

Practical Database Programming

With Visual C#.NET

Ying Bai



 WILEY

 **IEEE**
IEEE PRESS

 **ftp**
NOW AVAILABLE

**Practical Database
Programming With Visual
C#.NET**

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

IEEE Press Editorial Board
Lajos Hanzo, *Editor in Chief*

R. Abari	T. Chen	B. M. Hammerli
J. Anderson	T. G. Croda	O. Malik
S. Basu	M. El-Hawary	S. Nahavandi
A. Chatterjee	S. Farshchi	W. Reeve

Kenneth Moore, *Director of IEEE Book and Information Services (BIS)*

Practical Database Programming With Visual C#.NET

Ying Bai

*Department of Computer Science and Engineering
Johnson C. Smith University
Charlotte, North Carolina*

 **IEEE**
IEEE PRESS

 **WILEY**

A John Wiley & Sons, Inc., Publication

Copyright © 2010 by the Institute of Electrical and Electronics Engineers, Inc.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey. All rights reserved.
Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data:

Bai, Ying, 1956–

Practical database programming with Visual C#.NET / Ying Bai.

p. cm.

Includes index.

ISBN 978-0-470-46727-5 (cloth)

1. Microsoft Visual C# .NET. 2. C# (Computer program language) 3. Database design. 4. Microsoft .NET. I. Title.

QA76.73.C154B347 2009

006.7'882–dc22

2009025977

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To my wife, Yan Wang, and my daughter, Xue Bai

Contents

Preface **xxiii**

Acknowledgment **xxv**

1 Introduction **1**

Outstanding Features of the Book	2
Target Audience	2
Topics Covered	3
Organization of the Book and How to Use It	5
How to Use the Source Code and Sample Databases	6
Instructor and Customer Support	7
Homework Solutions	8

2 Introduction to Databases **11**

2.1	What Are Databases and Database Programs?	12
2.1.1	File Processing System	12
2.1.2	Integrated Databases	13
2.2	Develop a Database	14
2.3	Sample Database	15
2.3.1	Relational Data Model	18
2.3.2	Entity-Relationship Model (ER)	18
2.4	Identifying Keys	19
2.4.1	Primary Key and Entity Integrity	19
2.4.2	Candidate Key	19
2.4.3	Foreign Keys and Referential Integrity	19
2.5	Define Relationships	20
2.5.1	Connectivity	20
2.6	ER Notation	23
2.7	Data Normalization	24
2.7.1	First Normal Form (1NF)	25
2.7.2	Second Normal Form (2NF)	25
2.7.3	Third Normal Form (3NF)	26
2.8	Database Components in Some Popular Databases	28
2.8.1	Microsoft Access Databases	28

viii Contents

2.8.1.1	Database File	29
2.8.1.2	Tables	29
2.8.1.3	Queries	29
2.8.2	SQL Server Databases	29
2.8.2.1	Data Files	30
2.8.2.2	Tables	30
2.8.2.3	Views	31
2.8.2.4	Stored Procedures	31
2.8.2.5	Keys and Relationships	31
2.8.2.6	Indexes	32
2.8.2.7	Transaction Log Files	32
2.8.3	Oracle Databases	32
2.8.3.1	Data Files	33
2.8.3.2	Tables	33
2.8.3.3	Views	33
2.8.3.4	Stored Procedures	33
2.8.3.5	Indexes	34
2.8.3.6	Initialization Parameter Files	35
2.8.3.7	Control Files	35
2.8.3.8	Redo Log Files	35
2.8.3.9	Password Files	35
2.9	Create Microsoft Access Sample Database	36
2.9.1	Create LogIn Table	36
2.9.2	Create Faculty Table	38
2.9.3	Create Other Tables	40
2.9.4	Create Relationships Among Tables	42
2.10	Create Microsoft SQL Server 2005 Sample Database	46
2.10.1	Create LogIn Table	48
2.10.2	Create Faculty Table	50
2.10.3	Create Other Tables	51
2.10.4	Create Relationships Among Tables	56
2.10.4.1	Create Relationship Between LogIn and Faculty Tables	56
2.10.4.2	Create Relationship Between LogIn and Student Tables	59
2.10.4.3	Create Relationship Between Faculty and Course Tables	59
2.10.4.4	Create Relationship Between Student and StudentCourse Tables	60
2.10.4.5	Create Relationship Between Course and StudentCourse Tables	62
2.11	Create Oracle 10g XE Sample Database	63
2.11.1	Create Oracle User Database	64
2.11.2	Add New Data Tables into Oracle User Database	65
2.11.2.1	Create LogIn Table	66
2.11.2.2	Create Faculty Table	72
2.11.2.3	Create Other Tables	75
2.11.3	Create Constraints Between Tables	78
2.11.3.1	Create Constraints Between LogIn and Faculty Tables	80
2.11.3.2	Create Constraints Between LogIn and Student Tables	81

2.11.3.3	Create Constraints Between Course and Faculty Tables	83
2.11.3.4	Create Constraints Between StudentCourse and Student Tables	83
2.11.3.5	Create Constraints Between StudentCourse and Course Tables	84
2.12	Chapter Summary	86
	Homework	87

3 Introduction to ADO.NET

91

3.1	ADO and ADO.NET	91
3.2	Overview of ADO.NET 2.0	92
3.3	Architecture of ADO.NET 2.0	93
3.4	Components of ADO.NET 2.0	95
3.4.1	Data Provider	95
3.4.1.1	ODBC Data Provider	96
3.4.1.2	OleDb Data Provider	97
3.4.1.3	SQL Server Data Provider	97
3.4.1.4	Oracle Data Provider	98
3.4.2	Connection Class	98
3.4.2.1	Open() Method of Connection Class	100
3.4.2.2	Close() Method of Connection Class	101
3.4.2.3	Dispose() Method of Connection Class	102
3.4.3	Command and Parameter Classes	102
3.4.3.1	Properties of Command Class	103
3.4.3.2	Constructors and Properties of Parameter Class	103
3.4.3.3	Parameter Mapping	104
3.4.3.4	Methods of ParameterCollection Class	106
3.4.3.5	Constructor of Command Class	107
3.4.3.6	Methods of Command Class	108
3.4.4	DataAdapter Class	111
3.4.4.1	Constructor of DataAdapter Class	111
3.4.4.2	Properties of DataAdapter Class	111
3.4.4.3	Methods of DataAdapter Class	112
3.4.4.4	Events of DataAdapter Class	113
3.4.5	DataReader Class	114
3.4.6	DataSet Component	117
3.4.6.1	DataSet Constructor	118
3.4.6.2	DataSet Properties	119
3.4.6.3	DataSet Methods	119
3.4.6.4	DataSet Events	119
3.4.7	DataTable Component	122
3.4.7.1	DataTable Constructor	123
3.4.7.2	DataTable Properties	124
3.4.7.3	DataTable Methods	124
3.4.7.4	DataTable Events	125
3.4.8	ADO.NET 3.5 Entity Framework	127

x Contents

3.4.8.1	ADO.NET 3.5 Entity Data Model	129
3.4.8.2	Using ADO.NET 3.5 Entity Data Model Wizard	132
3.5	Chapter Summary	142
	Homework	144

4 Introduction to Language-Integrated Query (LINQ)

147

4.1	Overview of Language-Integrated Query	147
4.1.1	Some Special Interfaces Used in LINQ	148
4.1.1.1	IEnumerable and IEnumerable<T> Interfaces	148
4.1.1.2	IQueryable and IQueryable<T> Interfaces	149
4.1.2	Standard Query Operators	150
4.1.3	Deferred Standard Query Operators	152
4.1.4	Nondeferred Standard Query Operators	156
4.2	Introduction to LINQ Query	158
4.3	Architecture and Components of LINQ	161
4.3.1	Overview of LINQ to Objects	162
4.3.2	Overview of LINQ to DataSet	163
4.3.3	Overview of LINQ to SQL	163
4.3.4	Overview of LINQ to Entities	164
4.3.5	Overview of LINQ to XML	165
4.4	LINQ to Objects	165
4.4.1	LINQ and ArrayList	166
4.4.2	LINQ and Strings	167
4.4.2.1	Query a String to Determine Number of Numeric Digits	168
4.4.2.2	Sort Lines of Structured Text by Any Field in Line	169
4.4.3	LINQ and File Directories	171
4.4.3.1	Query Contents of Files in a Folder	172
4.4.4	LINQ and Reflection	175
4.5	LINQ to DataSet	176
4.5.1	Operations to DataSet Objects	177
4.5.1.1	Query Expression Syntax	178
4.5.1.2	Method-Based Query Syntax	179
4.5.1.3	Query the Single Table	182
4.5.1.4	Query the Cross Tables	183
4.5.1.5	Query Typed DataSet	186
4.5.2	Operations to DataRow Objects Using Extension Methods	189
4.5.3	Operations to DataTable Objects	193
4.6	LINQ to SQL	194
4.6.1	LINQ to SQL Entity Classes and DataContext Class	195
4.6.2	LINQ to SQL Database Operations	199
4.6.2.1	Data Selection Query	201
4.6.2.2	Data Insertion Query	201
4.6.2.3	Data Updating Query	202
4.6.2.4	Data Deletion Query	204
4.6.3	LINQ to SQL Implementations	206

4.7	LINQ to Entities	206
4.7.1	Object Services Component	207
4.7.2	ObjectContext Component	207
4.7.3	ObjectQuery Component	208
4.7.4	LINQ to Entities Flow of Execution	208
4.7.5	Implementation of LINQ to Entities	210
4.8	LINQ to XML	211
4.8.1	LINQ to XML Class Hierarchy	211
4.8.2	Manipulate XML Elements	212
4.8.2.1	Creating XML from Scratch	212
4.8.2.2	Insert XML	214
4.8.2.3	Update XML	215
4.8.2.4	Delete XML	216
4.8.3	Manipulate XML Attributes	216
4.8.3.1	Add XML Attributes	217
4.8.3.2	Get XML Attributes	217
4.8.3.3	Delete XML Attributes	217
4.8.4	Query XML with LINQ to XML	218
4.8.4.1	Standard Query Operators and XML	219
4.8.4.2	XML Query Extensions	219
4.8.4.3	Using Query Expressions with XML	220
4.8.4.4	Using XPath and XSLT with LINQ to XML	221
4.8.4.5	Mixing XML and Other Data Models	221
4.9	C# 3.0 Language Enhancement for LINQ	222
4.9.1	Lambda Expressions	223
4.9.2	Extension Methods	225
4.9.3	Implicitly Typed Local Variables	226
4.9.4	Query Expressions	227
4.10	Chapter Summary	228
	Homework	230

5 Data Selection Query with Visual C#.NET

235

PART I Data Query with Visual Studio Design Tools and Wizards 236

5.1	Completed Sample Database Application Example	236
5.2	Visual Studio.NET 2008 Design Tools and Wizards	239
5.2.1	Data Design Tools in Toolbox Window	239
5.2.1.1	DataSet	240
5.2.1.2	DataGridView	241
5.2.1.3	BindingSource	242
5.2.1.4	BindingNavigator	242
5.2.1.5	TableAdapter	243
5.2.2	Data Design Wizards in Data Source Window	243
5.2.2.1	Add New Data Source	244
5.2.2.2	Data Source Configuration Wizard	244
5.2.2.3	DataSet Designer	249

- 5.3 Build a Sample Database Project—SelectWizard with SQL Server Database 251
 - 5.3.1 Application User Interfaces 251
 - 5.3.1.1 LogIn Form 252
 - 5.3.1.2 Selection Form 255
 - 5.3.1.3 Faculty Form 256
 - 5.3.1.4 Course Form 257
 - 5.3.1.5 Student Form 257
- 5.4 Add and Utilize Visual Studio.NET Wizards and Design Tools 259
 - 5.4.1 Add and Configure a New Data Source 259
- 5.5 Query and Display Data Using the DataGridView Control 263
 - 5.5.1 View Entire Table 264
 - 5.5.2 View Each Record or Specified Columns 266
- 5.6 Use DataSet Designer to Edit the Structure of DataSet 268
- 5.7 Bind Data to Associated Controls in LogIn Form 271
- 5.8 Develop Codes to Query Data Using Fill() Method 274
- 5.9 Use Return a Single Value to Query Data for LogIn Form 276
- 5.10 Coding for Selection Form 280
- 5.11 Bind Data to Associated Controls in Faculty Form 282
- 5.12 Develop Codes to Query Data from Faculty Table 284
 - 5.12.1 Develop Codes to Query Data Using SQL SELECT Method 285
 - 5.12.2 Develop Codes to Query Data Using LINQ Method 287
- 5.13 Display Pictures for Faculty Form 288
- 5.14 Binding Data to Associated Controls in Course Form 291
- 5.15 Develop Codes to Query Data for Course Form 295
 - 5.15.1 Query Data from the Course Table Using TableAdapter Method 295
 - 5.15.2 Query Data from the Course Table Using LINQ Method 297
- 5.16 Build a Sample Database Project—SelectWizardOracle with Oracle Database 299
 - 5.16.1 Create a New Visual C# Project—SelectWizardOracle 299
 - 5.16.2 Select and Add Oracle Database 10g XE as Data Source 300

