144–146
    processing complicated results 146–151
    returning data rows 142–144
    returning single values 141–142
  SqlTransaction class in 195
  support for Common Language Runtime (CLR) 324
  table updates 195
  Visual Studio 2010 Entity Framework support for 228
SQL Server 2005
  INSERT statements in 141
SQL Server 2008 Express Edition
  connection keys for 123
  naming the server 12

SQL Server Authentication 12
"SqlServer" functions in Entity SQL language 251–252
SqlTransaction class 195
SqlTransaction instance 199
SqlTypes namespace 145
SqlValue property 157
SQRT function 322
Square function 323
SquareRoot function 323
SSDL (Store Schema Definition Language) 217, 219, 226, 325
StandardDeviation function 323
StartsWith function 251
StatesByYear class 326
statistical functions 323
STDEV function 90, 253
STDEVP function 253
storage models (storage layers)
  in Entity Framework 217
  linking properties from multiple 237
  linking with conceptual models 226
  modifying 236
  using 219
  using as conditions 237
stored procedures
  about 137
  calling 138
  editing function imports 233–234
  in Entity SQL language 246
  managing entity data using 239–240
  row-producing 142
  table-valued, in EF model 229
  update database-side content from entity changes 243
  using parameters in 161–166
Store Schema Definition Language (SSDL) 217, 219, 226, 325
storing
  data in DataTables 37–57
    adding data process 37–41
    batch processing 46–49
    examining in changing data 42–44
    removing data 45–46
    validating changes 49–56
  rows in tables 40–41
string concatenations 139
StringConvert function 323
string functions 251, 322
strings, Unicode and non-Unicode 249
strongly typed DataSets in .NET Framework 214
Stuff function 323
sub operator in REST 380
subqueries, in Entity SQL language 254
Substring function 64, 251
Sum function 89, 252, 301
synonyms, for Boolean operators in Entity SQL language 250
System.Data
  Constraint classes 81–87
  .DataTable 17, 18
  namespace 3, 21
System.DBNull in C# 40
System.Object instances 144
System.Security library 3

T

TableAdapter instance 352, 360
TableAdapterManager instance 352, 360
TableAttribute 334
TableCleared event 51
TableClearing event 51
TableDetails form 26
TableMapping rules, adjusting 214
TableName properties, modifying 19
TableNewRow event 51
tables. *See* DataTables
tabular data format, manipulating 4
tags for DataTable rows 113
Take clause 300
Tan function 323
testing
  database connections 129, 131
  tabular results of user supplied query 145
  update and delete rules 86
text templates 241
ThenByDescending extension method 297
ThenBy extension method 297
this keyword 280
TIME keyword 249
ToLower function 251
Toolbox, using with DataSet Designer 29, 35
TOP
  clause in Entity SQL language 253
  keyword 300
  method 281
ToTable method 103, 105
ToUpper function 251
transactions
  concurrency and 191–195

definition of 192
distributed, using 204–208
issuing savepoints within 203
using local 195–203, 209
TransactionScope class 204
TransactionScope object 206, 207
TransferDistributed function 206, 207
TransferLocal routine 199, 200
Trim function 64, 251
true keyword 249
Truncate function 323
truncate, using time and date functions 251
TrustServerCertificate and Encrypt keys 124
T-SQL query language 246, 253, 254–255

U

Unicode function 323
Unicode strings 249
UnionAll function 302
UnionAll method 281, 282
Union function 302
UNION keyword 252
Union method 281, 282
UniqueConstraint class 81–84
Unique Constraint dialog box 30–31
Unique, DataColumn class property 24
UnitEditor_Load event handler 181, 184
Update method 180–182
Update Model From Database shortcut command 233
UPDATE queries
adding in SqlDataAdapter 184
parameterized 199, 201
UpdateRule properties, setting 82
UPDATE statements
building 274
generating automatically 180
update operations and 169–170, 171, 180
UpdateStatus.SkipCurrentRow 180
user configuration files, modifying settings in 13
user-defined
data type for entities 215
functions 255
functions (LINQ) 324–329, 339–342
UserName function 323

V

validation-based errors 51–56
VALUE keyword 248
Value property 157
Var function 90, 253, 323
VarP function 253, 323
“VCR” controls 353
views, indexed 98–105
Visual Basic
accessing data through model-generated objects 268–270
accessing field values 144
adding aggregate columns 95
adding BindingNavigator control to Windows forms 353
adding columns to DataTables 26–27
adding database tables to DataSet Designer 32–33
adding DelimSubstring method to System.String class 279
adding expression columns 70
adding new entities through objects 271–272
adding relationships between two tables 79
adding rows to DataTables 41
adding tables to DataSets 75
building connection strings 125
ByRef parameter 255
calling BeginExecuteNonQuery method 140
calling Complete method of TransactionScope 205–206
calling DeleteObject method 273
calling stored procedures with parameters 165–166
computing aggregate values 91, 93–94
configuring update commands 175–177
creating custom tables in DataSet Designer 28–32
creating data-pound WPF applications 355–361
creating data service from EF models 373–375
creating DataTables 20–21
creating DataViews 99–101
creating instances of TransactionScope 204
creating new instance of command builder 180
creating SQL Server connections 128–129
defining row values 38–40