PART II Data Query with Runtime Objects 303

- 5.17 Introduction to Runtime Objects 304
 - 5.17.1 Procedure of Building a Data-Driven Application Using Runtime Objects 306
- 5.18 Query Data Using Runtime Objects to Microsoft Access 2007 Database 307
 - 5.18.1 Query Data Using Runtime Objects for LogIn Form 307
 - 5.18.1.1 Declare Runtime Objects 307
 - 5.18.1.2 Connect to Data Source with Runtime Objects 308
 - 5.18.1.3 Coding for Method 1: Using DataSet–DataAdapter to Query Data 310
 - 5.18.1.4 Coding for Method 2: Using DataReader to Query Data 312

5.18.2	Coding for Selection Form	314	
5.18.3	Query Data Using Runtime Objects for Faculty Form	316	
5.18.4	Query Data Using Runtime Objects for Course Form	324	
5.18.5	Query Data Using Runtime Objects for Student Form	335	
5.18.5.1	Coding for Constructor of Student Form	335	
5.18.5.2	Coding for Student Select Button Click Method	336	
5.19	Query Data Using Runtime Objects to SQL Server Database	344	
5.19.1	Migrating from Access to SQL Server and Oracle Databases	345	
5.19.2	Query Data Using General Runtime Objects	348	
5.19.2.1	Query Data Using the General Runtime Objects for LogIn Form	349	
5.19.2.2	Coding for Selection Form	355	
5.19.2.3	Query Data Using General Runtime Objects for Faculty Form	355	
5.19.2.4	Query Data Using General Runtime Objects for Course Form	358	
5.19.2.5	Retrieve Data from Multiple Tables Using Joined Tables Method	360	
5.19.2.6	Query Data Using General Runtime Objects for Student Form	364	
5.19.2.7	Query Data Using Stored Procedures	366	
5.19.3	Query Data Using LINQ to SQL Technique	386	
5.19.3.1	Create Entity Classes and Connect DataContext to Database	388	
5.19.3.2	Query Data Using LINQ to SQL for LogIn Form	388	
5.19.3.3	Coding for Selection Form	391	
5.19.3.4	Query Data Using LINQ to SQL for Faculty Form	393	
5.19.3.5	Query Data Using Joined LINQ to SQL for Course Form	397	
5.19.3.6	Query Data Using LINQ to SQL Stored Procedures for Student Form	401	
5.20	Query Data Using Runtime Objects to Oracle Database	405	
5.20.1	Oracle Database 10g Express Edition Release 2	405	
5.20.2	Configure Oracle Database Connection String	406	
5.20.3	Query Data Using General Runtime Objects	407	
5.20.3.1	Query Data Using General Runtime Objects for LogIn Form	408	
5.20.3.2	Coding for Selection Form	413	
5.20.3.3	Query Data Using Runtime Objects for Faculty Form	414	
5.20.3.4	Query Data Using Runtime Objects for Course Form	417	
5.20.3.5	Stored Procedures in Oracle Database Environment	419	
5.20.3.6	Syntax of Creating a Stored Procedure in Oracle	420	
5.20.3.7	Syntax of Creating a Package in Oracle	420	
5.20.3.8	Create Faculty_Course Package for Course Form	422	
5.20.3.9	Query Data Using Oracle Package for Course Form	426	
5.21	Chapter Summary	432	
	Homework	434	

PART I	Data Inserting with Visual Studio.NET Design Tools and Wizards	440
6.1	Insert New Data into a Database	440
6.1.1	Insert New Records into a Database Using TableAdapter.Insert Method	441
6.1.2	Insert New Records into a Database Using TableAdapter.Update Method	442
6.2	Insert Data into Microsoft Access Database Using Sample Project InsertWizard	442
6.2.1	Create New Project Based on SampleWizards Project	443
6.2.2	Application User Interfaces	443
6.2.3	Create Insert Faculty Form Window	443
6.2.4	Duplicate Visual C#.NET Projects with Installed DataSet	445
6.2.5	Validate Data Before Data Insertion	446
6.2.5.1	The .NET Framework Collection Classes	447
6.2.5.2	Validate Data Using Generic Collection	447
6.2.6	Initialization and Termination Coding for Data Insertion	451
6.2.7	Build Insert Query	453
6.2.7.1	Configure TableAdapter and Build Data Inserting Query	453
6.2.8	Develop Codes to Insert Data Using TableAdapter.Insert Method	455
6.2.9	Develop Codes to Insert Data Using TableAdapter.Update Method	457
6.2.10	Validate Data After Data Insertion	462
6.2.10.1	Modifications to Faculty Form Window	462
6.2.10.2	Modifications to Insert Faculty Form Window	465
6.3	Insert Data into SQL Server Database Using Sample Project SQLInsertWizard	468
6.3.1	Modify Existing Project to Get New Data Insertion Project	469
6.3.2	Create New Form Window to Insert Data for Course Form	470
6.3.3	Trigger and Connect to Visual Studio Design Tools	472
6.3.4	Project Initialization and Validate Data Before Data Insertion	473
6.3.5	Configure TableAdapter and Build Data Insertion Query	476
6.3.6	Develop Codes to Insert Data Using TableAdapter.Insert Method	477
6.3.7	Develop Codes to Insert Data Using TableAdapter.Update Method	481
6.3.8	Use Select Button in Course Form to Perform Data Validation	484
6.3.9	Insert Data into Database Using Stored Procedures	485
6.3.9.1	Create Stored Procedure Using TableAdapter Query Configuration Wizard	485

- 6.3.9.2 Modify Codes to Perform Data Insertion Using Stored Procedure 487
- 6.4 Insert Data into Oracle Database Using Sample Project OracleInsertWizard 489

PART II Data Insertion with Runtime Objects 489

- 6.5 General Runtime Objects Method 490
- 6.6 Insert Data into SQL Server Database Using Runtime Objects Method 491
 - 6.6.1 Add Inserting Data Form Window: Insert Faculty Form 491
 - 6.6.2 Modify Codes to Copied Project 493
 - 6.6.3 Startup Coding and Data Validation Before Data Insertion 494
 - 6.6.4 Insert Data into Faculty Table 498
 - 6.6.5 Validate Data After Data Insertion 502
 - 6.6.5.1 Modifications to Faculty Form Window 503
 - 6.6.5.2 Insert New Faculty Photo 505
 - 6.6.5.3 Modifications to Insert Faculty Form Window 506
- 6.7 Insert Data into Microsoft Access Database Using Runtime Objects 510
 - 6.7.1 Modifications to Namespaces 510
 - 6.7.2 Remove SP Form and Student Form 511
 - 6.7.3 Modify Database Connection String 512
 - 6.7.4 Modify LogIn Query Strings 514
 - 6.7.5 Modify Faculty Query String 515
 - 6.7.6 Modifications to Other Forms 515
- 6.8 Insert Data into Oracle Database Using Runtime Objects 518
 - 6.8.1 Add Oracle Reference and Oracle Namespace 519
 - 6.8.2 Modify Project Namespaces 520
 - 6.8.3 Modify Database Connection String 521
 - 6.8.4 Modify LogIn Query Strings 522
 - 6.8.5 Modify Faculty Query String 523
 - 6.8.6 Modifications to Other Forms 525
- 6.9 Insert Data into Database Using LINQ Queries 526
 - 6.9.1 Insert Data into SQL Server Database Using LINQ to SQL Queries 527
- 6.10 Insert Data into Database Using Stored Procedures 527
 - 6.10.1 Insert Data into SQL Server Database Using Stored Procedures 527
 - 6.10.1.1 Add an Inserting Data Form Window: Insert Course Form 528
 - 6.10.1.2 Develop Stored Procedures of SQL Server Database 529
 - 6.10.1.3 Develop Codes to Call Stored Procedures to Insert Data into Course Table 532
 - 6.10.2 Insert Data into Oracle Database Using Stored Procedures 538
 - 6.10.2.1 Develop Stored Procedures in Oracle Database 539
 - 6.10.2.2 Develop Codes to Call Stored Procedures to Insert Data into Course Table 543

6.11 Chapter Summary 547
Homework 548

7 Data Updating and Deleting with Visual C#.NET

PART I Data Updating and Deleting with Visual Studio.NET Design Tools and Wizards 552

7.1 Update or Delete Data in Databases 553
7.1.1 Updating and Deleting Data in Related Tables in DataSet 553
7.1.2 Update or Delete Data in Database Using TableAdapter DBDirect Methods—TableAdapter.Update and TableAdapter.Delete 554
7.1.3 Update or Delete Data in Database Using TableAdapter.Update Method 554
7.2 Update and Delete Data for Microsoft Access Database 555
7.2.1 Create New Project Based on InsertWizard Project 556
7.2.2 Application User Interfaces 556
7.2.2.1 Modify Faculty Form Window 557
7.2.2.2 Bind Data for All Textboxes of Faculty Form Window 557
7.2.3 Validate Data Before Data Updating and Deleting 558
7.2.4 Build Update and Delete Queries 558
7.2.4.1 Build Data Updating Query Function 558
7.2.4.2 Build Data Deleting Query Function 559
7.2.5 Develop Codes to Update Data Using TableAdapter DBDirect Method 560
7.2.5.1 Modifications of Coding 561
7.2.5.2 Updating Coding 561
7.2.6 Develop Codes to Update Data Using TableAdapter.Update Method 562
7.2.7 Develop Codes to Delete Data Using TableAdapter DBDirect Method 564
7.2.8 Develop Codes to Delete Data Using TableAdapter.Update Method 565
7.2.9 Validate Data After Data Updating and Deleting 566
7.3 Update and Delete Data for SQL Server Database 568
7.4 Update and Delete Data for Oracle Database 572

PART II Data Updating and Deleting with Runtime Objects 572

7.5 Runtime Objects Method 572
7.6 Update and Delete Data for SQL Server Database Using Runtime Objects 574
7.6.1 Update Data in Faculty Table for SQL Server Database 574
7.6.1.1 Modify Faculty Form Window 575
7.6.1.2 Modify Original Coding in Faculty Form 576
7.6.1.3 Develop Codes to Update Data 577
7.6.1.4 Validate Data Updating 579

7.6.2	Delete Data from Faculty Table for SQL Server Database	579
7.6.2.1	Develop Codes to Delete Data	579
7.6.2.2	Validate Data Updating and Deleting	581
7.7	Update and Delete Data for Oracle Databases Using Runtime Objects	584
7.7.1	Add Oracle Namespace Reference	585
7.7.2	Modify Connection String and Query String for LogIn Form	585
7.7.2.1	Modify Connection String in Constructor of LogIn Class	585
7.7.2.2	Modify SELECT Query String in TabLogIn Button Click Method	585
7.7.2.3	Modify SELECT Query String in ReadLogIn Button Click Method	586
7.7.3	Modify Query Strings in Faculty Form	586
7.7.3.1	Modify SELECT Query String for Select Button Click Method	586
7.7.3.2	Modify UPDATE Query String for Update Button Click Method	586
7.7.3.3	Modify DELETE Query String for Delete Button Click Method	587
7.7.4	Modify Query Strings for Course Form	587
7.7.4.1	Modify SELECT Query String for Select Button Click Method	587
7.7.4.2	Modify SELECT Query String for CourseList Click Method	587
7.7.5	Other Modifications	588
7.8	Update and Delete Data in Database Using Stored Procedures	589
7.8.1	Update and Delete Data in Access Database Using Stored Procedures	590
7.8.1.1	Modify Existing Project	590
7.8.1.2	Create Stored Procedures in Microsoft Access Database	592
7.8.1.3	Call Stored Procedure to Update Faculty Information	594
7.8.1.4	Call Stored Procedure to Delete Faculty Information	595
7.8.2	Update and Delete Data in SQL Server Database Using Stored Procedures	598
7.8.2.1	Modify Existing Project to Create New Project	598
7.8.2.2	Develop Stored Procedure in SQL Server Database	601
7.8.2.3	Call Stored Procedure to Perform Data Updating and Deleting	605
7.8.3	Update and Delete Data in Oracle Database Using Stored Procedures	606
7.8.3.1	Modify Existing Project to Create New Project	607
7.8.3.2	Develop Stored Procedure in Oracle Database	610
7.8.3.3	Call Stored Procedures to Perform Data Updating and Deleting	614

7.9	Update and Delete Data in Databases Using LINQ to SQL Query	615
7.9.1	Create New Object of DataContext Class	616
7.9.2	Develop Codes for Select Button Click Method	617
7.9.3	Develop Codes for Update Button Click Method	618
7.9.4	Develop Codes for Delete Button Click Method	620
7.10	Chapter Summary	621
	Homework	621

8 Accessing Data in ASP.NET

8.1	What Is .NET Framework?	626
8.2	What Is ASP.NET and ASP.NET 3.5?	627
8.2.1	ASP.NET Web Application File Structure	629
8.2.2	ASP.NET Execution Model	630
8.2.3	What Really Happens When a Web Application Is Executed?	630
8.2.4	Requirements to Test and Run a Web Project	631
8.3	Develop ASP.NET Web Application to Select Data from SQL Server Databases	633
8.3.1	Create the User Interface—LogIn Form	634
8.3.2	Develop Codes to Access and Select Data from Database	635
8.3.3	Validate Data on Client Side	639
8.3.4	Create Second User Interface—Selection Page	640
8.3.5	Develop Codes to Open Other Page	641
8.3.6	Create Third User Interface—Faculty Page	643
8.3.7	Develop Codes to Select Desired Faculty Information	645
	8.3.7.1 Develop Codes for Page_Load Method	645
	8.3.7.2 Develop Codes for Select Button Method	646
	8.3.7.3 Develop Codes for Other Methods	648
8.3.8	Create Fourth User Interface—Course Page	651
	8.3.8.1 AutoPostBack Property of Listbox Control	653
8.3.9	Develop Codes to Select Desired Course Information	654
	8.3.9.1 Coding for Course Page Loading and Ending Methods	655
	8.3.9.2 Coding for Select Button Click Method	656
	8.3.9.3 Coding for SelectedIndexChanged Method of Listbox Control	658
	8.3.9.4 Coding for Other User-Defined Methods	659
8.4	Develop ASP.NET Web Application to Insert Data into SQL Server Databases	661
8.4.1	Create New Web Page Insert.aspx	662
8.4.2	Develop Codes to Perform Data Insertion Function	663
8.4.3	Develop Codes for Page_Load and Back Button Click Methods	664
8.4.4	Develop Codes for Insert Button Click Method	664
8.4.5	Develop Codes for Other Methods	666
8.4.6	Validate Data Insertion	668

- 8.5 Develop Web Applications to Update and Delete Data in SQL Server Databases 671
 - 8.5.1 Application User Interfaces 672
 - 8.5.2 Modify Coding for Faculty Page 672
 - 8.5.3 Develop Codes for Update Button Click Method 674
 - 8.5.4 Develop Codes for Delete Button Click Method 678
 - 8.5.4.1 Relationships Between Five Tables in Our Sample Database 678
 - 8.5.4.2 Data Deleting Order Sequence 679
 - 8.5.4.3 Use Cascade Deleting Option to Simplify Data Deleting 679
 - 8.5.4.4 Develop Codes to Perform Data Deleting 681
- 8.6 Develop ASP.NET Web Applications with LINQ to SQL Query 683
 - 8.6.1 Create New Web Form Page 684
 - 8.6.2 Create New Object of DataContext Class 685
 - 8.6.3 Coding for Data Selection Query 687
 - 8.6.4 Coding for Data Insertion Query 688
 - 8.6.5 Coding for Data Updating and Deleting Queries 689
- 8.7 Develop ASP.NET Web Application to Select Data from Oracle Databases 692
 - 8.7.1 Modify Connection String and Connection Object on LogIn Page 693
 - 8.7.2 Modify Query String in LogIn Page 694
 - 8.7.3 Modify Query String in Faculty Page 695
 - 8.7.4 Modify Query Strings in Course Page 696
- 8.8 Develop ASP.NET Web Application to Insert Data into Oracle Databases 700
 - 8.8.1 Add Two Controls to Faculty Page 701
 - 8.8.2 Modify Codes to Some Methods on Faculty Page 701
 - 8.8.3 Create Codes to Insert New Faculty on Faculty Page 704
- 8.9 Develop ASP.NET Web Application to Update and Delete Data in Oracle Databases 708
 - 8.9.1 Modify Project to Perform Data Updating 708
 - 8.9.1.1 Create Codes to Update Button Click Method 708
 - 8.9.2 Develop Stored Procedures to Perform Data Deleting 710
 - 8.9.2.1 Delete Existing Record from Faculty Table 711
 - 8.9.2.2 Develop Codes for Delete Button Click Method 712
 - 8.9.2.3 Validate Data Deleting Action 713
 - 8.9.2.4 Constraint Property—On Delete Cascade in Data Table 715
- 8.10 Chapter Summary 717
- Homework 718

9 ASP.NET Web Services

- 9.1 Web Services and Their Components 722
- 9.2 Procedures to Build a Web Service 723
 - 9.2.1 Structure of a Typical Web Service Project 724
 - 9.2.2 Real Considerations When Building a Web Service Project 724
 - 9.2.3 Procedures to Build an ASP.NET Web Service 725

xx Contents

9.3	Build ASP.NET Web Service Projects to Access SQL Server Database	726
9.3.1	Files and Items Created in the New Web Service Project	726
9.3.2	Feeling of Hello World Web Service Project as It Runs	729
9.3.3	Modify Default Web Service Project	732
9.3.4	Create a Base Class to Handle Error Checking for Our Web Service	734
9.3.5	Create Real Web Service Class	735
9.3.6	Add Web Methods into Our Web Service Class	736
9.3.7	Develop the Codes for Web Methods to Perform the Web Services	737
9.3.7.1	Web Service Connection Strings	737
9.3.7.2	Modify Existing Web Method	740
9.3.7.3	Develop Codes to Perform Database Queries	741
9.3.7.4	Develop Codes for User-Defined Methods	743
9.3.8	Develop Stored Procedures to Perform the Data Query	746
9.3.8.1	Develop Stored Procedure WebSelectFacultySP	746
9.3.8.2	Add Another Web Method to Call the Stored Procedure	747
9.3.9	Use DataSet as Returning Object for Web Method	748
9.3.10	Build Windows-Based Web Service Clients to Use the Web Services	751
9.3.10.1	Create a Web Service Proxy Class	752
9.3.10.2	Develop Graphic User Interface for Windows-Based Client Project	754
9.3.10.3	Develop Code to Use the Web Service	755
9.3.11	Build Web-Based Web Service Clients to Use the Web Service	762
9.3.11.1	Create a New Website Project and Add an Existing Web Page	763
9.3.11.2	Add a Web Service Reference and Modify the Web Form Window	763
9.3.11.3	Modify Codes for Related Methods	764
9.3.12	Deploy the Completed Web Service to Production Servers	769
9.3.12.1	Copy Web Service Files to Virtual Directory	770
9.3.12.2	Publish Precompiled Web Service	771
9.4	Build ASP.NET Web Service Project to Insert Data into SQL Server Database	772
9.4.1	Modify Existing Web Service Project	773
9.4.2	Web Service Project Development Procedure	774
9.4.3	Develop and Modify Codes for Code-Behind Page	774
9.4.3.1	Develop and Modify First Web Method SetSQLInsertSP	775
9.4.3.2	Develop Second Web Method GetSQLInsert	779
9.4.3.3	Develop and Modify Third Web Method SQLInsertDataSet	782
9.4.3.4	Develop Fourth Web Method GetSQLInsertCourse	788
9.4.4	Build Windows-Based Web Service Clients to Use Web Services	792
9.4.5	Build Web-Based Web Service Clients to Use Web Services	793
9.4.5.1	Create a New Website Project and Add Existing Web Page	793
9.4.5.2	Add Web Service Reference and Modify Web Form Window	794
9.4.5.3	Modify Codes for Related Methods	795

9.5	Build ASP.NET Web Service to Update and Delete Data for SQL Server Database	805
9.5.1	Modify Existing Web Service Project	807
9.5.2	Modify Related Web Methods	807
9.5.2.1	Modify Web Method from SetSQLInsertSP to SQLUpdateSP	808
9.5.2.2	Modify Web Method GetSQLInsert to GetSQLCourse	810
9.5.2.3	Modify Web Method GetSQLInsertCourse to GetSQLCourseDetail	811
9.5.2.4	Add New Web Method SQLDeleteSP	813
9.5.3	Develop Two Stored Procedures WebUpdateCourseSP and WebDeleteCourseSP	815
9.5.3.1	Develop Stored Procedure WebUpdateCourseSP	815
9.5.3.2	Develop Stored Procedure WebDeleteCourseSP	817
9.6	Build Windows-Based Web Service Clients to Use Web Services	827
9.7	Build Web-Based Web Service Clients to Use Web Services	827
9.7.1	Create New Website Project and Add Existing Web Page	828
9.7.2	Add Web Service Reference and Modify Web Form Window	828
9.7.3	Modify Codes for Related Methods	829
9.7.3.1	Modify Codes in Page_Load Method	830
9.7.3.2	Develop Codes for Update Button's Click Method	830
9.7.3.3	Develop Codes for Delete Button's Click Method	832
9.7.3.4	Modify Codes in Select Button's Click Method and Related Methods	833
9.7.3.5	Modify Codes in SelectedIndexChanged Method	835
9.8	Build ASP.NET Web Service Project to Access Oracle Database	838
9.8.1	Build Web Service Project WebServiceOracleSelect	839
9.8.2	Modify Connection String	840
9.8.3	Modify Namespace Directories	840
9.8.4	Modify Web Method GetSQLSelect and Related Methods	841
9.8.5	Modify Web Method GetSQLSelectSP and Related Methods	843
9.8.5.1	Modifications to Stored Procedure WebSelectFacultySP	843
9.8.5.2	Modifications to Codes in Web Method GetSQLSelectSP	847
9.8.6	Modify Web Method GetSQLSelectDataSet	848
9.9	Build Web Service Clients to Use the Web Service WebServiceOracleSelect	854
9.10	Build ASP.NET Web Service Project to Insert Data into Oracle Database	854
9.10.1	Build Web Service Project WebServiceOracleInsert	855
9.10.2	Modify Connection String	855
9.10.3	Modify Namespace Directories	856
9.10.4	Modify Web Method SetSQLInsertSP and Related Methods	856
9.10.5	Modify Web Method GetSQLInsert and Related Methods	858
9.10.6	Modify Web Method SQLInsertDataSet	860
9.10.7	Modify Web Method GetSQLInsertCourse and Related Methods	862
9.11	Build Web Service Clients to Use Web Service WebServiceOracleInsert	871
9.12	Build ASP.NET Web Service to Update and Delete Data for Oracle Database	872

xxii Contents

9.12.1	Build Web Service Project WebServiceOracleUpdateDelete	873
9.12.2	Modify Connection String	873
9.12.3	Modify Namespace Directories	874
9.12.4	Modify Web Method SQLUpdateSP and Related Methods	874
9.12.4.1	Develop Stored Procedure UpdateCourse_SP	877
9.12.5	Modify Web Method GetSQLCourse and Related Methods	879
9.12.6	Modify Web Method GetSQLCourseDetail and Related Methods	881
9.12.7	Modify Web Method SQLDeleteSP	883
9.12.7.1	Develop Stored Procedure WebDeleteCourseSP	885
9.12.7.2	Implement and Test Web Service Project	887
9.13	Build Web Service Clients to Use Web Service	889
9.14	Chapter Summary	890
	Homework	891

Index 895

About the Author 903

Preface

Databases have become an integral part of our modern daily life. We are an information-driven society. Database technology has a direct impact on our daily lives. Decisions are routinely made by organizations based on the information collected and stored in databases. A record company may decide to market certain CDs in selected regions based on the music preferences of teenagers. Grocery stores display more popular items at eye level and reorders are based on the inventories taken at regular intervals. Other examples include patients' records in hospitals, customers' account information in banks, book orders by the libraries, club memberships, auto part orders, winter cloth stock by department stores, and many others.

In addition to database management systems, in order to effectively apply and implement databases in real industrial or commercial systems, a good graphic user interface (GUI) is needed to enable users to access and manipulate their records or data in databases. Visual C#.NET is an ideal candidate to provide this GUI functionality. Unlike other programming languages, Visual C#.NET has advantages such as easy-to-learn and easy-to-be-understood with few learning curves. Beginning from Visual Studio.NET 2003, Microsoft integrated a few programming languages such as Visual C++, Visual Basic, C#, and Visual J# into a dynamic model called .NET Framework that makes Internet and Web programming easy and simple, and any language integrated in this model can be used to develop professional and efficient Web applications that can be used to communicate with others via Internet. ADO.NET and ASP.NET are two important submodels of .NET Framework. The former provides all components, including Data Providers, DataSet and DataTable, to access and manipulate data against different databases. The latter provides support to develop Web applications and Web services in ASP.NET environment to assist users to exchange information between clients and servers easily and conveniently.