defining update and delete rules in Data Relations 86–87
employing savepoints 203
finding rows by primary key 61–62
generating DataView tables from DataView 104–105
generating EF objects 221
generating summaries 97
generating XML from DataSet 116–117
GetOrdinal method in 144
LINQ- specific language features 8
locating parent-child records 81
Me keyword in 280
modifying databases through entity objects 276–277
modifying existing properties 271
modifying rows in DataTable 44
moving data into DataSets 173–174
moving Data into DataTables 172–174
nesting child tables 113
Nothing values 290
opening SQL Server connections 130–131
processing database queries in 148–149
processing local transactions 200–202
processing with distributed transactions 207–208
ReadXml in 110
referencing parent fields in expressions 98
retrieving entity data through ObjectQuery 258–259
retrieving entity data through provider 263–265
running nonqueries 138
selecting and sorting DataRow objects 66–67
syncing data with SqlDataAdapter 183–185
transactions on open database connections 196
updating data in with parameters 159–160
updating DataTables 179
using batch processing 47
using BeginEdit 51
using ColumnMapping property 114
using Commit and Rollback methods 197–198
using CreateObjectSet method 222
using DataRow class types 38
using DataRow.HasVersion method 49
using Dataset Designer with 27
using DataTableMapping objects 186–187
using ExecuteReader method to generate data reader 260
using LINQ in
  applying set operations 303
  calling custom database functions 327–328
  creating implicit joins 306–307
  filtering results with Where clause 296
  limiting data returned 299–300
  projecting results with Select clause 294–295
  querying LINQ to SQL 341–342
  querying the data set 310–312
  querying to Entities 319–321
  selecting linked results with Join keyword 298–299
  sorting results with OrderBy clause 297
  starting queries with From clause 293
  summarizing data using aggregates 301–302
  to SQL provider structure 334–336
  using database functions 323
  using EntityFunctions class 323
  Where clause not supported 321–322
using namespace elements 19
using namespace properties in 112–113
using query builder methods 284
using Rollback method 203
validating data 51–55
viewing source code 233

WriteXML method in 109–110
XML literals functionality in 291
visual designer 332. *See also* Object Relational (O/R) Designer
Visual Studio
DataAdapterManager in 352
DataSet Designer and 28
design tool file extensions 217
editor 235
edmgen.exe in 220, 223
GetSchemaTable method 146
in ASP.NET and 362
IntelliSense in 326, 328
managing code generation process 241
Mapping Details panel 226
Model Browser of 240
providers for 126
queryable methods 283
regenerating Visual Basic or C# source code object layer 233
support for building WPF/XAML applications 354
Windows SDK in 336
Visual Studio 2005, strongly typed DataSets in 214
Visual Studio 2008
ADO.NET Data Services in 369
LINQ to SQL provider in 332
.NET Framework and 213
Visual Studio 2010, creating projects using 8–9
Visual Studio 2010 Entity Framework 228
Visual Studio Express Edition
ASP.NET and 362
Database Explorer in 336, 337–339
Visual Studio IDE, menu choices hidden in 14
Visual Studio Properties panel, selecting associations with 232
Visual Studio Toolbox 336

W

WCF (Windows Communication Foundation) Data Services
about 369–370
as querying method in EF 222
as RESTful system 371
defining service rights 375–377
setting up 371–376
web browser content 362
WHEN clause, in Entity SQL language 252
WHERE clause
as query builder method in EF 222
comparison operators within 250
Where clause, in LINQ 295, 306–307, 321–322
Where method 281, 282, 283, 284
Windows Communication Foundation (WCF) Data Services
about 369–370
as querying method in EF 222
as RESTful system 371
defining service rights 375–377
setting up 371–376
Windows forms, binding data in 347–354
creating complex data-bound forms 348–350
creating simple-bound forms 351–352
understanding 352–354
Windows Presentation Foundation (WPF), binding data in 354–362
Windows SDK 336
With clause 292
WITH (NOLOCK) hint 198
Wizards
ADO.NET Entity Data Model 371
Connection
creating a data source using 8–14
Data Source Configuration 355, 360
forgoing 13
Data Source Connection 27, 122–124, 126
Entity Data Model 218, 225–229, 325, 372
Generate Database 217
WPF (Windows Presentation Foundation), binding data in 354–362
wrappers for SQL Server queries
SqlClient.SqlCommand class 150
Write..., as EntitySetRights member 376
WriteXML method 108–110, 113
writing XML 108–110

X

XAML language 354, 360–361
XML (Extensible Markup Language)
about 107

accessing Framework modeling layers 226
accessing through Internet Explorer 8 379
axis properties 291
building WPF/XAML applications 354, 360–361
columns, managing and positioning 113–117
documents 370
embedded expressions 291
exporting DataSet file as 118
generation of 111–117
  storing connection strings 126
identifying namespaces 111–113
LINQ to 291
literals 291
nesting child tables 113
processing return data from queries in 146
reading 110–111
schema dialects in Entity Framework 216, 217
writing 108–110
xmlns attributes 112
XmlReadMode enumerated values 110–111
XmlWriteMode arguments 109–110

Z

zoom controls in Entity Data Model Designer 229

About the Author

Tim Patrick is an author and software architect with over 25 years of experience in software development and technical writing. He has written seven books and several articles on programming and other topics. In 2007, Microsoft awarded him with its *Most Valuable Professional* (MVP) award in recognition of the benefits his writings bring to Visual Basic and .NET programmers. Tim earned his undergraduate degree in Computer Science from Seattle Pacific University.