This book is mainly designed for college students and software programmers who want to develop practical and commercial database programming with Visual C#.NET 2008 and relational databases such as Microsoft Access, SQL Server 2005, and Oracle Database 10g XE. The book provides a detailed description about practical considerations and applications in database programming with Visual C# 2008 with authentic examples and detailed explanations. More important, a new writing style is developed and implemented in this book, combined with real examples, to provide readers with a clear picture as how to handle the database programming issues in Visual C#.NET 2008 environment.

The outstanding features of this book include, but are not limited to:

1. A novel writing style is adopted to try to attract students' or beginning programmers' interested in learning and developing practical database programs, and to avoid the headache caused by using huge blocks of codes in the traditional database programming books.

2. A real completed sample database CSE_DEPT with three versions (Microsoft Access 2007, SQL Server 2005 SP2, and Oracle Database 10g XE Release 2) is provided and used throughout the entire book. Step by step, detailed illustrations and descriptions about how to design and build a practical relational database are provided.
3. Both fundamental and advanced database-programming techniques are covered to convenience both beginning students and experienced programmers.
4. The new database query technique LINQ API, which includes LINQ to Objects, LINQ to DataSet, LINQ to SQL, LINQ to Entities, and LINQ to XML, is discussed, analyzed, and implemented in actual projects with line-by-line explanations.
5. More than 60 real sample database programming projects are covered in the book with detailed illustrations and explanations to help students understand key techniques and programming technologies.
6. Three types of popular databases are covered and discussed in detail with practical sample examples: Microsoft Access, SQL Server 2005, and Oracle Database 10g Express Edition (XE).
7. Various actual data providers are discussed and implemented in the sample projects, such as the SQL Server and Oracle data providers. Instead of using the OleDb to access the SQL Server or Oracle databases, the real SQL Server and Oracle data providers are utilized to connect to the Visual C#.NET 2008 directly to perform data operations.
8. Homework and selected solutions are provided for each chapter to strengthen and improve students' learning and understanding topics they have studied.
9. PowerPoint teaching slides are also provided to help instructors in teaching and organizing their classes.
10. This is a good textbook for college students, as well as a good reference book for programmers, software engineers, and academic researchers.

I sincerely hope that this book provides useful and practical assistance and guidelines to all readers and users who adopt this book. I will be more than happy to know that you have been able to develop and build professional and practical database applications with the help of this book.

YING BAI

*Charlotte, North Carolina
January 2010*

Acknowledgment

The first and most special thanks to my wife, Yan Wang. I could not have finished this book without her sincere encouragement and support.

Special thanks to Dr. Satish Bhalla, who made great contributions to Chapter 2. Dr. Bhalla is a specialist in database programming and management, especially in SQL Server, Oracle, and DB2. Dr. Bhalla spent a lot of time preparing materials for the first part on Chapter 2 and he deserves this thank you.

Many thanks to Mr. Steve Welch who makes this book available to you. You would not find this book on the market without Mr. Steve Welch's deep perspective and hard work. The same thanks are extended to the editorial team of this book. Without their contributions, it wouldn't have been impossible for this book to be published.

This thank you should also be extended to the following book reviewers for their precious opinions to this book:

- Dr. Jifeng Xu, Research Scientist, The Boeing Company
- Dr. Xiaohong Yuan, Associate Professor, Department of Computer Science, North Carolina A&T State University
- Dr. Daoxi Xiu, Application Analyst Programmer, North Carolina Administrative Office of the Courts
- Dr. Dali Wang, Assistant Professor, Department of Physics and Computer Science, Christopher Newport University

Finally but not the last, thanks should be forwarded to all the guys who supported me to finish this book.

Y. B.

Chapter 1

Introduction

During my years of teaching database programming at the college level, it was difficult to find a good textbook on this topic. I have had to combine a few different professional books together as references to teach this course. Most of these books are specially designed for programmers or software engineers, which cover a lot of programming strategies and present huge blocks of code, which can give a terrible headache to the college students or beginning programmers who are new to Visual Studio.NET and database programming. I have had to prepare class presentations and develop homework problems and exercises for my students. I used to dream that one day I would find a good textbook that would be suitable for college students or beginning programmers and help them to learn and master database programming with Visual C#.NET easily and conveniently. Finally, I decided to do something myself about filling this lack of a good textbook.

Another reason for writing this book is the job market. As you know, most industrial and commercial companies in the United States are database applications businesses such as manufactures, banks, hospitals, and retailers. The majority of them need professional people to develop and build database-related applications, but not database management and design systems. To enable our students to become good candidates for these companies, we need to create a book like this one.

Unlike most database programming books on the current market, which discuss and present database programming techniques with huge blocks of programming codes from the first page to the last page, this book uses a new writing style to show readers, especially college students, how to develop professional and practical database programs in Visual C# 2008 by using Visual Studio.NET 2008 Design Tools and Wizards related to ADO.NET 3.5 and to apply codes that are autogenerated by using Wizards. By using this new style, the headaches caused by using huge blocks of programming code can be eliminated. Instead, a simple and easy way to create database programs using the design tools can be developed to enable students to build professional and practical database programs in more efficient and interesting ways.

There are many different database programming books available on the market. However, rarely do you find a book like this one that implements a novel writing style to attract students interested in this topic. To meet the needs of some experienced or advanced students or software engineers, the book contains two programming

methods: the fundamental database programming method—Visual Studio.NET 2008 Design Tools and Wizards—and an advanced database programming method—the runtime objects method. In the second method, all database-related objects are created and applied during or when your project is running by utilizing quite few blocks of code.

OUTSTANDING FEATURES OF THE BOOK

1. A novel writing style is adopted to try to attract students and beginning programmers interesting in learning and developing practical database programs, and to avoid the headaches caused by using huge blocks of code in the traditional database programming books.
2. Covers both fundamental and advanced database programming techniques for both beginning and experienced students as well as programmers.
3. Three types of popular databases are covered and discussed in detail with practical examples: Microsoft Access 2007, SQL Server 2005 Express SP2, and Oracle Database 10g Express Edition Release 2. Unlike traditional database programming books, in which some classical commercial databases such as Northwind are used, three customer-built databases, CSE_DEPT.accdb, CSE_DEPT.mdf, and the Oracle version of CSE_DEPT, are created and implemented in all sample projects throughout the whole book.
4. The new database query technique, LINQ API, which includes LINQ to Objects, LINQ to DataSet, LINQ to SQL, LINQ to Entities, and LINQ to XML, is discussed, analyzed, and implemented in actual projects with line-by-line explanations.
5. The various actual data providers are discussed and implemented in the sample projects, such as the SQL Server and Oracle data providers. Instead of using the OleDb to access the SQL Server or Oracle databases, the real SQL Server and Oracle data providers are utilized to connect to the Visual C#.NET 2008 directly to perform data operations.
6. More than 60 real sample database programming projects are covered with detailed illustrations and explanations to help students understand key techniques and programming technologies.
7. Homework and selected solutions are provided for each chapter to strengthen and improve student learning and understanding.
8. PowerPoint teaching slides are also provided to help instructors in teaching and organizing their classes.
9. It is a good textbook for college students and a good reference book for programmers, software engineers, and academic researchers.

TARGET AUDIENCE

This book is designed for college students and software programmers who want to develop practical and commercial database programming with Visual C#.NET 2008 and relational databases such as Microsoft Access, SQL Server 2005, and Oracle Database 10g XE. Fundamental knowledge and understanding of Visual C#.NET and Visual Studio.NET IDE is assumed.

TOPICS COVERED

Nine chapters are included in this book. The contents of each chapter is summarized as follows:

- Chapter 1 provides an introduction and summary of the whole book.
- Chapter 2 provides a detailed discussion and analysis of the structure and components about relational databases. Some key technologies in developing and designing databases are also discussed. The procedures and components used to develop a practical relational database with three database versions such as Microsoft Access, SQL Server 2005, and Oracle Database 10g XE are analyzed in detail with some real data tables in the sample database CSE_DEPT.
- Chapter 3 provides an introduction to ADO.NET, which includes the architecture, organization, and components of ADO.NET 2.0 as well as ADO.NET 3.5. Detailed discussions and descriptions are provided to give readers both fundamental and practical ideas and pictures on how to use components in ADO.NET to develop professional data-driven applications. Two ADO.NET architectures are discussed to enable users to follow the directions to design and build their own projects based on the different organizations of ADO.NET. Four popular data providers, such as OleDb, ODBC, SQL Server, and Oracle, are discussed in detail. The basic ideas and implementation examples of DataTable and DataSet are also analyzed and described with some real coding examples. The new query technique, LINQ to ADO.NET, is introduced and discussed in detail with some actual sample projects. The latest version of ADO.NET, ADO.NET 3.5, is discussed with some examples given in the last section. The properties and functionalities of ADO.NET 3.5 Entity Framework (EF) and ADO.NET 3.5 Entity Framework Tools (EFT) are discussed in detail. The core of ADO.NET 3.5 EF, Entity Data Model, and associated Item template, Wizard and Designer, is also discussed and analyzed with a real project example EDMModel.
- Chapter 4 provides a detailed discussion and analysis about the Language-Integrated Query (LINQ), which includes LINQ to Objects, LINQ to DataSet, LINQ to SQL, LINQ to Entities, and LINQ to XML. An introduction to the LINQ general programming guide is provided in the first part in this chapter. Some popular interfaces widely used in LINQ, such as IEnumerable, IEnumerable<T>, IQueryable, and IQueryable<T>, and Standard Query Operators (SQO) including the deferred and nondeferred SQO, are discussed. An introduction to LINQ Query is given in the second section of this chapter. Following this introduction, a detailed discussion and analysis about the LINQ queries implemented for different data sources are provided in detail.
- Starting in Chapter 5, the real database programming techniques with Visual C#.NET 2008 such as data selection queries are provided and discussed. This chapter is divided into two parts: Part I contains the detailed descriptions on how to develop professional data-driven applications with the help of the Visual Studio.NET design tools and wizards with some real projects, and this part contains a lot of hidden codes that are created by Visual Studio.NET 2008 automatically when using these design tools and wizards. Therefore, the coding for this part is very simple and easy. Part II covers an advanced technique, the runtime objects method, used in developing and building professional data-driven applications. Detailed discussions and descriptions about how to build professional and practical database applications using this runtime method are provided combined with four real projects. New query techniques, such as LINQ to DataSet and LINQ to SQL, are also discussed and implemented with some real sample projects.

4 Chapter 1 Introduction

- Chapter 6 provides detailed discussions and analyses about three popular data insertion methods with three different databases—Microsoft Access, SQL Server 2005, and Oracle:
 - Using the TableAdapter DBDirect method and TableAdapter.Insert() method
 - Using the TableAdapter Update() method to insert new records that have already been added into the DataTable in the DataSet
 - Using the Command object ExecuteNonQuery() method

This chapter is also divided into two parts: Methods 1 and 2 are related to Visual Studio.NET design tools and wizards and therefore are covered in Part I. The third method is related to runtime objects and therefore is covered in Part II. Nine actual projects are used to illustrate how to perform the data insertion into three different databases: Microsoft Access, SQL Server 2005, and Oracle Database 10g XE. Some professional and practical data validation methods are also discussed in this chapter to confirm the data insertion. LINQ to DataSet and LINQ to SQL queries are introduced and implemented in this chapter with some real sample projects.

- Chapter 7 provides discussions and analyses on three popular data updating and deleting methods with seven actual project examples:
 - Using TableAdapter DBDirect methods such as TableAdapter.Update() and TableAdapter.Delete() to update and delete data directly from the databases
 - Using TableAdapter.Update() method to update and execute the associated TableAdapter's properties such as UpdateCommand or DeleteCommand to save changes made for the tables in the DataSet to the tables in the database
 - Using the runtime objects method to develop and execute the Command's method ExecuteNonQuery() to update or delete data in the database directly

Chapter 7 is also divided into two parts: Methods 1 and 2 are related to Visual Studio.NET design tools and wizards and therefore are covered in Part I. The third method is related to runtime objects and is covered in Part II. Seven actual projects are used to illustrate how to perform updating and deleting data against three different databases: Microsoft Access, SQL Server 2005, and Oracle Database 10g XE. Some professional and practical data validation methods are also discussed in this chapter to confirm the data updating and deleting actions. The key points in performing the data updating and deleting actions against a relational database such as the order to execute data updating and deleting between the parent and child tables are also discussed and analyzed. Updating and deleting data using LINQ to DataSet and LINQ to SQL queries are also discussed and implemented in this chapter with some real sample projects.

- Chapter 8 provides introductions and discussions about the developments and implementations of ASP.NET Web applications in the Visual C#.NET 2008 environment. At the beginning of this chapter, a detailed and completed description about ASP.NET and .NET Framework is provided, and this part is especially useful and important to students and programmers who do not have any knowledge or background in the Web application development and implementation. Following the introduction section, a detailed discussion on how to install and configure the environment to develop ASP.NET Web applications is provided. Some essential tools such as the Web server, IIS, and FrontPage Server Extension 2000, as well as the installation process of these tools, are introduced and discussed in detail. Starting from Section 8.3, the detailed development and building process of ASP.NET Web applications to access databases are discussed with six actual Web application projects. Two popular databases, SQL Server and Oracle, are utilized as the target databases for those development and building processes. LINQ to DataSet and LINQ to SQL queries are also discussed and involved in the Web application projects.

- Chapter 9 provides an introduction and discussions about the development and implementation of ASP.NET Web services in the Visual C#.NET 2008 environment. A detailed discussion and analysis about the structure and components of the Web services is provided at the beginning of this chapter. Two popular databases, SQL Server and Oracle, are discussed and used for three pairs of example Web service projects, which include:
 - WebServiceSQLSelect and WebServiceOracleSelect
 - WebServiceSQLInsert and WebServiceOracleInsert
 - WebServiceSQLUpdateDelete and WebServiceOracleUpdateDelete

Each Web service contains different Web methods that can be used to access different databases and perform the desired data actions such as Select, Insert, Update, and Delete via the Internet. To implement these Web services, different Web service client projects are also developed in this chapter. Both Windows-based and Web-based Web service client projects are discussed and built for each kind of Web service listed above. Eighteen projects, including the Web service projects and the associated Web service client projects, are developed in this chapter. All projects have been debugged and tested and can be run in any Windows-compatible operating systems such as Windows 95, 98, 2000, XP, and Vista.

ORGANIZATION OF THE BOOK AND HOW TO USE IT

This book is designed for both college students who are new to database programming with Visual C#.NET and professional database programmers who have professional experience with this topic.

Chapters 2 and 3 provide the fundamentals on database structures and components, ADO.NET, and components it covers. Chapter 4 covers the introduction and implementation of LINQ queries, which include popular LINQ queries, such as LINQ to Objects, LINQ to ADO.NET, and LINQ to XML, with sample projects. Chapters 5, 6, and 7 are each divided into two parts: a fundamental part and an advanced part. The data-driven applications developed with design tools and wizards provided by Visual Studio .NET, which can be considered as the fundamental part, have less coding loads and therefore are more suitable for students and programmers who are new to database programming with Visual C#.NET. Part II contains the runtime objects method and covers a lot of coding developments to perform the different data actions against the database, and this method is more flexible and convenient for experienced programmers.

Chapters 8 and 9 give a full discussion and analysis of the development and implementation of ASP.NET Web applications and Web services. These technologies are necessary to students and programmers who want to develop and build Web applications and Web services to access and manipulate data via the Internet.

Based on the organization described above, this book can be used as two categories such as Level I and Level II, which is shown in Figure 1.1.

For undergraduate college students or beginning software programmers, it is highly recommended to learn and understand the contents of Chapters 2, 3, and 4 and Part I of Chapters 5, 6, and 7 since these cover the fundamental techniques in database programming with Visual C#.NET 2008. Chapters 8 and 9 are optional depending on the time and schedule.

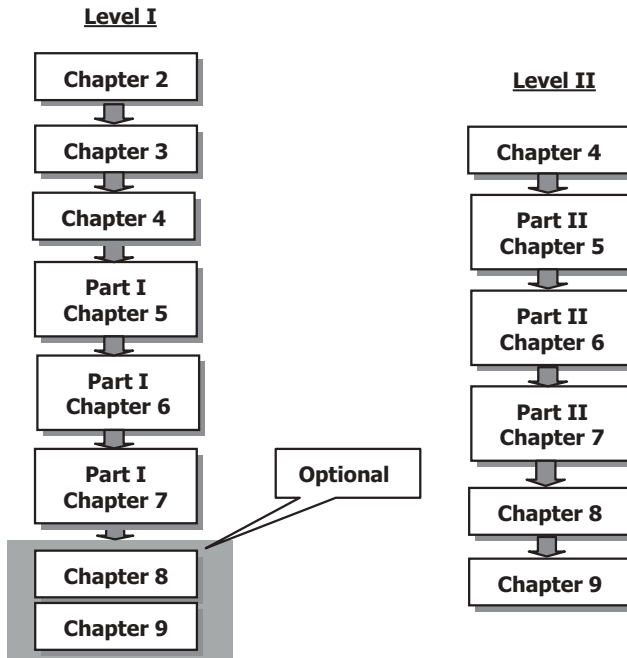


Figure 1.1 Two levels in this book.

In Chapter 2, a detailed introduction on how to design and build a practical relational sample database CSE_DEPT with three database versions is provided. Step by step, a detailed description is given to illustrate how to design and set up relationships between parent and child tables using the primary key and foreign keys for Microsoft Access 2007, SQL Server 2005 SP2, and Oracle Database 10g XE Release 2 databases. In Part I of Chapters 5 to 7, Visual Studio.NET design tools and wizards are discussed and analyzed in detail to show readers how to use them to design and build professional database programs with Visual C#.NET easily and conveniently.

For experienced college students and software programmers who have already some knowledge of database programming, it is recommended to learn and understand the contents of Part II of Chapters 5 to 7 as well as Chapters 4, 8, and 9 since the runtime data objects method and some sophisticated database programming techniques such as LINQ queries, joined-table query, nested stored procedures, and Oracle Package are discussed and illustrated with real examples. Also the ASP.NET Web applications and ASP.NET Web services are discussed and analyzed with 24 real database program examples for SQL Server 2005 and Oracle Database 10g XE.

HOW TO USE THE SOURCE CODE AND SAMPLE DATABASES

All source codes for each actual project developed in this book are available. All projects are categorized into the associated chapters that are located in the folder **DBProjects**

located at ftp://ftp.wiley.com/public/sci_tech_med/practical_database. You can copy or download these codes into your computer and run each project as you like. To successfully run these projects on your computer, the following conditions must be met:

- Visual Studio.NET 2008 or higher must be installed on your computer.
- Three databases management systems—Microsoft Access 2007 (Microsoft Office 2007), Microsoft SQL Server 2005 Management Studio Express, and Oracle Database 10g Express Edition (XE)—must be installed on your computer.
- Three versions of sample database—CSE_DEPT.accdb, CSE_DEPT.mdf, and Oracle version of CSE_DEPT—must be installed on your computer in the appropriate folders.
- To run projects developed in Chapters 8 and 9, in addition to conditions listed above, an Internet Information Services (IIS) such as FrontPage Server Extension 2000 or 2002 must be installed on your computer. It works as a pseudoserver for those projects.

The following appendices are useful when one needs some references and practical knowledge to install database management systems and develop actual database application projects:

- **Appendix A:** Provides a completed SQL commands reference.
- **Appendix B:** Provides detailed descriptions about download and installation of Microsoft SQL Server 2005 Express SP2.
- **Appendix C:** Provides detailed descriptions about download and installation of Oracle Database 10g Express Edition (XE) Release 2.
- **Appendix D:** Provides detailed discussions on how to create user database in Oracle Database 10g XE and how to duplicate Oracle 10g user database using Unload and Load methods.
- **Appendix E:** Provides detailed discussions on how to add and connect Oracle 10g XE database into Visual C#.NET applications using the Visual Studio.NET design tools and wizards.
- **Appendix F:** Provides detailed discussions on how to use three sample databases: CSE_DEPT.accdb, CSE_DEPT.mdf, and Oracle version of CSE_DEPT.

All of these appendices can be found in the folder **Appendix** located at ftp://ftp.wiley.com/public/sci_tech_med/practical_database.

Three sample database files, such as **CSE_DEPT.accdb**, **CSE_DEPT.mdf**, and Oracle version of **CSE_DEPT**, are located at the different folders, such as **Access**, **SQLServer**, and **Oracle**, which are subfolders and under the folder **Database**, which is located at ftp://ftp.wiley.com/public/sci_tech_med/practical_database. To use these databases for your applications or sample projects, refer to Appendix F.

INSTRUCTOR AND CUSTOMER SUPPORT

The teaching materials for all chapters have been extracted and represented by a sequence of Microsoft PowerPoint files, one file for each chapter. The interested instructors can find those teaching materials in the folder **TeachingPPT**, which located at ftp://ftp.wiley.com/public/sci_tech_med/practical_database.

8 Chapter 1 Introduction

Email support is available to readers of this book. When you send email to us, please provide the following information:

- The detailed description about your problems, including the error message and debug message, as well as the error or debug number if it is provided.
- Your name, job title, and company name.

Please send all questions to the email address: baidbbook@bellsouth.net.

HOMEWORK SOLUTIONS

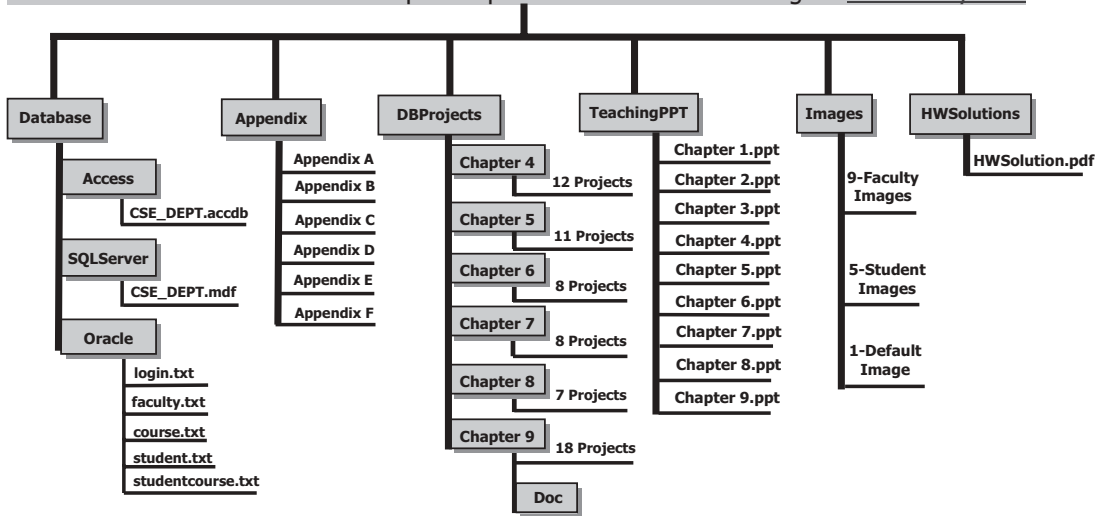
Selected homework solutions are available to instructors or preferred readers. This solution is user name/password protected, and instructors need to send a request to the publisher to get a valid user name/password if they want to access it. Then you can download homework solutions from the folder **HWSolutions** located at ftp://ftp.wiley.com/public/sci_tech_med/practical_database.

Figure 1.2 shows where the materials related to this book can be found on the web.

The Book Related Materials on the Web Sites

FOR INSTRUCTORS:

Instructor materials are available upon request from the book's listing on www.wiley.com



FOR STUDENTS:

ftp://ftp.wiley.com/public/sci_tech_med/practical_database

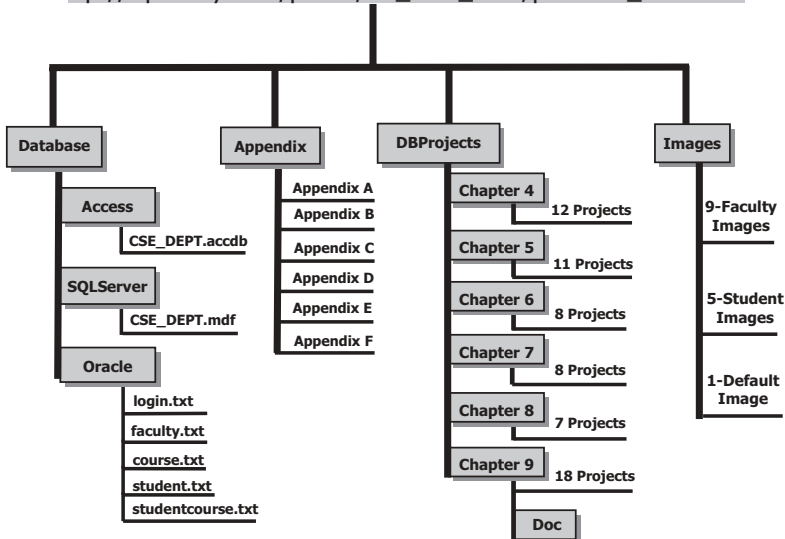


Figure 1.2

Chapter 2

Introduction to Databases

SATISH BHALLA AND YING BAI

Databases have become an integral part of our modern daily life. We are an information-driven society. We generate large amounts of data that is analyzed and converted into information. A recent example of biological data generation is the Human Genome Project, which was jointly sponsored by the Department of Energy and the National Institutes of Health. Many countries in the world participated in this venture for 10 years. The project was a tremendous success. It was completed in 2003 and resulted in the generation of huge amounts of genome data, currently stored in databases around the world. Scientists will be analyzing this data for years to come.

Database technology has a direct impact on our daily lives. Decisions are routinely made by organizations based on the information collected and stored in the databases. A record company may decide to market certain CDs in selected regions based on the music preferences of teenagers. Grocery stores display more popular items at eye level and reorders are based on the inventories taken at regular intervals. Other examples include book orders by libraries, club memberships, auto parts orders, winter clothes stock by department stores, and many others.

Database management programs have been in existence since the 1960s. However, it was not until the 1970s when E. F. Codd proposed the then revolutionary Relational Data Model that database technology really took off. In the early 1980s it received a further boost with the arrival of personal computers (PCs) and microcomputer-based data management programs such as dBase II (later followed by dBase III and IV). Today we have a plethora of vastly improved programs for PCs and mainframe computers, including Microsoft Access, IBM DB2, Oracle, Sequel Server, My SQL, and others.

This chapter covers the basic concepts of database design followed by implementation of a specific relational database to illustrate the concepts discussed here. The sample database, CSE_DEPT, is used as a running example. The database creation is shown in detail using Microsoft Access, SQL Server, and Oracle. The topics discussed in this chapter include:

12 Chapter 2 Introduction to Databases

- What are databases and database programs?
 - File processing system
 - Integrated databases
- Various approaches to developing a database
- Relational Data Model and Entity-Relationship Model (ER)
- Identifying keys
 - Primary keys, foreign keys, and referential integrity
- Defining relationships
- Normalizing the data
- Implementing the relational database
 - Create Microsoft Access sample database
 - Create Microsoft SQL Server 2005 sample database
 - Create Oracle sample database

2.1 WHAT ARE DATABASES AND DATABASE PROGRAMS?

A modern-day database is a structured collection of data stored in a computer. The term *structured* implies that each record in the database is stored in a certain format. For example, all entries in a phone book are arranged in a similar fashion. Each entry contains a name, an address, and a telephone number of a subscriber. This information can be queried and manipulated by database programs. The data retrieved in answer to queries become information that can be used to make decisions. The databases may consist of a single table or related multiple tables. The computer programs used to create, manage, and query databases are known as a database management systems (DBMS). Just like the databases, the DBMSs vary in complexity. Depending on the need of a user, one can use either a simple application or a robust program. Some examples of these programs were given earlier.

2.1.1 File Processing System

A file processing system is a precursor of the integrated database approach. The records for a particular application are stored in a file. An application program is needed to retrieve or manipulate data in this file. Thus various departments in an organization will have their own file processing systems with their individual programs to store and retrieve data. The data in various files may be duplicated and not available to other applications. This causes redundancy and may lead to inconsistency, meaning that various files that supposedly contain the same information may actually contain different data values. Thus duplication of data creates problems of data integrity. Moreover, it is difficult to provide access to multiple users with the file processing systems without granting them access to the respective application programs, which manipulate the data in those files.

The file processing system may be advantageous under certain circumstances. For example, if data is static and a simple application will solve the problem, a more expensive DBMS is not needed. For example, in a small business environment you want to keep

track of the inventory of the office equipment purchased only once or twice a year. The data can be kept in an Excel spreadsheet and manipulated with ease from time to time. This avoids the need to purchase an expensive database program and hiring a knowledgeable database administrator. Before the DBMSs became popular, the data was kept in files and application programs were developed to delete, insert, or modify records in the files. Since specific application programs were developed for specific data, these programs lasted for months or years before modifications were necessitated by business needs.

2.1.2 Integrated Databases

A better alternative to a file processing system is an integrated database approach. In this environment all data belonging to an organization is stored in a single database. The database is not a mere collection of files; there is a relation between the files. Integration implies a logical relationship, usually provided through a common column in the tables. The relationships are also stored within the database. A set of sophisticated programs known as a database management system (DBMS) is used to store, access, and manipulate the data in the database. Details of data storage and maintenance are hidden from the user. The user interacts with the database through the DBMS. A user may interact either directly with the DBMS or via a program written in a programming language such as C++, Java, or Visual Basic. Only the DBMS can access the database. Large organizations employ database administrators (DBAs) to design and maintain large databases.

There are many advantages to using an integrated database approach over that of a file processing approach:

1. **Data Sharing** The data in the database is available to a large number of users who can access the data simultaneously and create reports and manipulate the data given proper authorization and rights.
2. **Minimizing Data Redundancy** Since all the related data exists in a single database, there is a minimal need of data duplication. The duplication is needed to maintain relationships between various data items.
3. **Data Consistency and Data Integrity** Reducing data redundancy will lead to data consistency. Since data is stored in a single database, enforcing data integrity becomes much easier. Furthermore, the inherent functions of the DBMS can be used to enforce the integrity with minimum programming.
4. **Enforcing Standards** DBAs are charged with enforcing standards in an organization. A DBA takes into account the needs of various departments and balances it against the overall need of the organization. A DBA defines various rules such as documentation standards, naming conventions, update and recovery procedures, and the like. It is relatively easy to enforce these rules in a database system, since it is a single set of programs that is always interacting with the data files.
5. **Improving Security** Security is achieved through various means such as controlling access to the database through passwords, providing various levels of authorizations, data encryption, providing access to restricted views of the database and so forth.
6. **Data Independence** Providing data independence is a major objective for any database system. Data independence implies that even if the physical structure of a database changes, the applications are still accessed as before the changes were implemented. In other words

the applications are immune to the changes in the physical representation and access techniques.

The downside of using an integrated database approach has mainly to do with exorbitant costs associated with it. The hardware, the software, and maintenance are expensive. Providing security, concurrency, integrity, and recovery may add further to this cost. Furthermore, since DBMS consists of a complex set of programs, trained personnel are needed to maintain it.

2.2 DEVELOP A DATABASE

The database development process may follow a classical systems development life cycle.

1. **Problem Identification** Interview the user, identify user requirements. Perform preliminary analysis of user needs.
2. **Project Planning** Identify alternative approaches to solving the problem. Does the project need a database? If so, define the problem. Establish the scope of the project.
3. **Problem Analysis** Identify specifications for the problem. Confirm the feasibility of the project. Specify detailed requirements.
4. **Logical Design** Delineate detailed functional specifications. Determine screen designs, report layout designs, data models, and the like.
5. **Physical Design** Develop physical data structures.
6. **Implementation** Select DBMS. Convert data to conform to DBMS requirements. Code programs; perform testing.
7. **Maintenance** Continue program modification until desired results are achieved.

An alternative approach to developing a database is through a phased process that will include designing a conceptual model of the system that will imitate the real-world operation. It should be flexible and change when the information in the database changes. Furthermore it should not be dependent upon the physical implementation. This process follows the following phases:

1. **Planning and Analysis** This phase is roughly equivalent to the first three steps mentioned above in the systems development life cycle. This includes requirement specifications, evaluating alternatives, and determining input, output, and reports to be generated.
2. **Conceptual Design** Choose a data model and develop a conceptual schema based on the requirement specification that was laid out in the planning and analysis phase. This conceptual design focuses on how the data will be organized without having to worry about the specifics of the tables, keys and attributes. Identify the entities that will represent tables in the database; identify attributes that will represent fields in a table; and identify each entity attribute relationship. Entity relationship diagrams provide a good representation of the conceptual design.
3. **Logical Design** Conceptual design is transformed into a logical design by creating a roadmap of how the database will look before actually creating the database. Data model is identified; usually it is the relational model. Define the tables (entities) and fields (attributes). Identify primary and foreign keys for each table. Define relationships between the tables.

4. **Physical Design** Develop physical data structures; specify file organization, and data storage. Take into consideration the availability of various resources including hardware and software. This phase overlaps with the implementation phase. It involves the programming of the database taking into account the limitations of the DBMS used.
5. **Implementation** Choose the DBMS that will fulfill user needs. Implement the physical design. Perform testing. Modify if necessary or until the database functions satisfactorily.

2.3 SAMPLE DATABASE

We will use CSE_DEPT database to illustrate some essential database concepts. Tables 2.1 to 2.5 show sample data tables stored in this database.

The data in CSE_DEPT database is stored in five tables—LogIn, Faculty, Course, Student, and StudentCourse. A table consists of rows and columns (Figure 2.1). A row represents a record and the column represents a field. A row is called a tuple and a column is called an attribute. For example, Student table has seven columns or fields—student_id, name, gpa, credits, major, schoolYear, and email. It has five records or rows.

Table 2.1 LogIn Table

user_name	pass_word	faculty_id	student_id
abrown	america	B66750	
ajade	tryagain		A97850
awoods	smart		A78835
banderson	birthday	A52990	
bvalley	see		B92996
dangles	tomorrow	A77587	
hsmith	try		H10210
jerica	excellent		J77896
jhenry	test	H99118	
jking	goodman	K69880	
sbhalla	india	B86590	
sjohnson	jermany	J33486	
ybai	reback	B78880	

Table 2.2 Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.edu
A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@college.edu
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

Table 2.3 Course Table

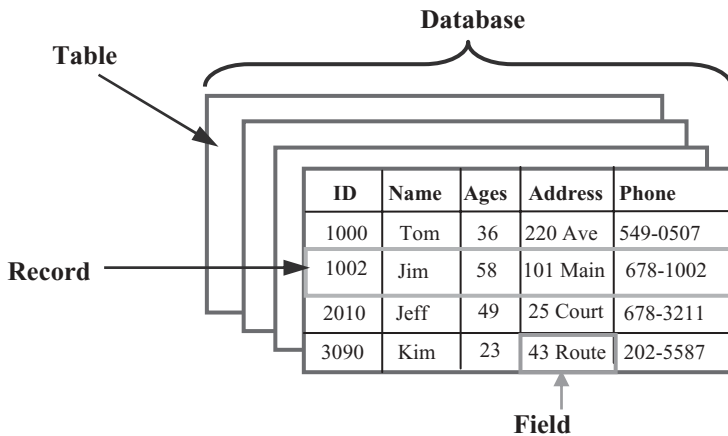
course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-131A	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	28	A52990
CSC-131B	Computers in Society	3	TC-114	M-W-F: 9:00-9:55 AM	20	B66750
CSC-131C	Computers in Society	3	TC-109	T-H: 11:00-12:25 PM	25	A52990
CSC-131D	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	30	B86590
CSC-131E	Computers in Society	3	TC-301	M-W-F: 1:00-1:55 PM	25	B66750
CSC-131I	Computers in Society	3	TC-109	T-H: 1:00-2:25 PM	32	A52990
CSC-132A	Introduction to Programming	3	TC-303	M-W-F: 9:00-9:55 AM	21	J33486
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-230	Algorithms & Structures	3	TC-301	M-W-F: 1:00-1:55 PM	20	A77587
CSC-232A	Programming I	3	TC-305	T-H: 11:00-12:25 PM	28	B66750
CSC-232B	Programming I	3	TC-303	T-H: 11:00-12:25 PM	17	A77587
CSC-233A	Introduction to Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	18	H99118
CSC-233B	Introduction to Algorithms	3	TC-302	M-W-F: 11:00-11:55 AM	19	K69880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSC-234B	Data Structure & Algorithms	3	TC-114	T-H: 11:00-12:25 PM	15	J33486
CSC-242	Programming II	3	TC-303	T-H: 1:00-2:25 PM	18	A52990
CSC-320	Object Oriented Programming	3	TC-301	T-H: 1:00-2:25 PM	22	B66750
CSC-331	Applications Programming	3	TC-109	T-H: 11:00-12:25 PM	28	H99118
CSC-333A	Computer Arch & Algorithms	3	TC-301	M-W-F: 10:00-10:55 AM	22	A77587
CSC-333B	Computer Arch & Algorithms	3	TC-302	T-H: 11:00-12:25 PM	15	A77587
CSC-335	Internet Programming	3	TC-303	M-W-F: 1:00-1:55PM	25	B66750
CSC-432	Discrete Algorithms	3	TC-206	T-H: 11:00-12:25 PM	20	B86590
CSC-439	Database Systems	3	TC-206	M-W-F: 1:00-1:55 PM	18	B86590
CSE-138A	Introduction to CSE	3	TC-301	T-H: 1:00-2:25 PM	15	A52990
CSE-138B	Introduction to CSE	3	TC-109	T-H: 1:00-2:25 PM	35	J33486
CSE-330	Digital Logic Circuits	3	TC-305	M-W-F: 9:00-9:55 AM	26	K69880
CSE-332	Foundations of Semiconductors	3	TC-305	T-H: 1:00-2:25 PM	24	K69880
CSE-334	Elec. Measurement & Design	3	TC-212	T-H: 11:00-12:25 PM	25	H99118
CSE-430	Bioinformatics in Computer	3	TC-206	Thu: 9:30-11:00 AM	16	B86590
CSE-432	Analog Circuits Design	3	TC-309	M-W-F: 2:00-2:55 PM	18	K69880
CSE-433	Digital Signal Processing	3	TC-206	T-H: 2:00-3:25 PM	18	H99118
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-436	Automatic Control and Design	3	TC-305	M-W-F: 10:00-10:55 AM	29	J33486
CSE-437	Operating Systems	3	TC-303	T-H: 1:00-2:25 PM	17	A77587
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880
CSE-439	Special Topics in CSE	3	TC-206	M-W-F: 10:00-10:55 AM	22	J33486

Table 2.4 Student Table

student_id	student_name	gpa	credits	major	schoolYear	email
A78835	Andrew Woods	3.26	108	Computer Science	Senior	awoods@college.edu
A97850	Ashly Jade	3.57	116	Information System Engineering	Junior	ajade@college.edu
B92996	Blue Valley	3.52	102	Computer Science	Senior	bvalley@college.edu
H10210	Holes Smith	3.87	78	Computer Engineering	Sophomore	hsmith@college.edu
J77896	Erica Johnson	3.95	127	Computer Science	Senior	ejohnson@college.edu

Table 2.5 StudentCourse Table

s_course_id	student_id	course_id	credit	major
1000	H10210	CSC-131D	3	CE
1001	B92996	CSC-132A	3	CS/IS
1002	J77896	CSC-335	3	CS/IS
1003	A78835	CSC-331	3	CE
1004	H10210	CSC-234B	3	CE
1005	J77896	CSC-234A	3	CS/IS
1006	B92996	CSC-233A	3	CS/IS
1007	A78835	CSC-132A	3	CE
1008	A78835	CSE-432	3	CE
1009	A78835	CSE-434	3	CE
10010	J77896	CSC-439	3	CS/IS
10011	H10210	CSC-132A	3	CE
10012	H10210	CSC-331	2	CE
10013	A78835	CSC-335	3	CE
10014	A78835	CSE-438	3	CE
10015	J77896	CSC-432	3	CS/IS
10016	A97850	CSC-132B	3	ISE
10017	A97850	CSC-234A	3	ISE
10018	A97850	CSC-331	3	ISE
10019	A97850	CSC-335	3	ISE
10020	J77896	CSE-439	3	CS/IS
10021	B92996	CSC-230	3	CS/IS
10022	A78835	CSE-332	3	CE
10023	B92996	CSE-430	3	CE
10024	J77896	CSC-333A	3	CS/IS
10025	H10210	CSE-433	3	CE
10026	H10210	CSE-334	3	CE
10027	B92996	CSC-131C	3	CS/IS
10028	B92996	CSC-439	3	CS/IS

**Figure 2.1** Records and fields in a table.

2.3.1 Relational Data Model

A data model is like a blueprint for developing a database. It describes the structure of the database and various data relationships and constraints on the data. This information is used in building tables, keys, and defining relationships. Relational model implies that a user perceives the database as made up of relations, a database jargon for tables. It is imperative that all data elements in the tables be represented correctly. In order to achieve these goals designers use various tools. The most commonly used tool is the Entity-Relationship Model (ER). A well-planned model will give consistent results and will allow changes if needed later on. The following section further elaborates on the ER model.

2.3.2 Entity-Relationship Model (ER)

The ER model was first proposed and developed by Peter Chen in 1976. Since then Charles Bachman and James Martin have added some refinements. The model was designed to communicate the database design in the form of a conceptual schema. The ER model is based on the perception that the real world is made up of entities, their attributes, and relationships. The ER model is graphically depicted as Entity-Relationship diagrams (ERD). ERDs are a major modeling tool; they graphically describe the logical structure of the database. ER diagrams can be used with ease to construct the relational tables and are a good vehicle for communicating the database design to the end user or a developer. The three major components of ERD are entities, relationships, and the attributes.

Entities An entity is a data object, either real or abstract, about which we want to collect information. For example, we may want to collect information about a person, a place, or a thing. An entity in an ER diagram translates into a table. It should preferably be referred to as an entity set. Some common examples are departments, courses, students. A single occurrence of an entity is an instance. There are four entities in the CSE_Dept database: LogIn, Faculty, Course, and Student. Each entity is translated into a table with the same name. An instance of the Faculty entity will be Alice Brown and her attributes.

Relationships A database is made up of related entities. There is a natural association between the entities; it is referred to as relationship. For example,

- Students take courses.
- Departments offer certain courses.
- Employees are assigned to departments.

The number of occurrences of one entity associated with single occurrence of a related entity is referred to as *cardinality*.

Attributes Each entity has properties or values called attributes associated with it. The attributes of an entity map into fields in a table. *Database Processing* is one attribute of

an entity called *Courses*. The domain of an attribute is a set of all possible values from which an attribute can derive its value.

2.4 IDENTIFYING KEYS

2.4.1 Primary Key and Entity Integrity

An attribute that uniquely identifies one and only one instance of an entity is called a *primary key*. Sometimes a primary key consists of a combination of attributes. It is referred to as a *composite key*. The *entity integrity rule* states that no attribute that is a member of the primary (composite) key may accept a null value.

A FacultyID may serve as a primary key for the Faculty entity, assuming that all faculty members have been assigned a unique FacultyID. However, caution must be exercised when picking an attribute as a primary key. Last Name may not make a good primary key because a department is likely to have more than one person with the same last name. Primary keys for the CSE_DEPT database are shown in Table 2.6.

Primary keys provide a tuple-level addressing mechanism in the relational databases. Once you define an attribute as a primary key for an entity, the DBMS will enforce the uniqueness of the primary key. Inserting a duplicate value for the primary key field will fail.

2.4.2 Candidate Key

There can be more than one attribute that uniquely identifies an instance of an entity. These are referred to as *candidate keys*. Any one of them can serve as a primary key. For example, ID Number as well as Social Security Number may make a suitable primary key. Candidate keys that are not used as primary key are called *alternate keys*.

2.4.3 Foreign Keys and Referential Integrity

Foreign keys are used to create relationships between tables. It is an attribute in one table whose values are required to match those of primary keys in another table. Foreign

Table 2.6 Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.edu
A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@college.edu
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

Table 2.7 Course (Partial Data Shown)

course_id	course	faculty_id
CSC-132A	Introduction to Programming	J33486
CSC-132B	Introduction to Programming	B78880
CSC-230	Algorithms & Structures	A77587
CSC-232A	Programming I	B66750
CSC-232B	Programming I	A77587
CSC-233A	Introduction to Algorithms	H99118
CSC-233B	Introduction to Algorithms	K69880
CSC-234A	Data Structure & Algorithms	B78880

Faculty (Partial Data Shown)

faculty_id	faculty_name	office
A52990	Black Anderson	MTC-218
A77587	Debby Angles	MTC-320
B66750	Alice Brown	MTC-257
B78880	Ying Bai	MTC-211
B86590	Satish Bhalla	MTC-214
H99118	Jeff Henry	MTC-336
J33486	Steve Johnson	MTC-118
K69880	Jenney King	MTC-324

keys are created to enforce *referential integrity*, which states that you may not add a record to a table containing a foreign key unless there is a corresponding record in the related table to which it is logically linked. Furthermore the referential integrity rule also implies that every value of a foreign key in a table must match the primary key of a related table or be null. MS Access also makes provision for cascade update and cascade delete, which imply that changes made in one of the related tables will be reflected in the other of the two related tables.

Consider two tables Course and Faculty in the sample database CSE_DEPT. The Course table has a foreign key entitled `faculty_id` that is primary key in the Faculty table. The two tables are logically related through the `faculty_id` link. Referential integrity rules imply that we may not add a record to the Course table with a `faculty_id` that is not listed in the Faculty table. In other words there must be a logical link between the two related tables. Second, if we change or delete a `faculty_id` in the Faculty table it must reflect in the Course table meaning that all records in the Course table, must be modified using a cascade update or cascade delete (Tables 2.7).

2.5 DEFINE RELATIONSHIPS

2.5.1 Connectivity

Connectivity refers to the types of relationships that entities can have. Basically, it can be *one-to-one*, *one-to-many*, and *many-to-many*. In ER diagrams these are indicated by placing 1, M, or N at one of the two ends of the relationship diagram. Figure 2.2 illustrates the use of this notation.

- A *one-to-one (1:1)* relationship occurs when one instance of entity A is related to only one instance of entity B, for example, **user_name** in the LogIn table and **user_name** in the Student table (Figure 2.2).
- A *one-to-many (1:M)* relationship occurs when one instance of entity A is associated with zero, one, or many instances of entity B. However, entity B is associated with only one instance of entity A. For example, one department can have many faculty members; each faculty member is assigned to only one department. In the CSE_DEPT database, the one-to-many relationship is represented by **faculty_id** in the Faculty table and

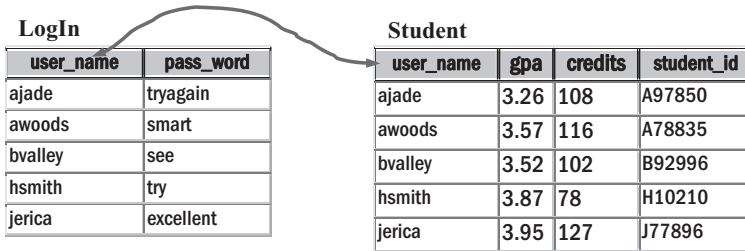


Figure 2.2 One-to-one relationship in the LogIn and the Student tables.

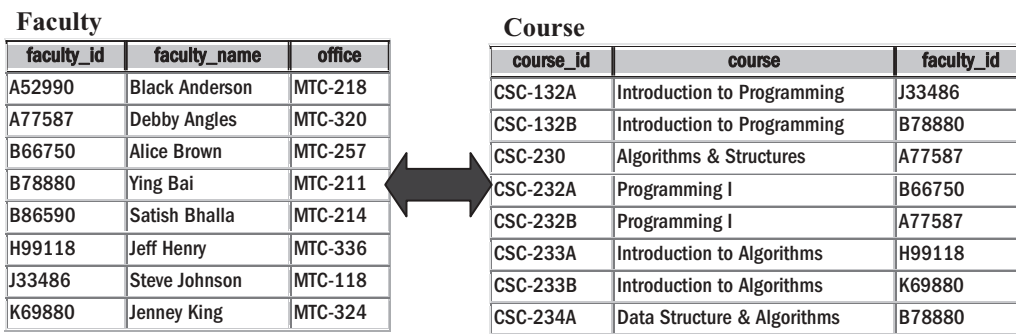


Figure 2.3 One-to-many relationship between Faculty and Course tables.

faculty_id in the Course table, **student_id** in the Student table and **student_id** in the StudentCourse table, **course_id** in the Course table and **course_id** in the StudentCourse table (Figure 2.3).

- A *many-to-many* ($M:N$) relationship occurs when one instance of entity A is associated with zero, one, or many instances of entity B. And one instance of entity B is associated with zero, one, or many instance of entity A. For example, a student may take many courses and one course may be taken by more than one student (Figure 2.4).

In the CSE_DEPT database, a many-to-many relationship can be realized by using the third table. For example, in this case, the StudentCourse that works as the third table, set a many-to-many relationship between the Student and the Course tables.

This database design assumes that the course table only contains courses taught by all faculty members in this department for one semester. Therefore each course can only be taught by a unique faculty. If one wants to develop a Course table that contains courses taught by all faculty in more than one semester, the third table, say FacultyCourse table, should be created to set up a many-to-many relationship between the Faculty and the Course table since one course may be taught by a different faculty member for the different semesters.

The relationships in the CSE_DEPT database are summarized in Figure 2.5.

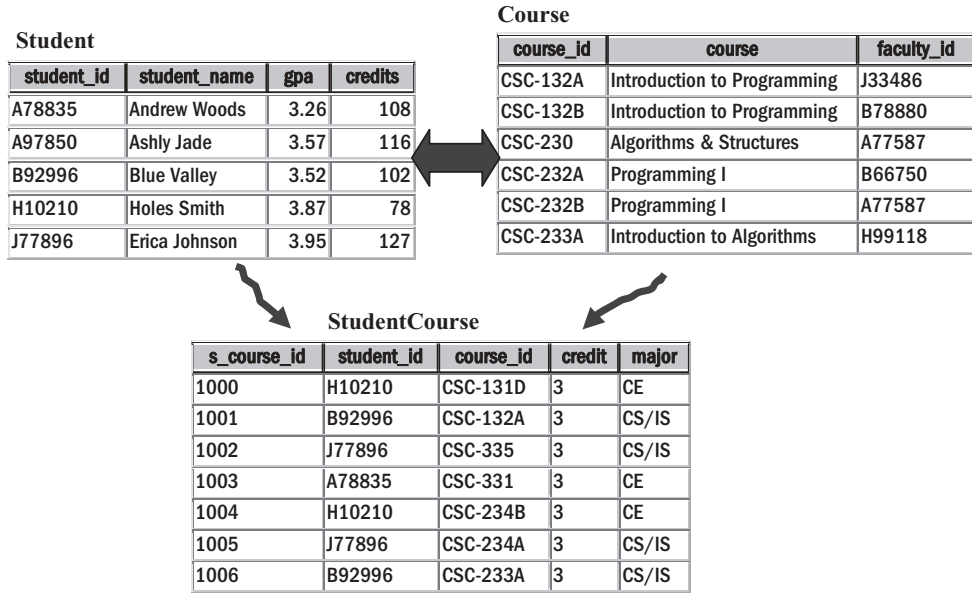


Figure 2.4 Many-to-many relationship between Student and Course tables.

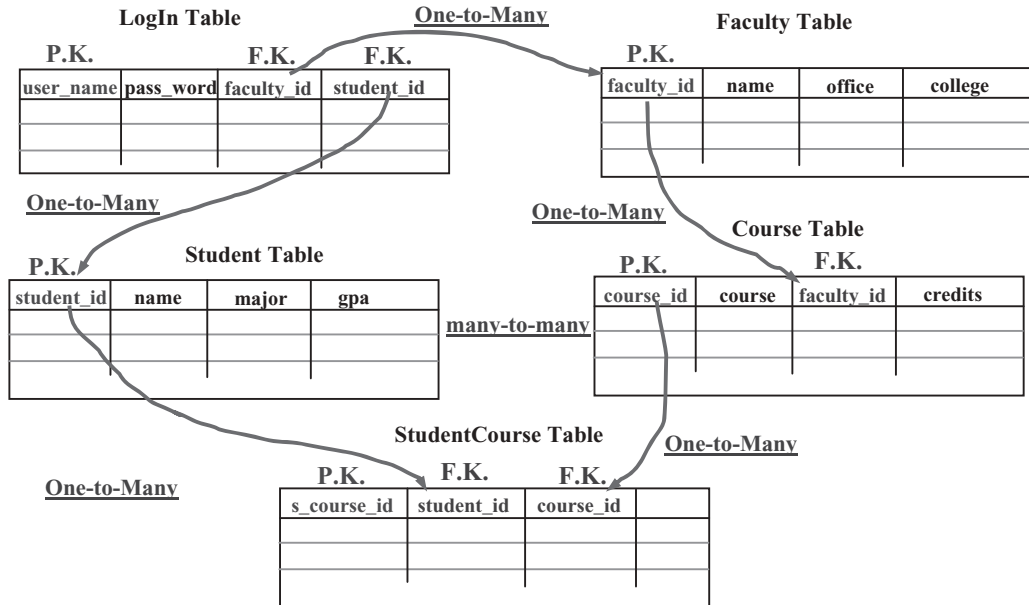


Figure 2.5 Relationships in CSE_DEPT database.

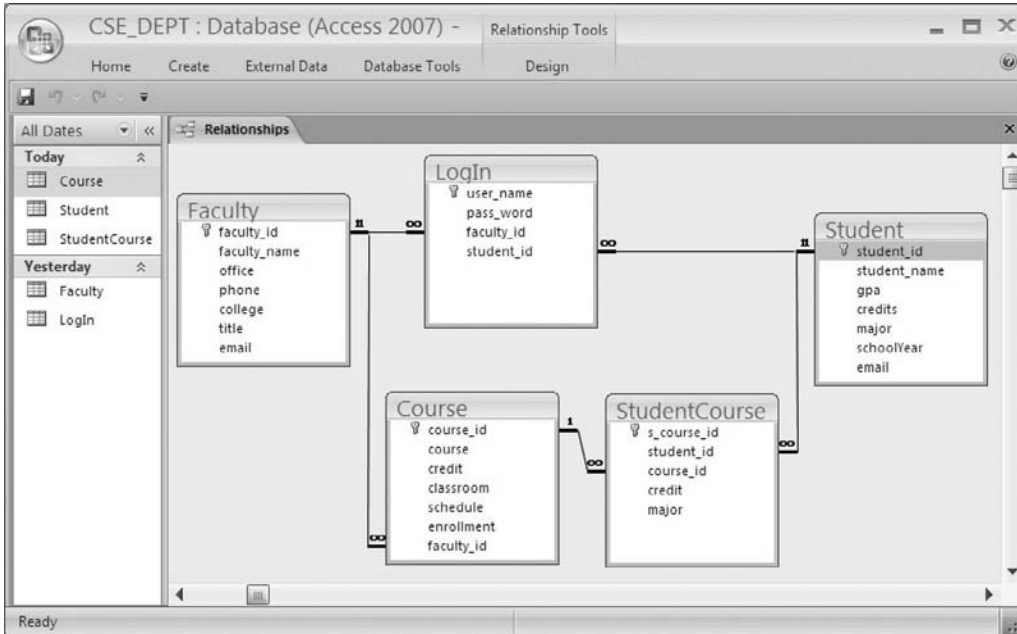


Figure 2.6 Relationships are illustrated using MS Access in the CSE_DEPT database.

Database name: **CSE_DEPT**

Five entities are:

- Login
- Faculty
- Course
- Student
- StudentCourse

The relationships between these entities are shown below. **P.K.** and **F.K.** represent the primary key and the foreign key, respectively.

Figure 2.6 displays the Microsoft Access relationships diagram among various tables in the CSE_Dept database. A one-to-many relationship is indicated by placing 1 at one end of the link and ∞ at the other. The many-to-many relationship between the Student and the Course table was broken down to two one-to-many relationships by creating a new StudentCourse table.

2.6 ER NOTATION

There are a number of ER notations available including Chen's, Bachman's, Crow foot's, and a few others. There is no consensus on the symbols and the styles used to draw ERDs.

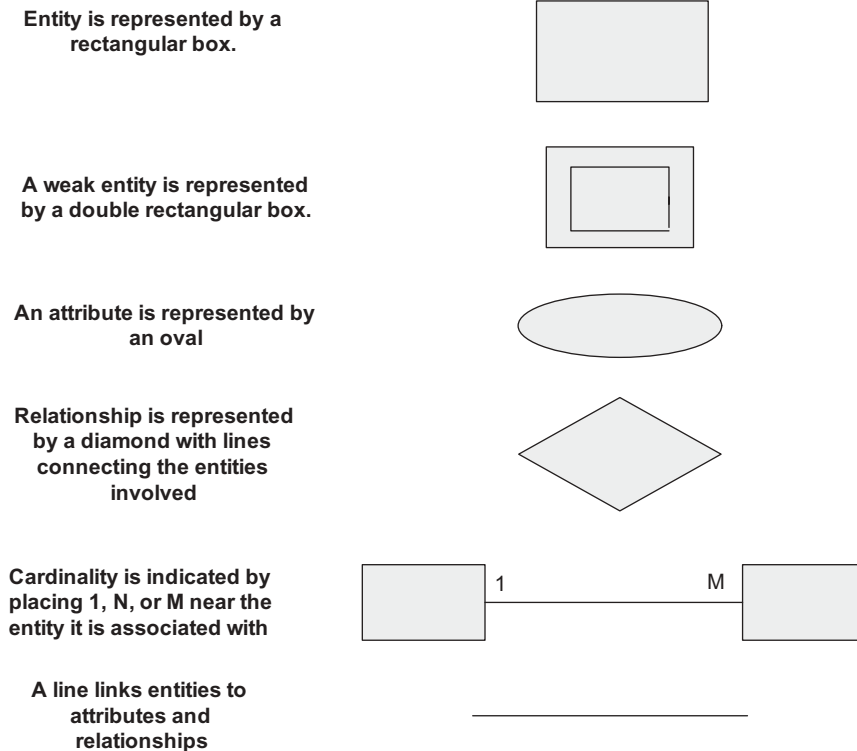


Figure 2.7 Commonly used symbols for ER notation.

A number of drawing tools are available to draw ERDs. These include ER Assistant, Microsoft Vision, and Smart Draw among others. Commonly used notations are shown in Figure 2.7.

2.7 DATA NORMALIZATION

After identifying tables, attributes, and relationships the next logical step in database design is to make sure that the database structure is optimum. Optimum structure is achieved by eliminating redundancies, various inefficiencies, and update and deletion anomalies that usually occur in the unnormalized or partially normalized databases. Data normalization is a progressive process. The steps in the normalization process are called normal forms. Each normal form progressively improves the database and makes it more efficient. In other words a database that is in second normal form is better than the one in the first normal form, and the one in third normal form is better than the one in second normal forms. To be in the third normal form a database has to be in the first and second normal forms. There are fourth and fifth normal forms but for most

practical purposes a database meeting the criteria of third normal form is considered to be of good design.

2.7.1 First Normal Form (1NF)

A table is in first normal form if values in each column are atomic, that is, there are no repeating groups of data. The Faculty table (Table 2.8) is not normalized. Some faculty members have more than one telephone number listed in the phone column. These are called repeating groups.

In order to convert this table to the first normal form (1NF), the data must be atomic. In other words the repeating rows must be broken into two or more atomic rows. Table 2.9 illustrates the Faculty table in 1NF where repeating groups have been removed. Now it is in first normal form.

2.7.2 Second Normal Form (2NF)

A table is in second normal form if it is already in 1NF and every nonkey column is fully dependent upon the primary key. This implies that if the primary key consists of a single

Table 2.8 Unnormalized Faculty Table with Repeating Groups

faculty_id	faculty_name	office	phone
A52990	Black Anderson	MTC-218, SHB-205	750-378-9987, 555-255-8897
A77587	Debby Angles	MTC-320	750-330-2276
B66750	Alice Brown	MTC-257	750-330-6650
B78880	Ying Bai	MTC-211, SHB-105	750-378-1148, 555-246-4582
B86590	Satish Bhalla	MTC-214	750-378-1061
H99118	Jeff Henry	MTC-336	750-330-8650
J33486	Steve Johnson	MTC-118	750-330-1116
K69880	Jenney King	MTC-324	750-378-1230

Table 2.9 Normalized Faculty Table

faculty_id	faculty_name	office	phone
A52990	Black Anderson	MTC-218	750-378-9987
A52990	Black Anderson	SHB-205	555-255-8897
A77587	Debby Angles	MTC-320	750-330-2276
B66750	Alice Brown	MTC-257	750-330-6650
B78880	Ying Bai	MTC-211	750-378-1148
B78880	Ying Bai	SHB-105	555-246-4582
B86590	Satish Bhalla	MTC-214	750-378-1061
H99118	Jeff Henry	MTC-336	750-330-8650
J33486	Steve Johnson	MTC-118	750-330-1116
K69880	Jenney King	MTC-324	750-378-1230

Old Faculty table in 1NF

faculty_id	faculty_name	office	phone
A52990	Black Anderson	MTC-218	750-378-9987
A52990	Black Anderson	SHB-205	555-255-8897
A77587	Debby Angles	MTC-320	750-330-2276
B66750	Alice Brown	MTC-257	750-330-6650
B78880	Ying Bai	MTC-211	750-378-1148
B78880	Ying Bai	SHB-105	555-246-4582
B86590	Satish Bhalla	MTC-214	750-378-1061
H99118	Jeff Henry	MTC-336	750-330-8650
J33486	Steve Johnson	MTC-118	750-330-1116
K69880	Jenney King	MTC-324	750-378-1230



New Faculty table

faculty_id	faculty_name
A52990	Black Anderson
A52990	Black Anderson
A77587	Debby Angles
B66750	Alice Brown
B78880	Ying Bai
B78880	Ying Bai
B86590	Satish Bhalla
H99118	Jeff Henry
J33486	Steve Johnson
K69880	Jenney King

New Office table

office	phone	faculty_id
MTC-218	750-378-9987	A52990
SHB-205	555-255-8897	A52990
MTC-320	750-330-2276	A77587
MTC-257	750-330-6650	B66750
MTC-211	750-378-1148	B78880
SHB-105	555-246-4582	B78880
MTC-214	750-378-1061	B86590
MTC-336	750-330-8650	H99118
MTC-118	750-330-1116	J33486
MTC-324	750-378-1230	K69880

Figure 2.8 Converting Faculty table into 2NF by decomposing the old table in two, Faculty and Office.

column, then the table in 1NF is automatically in 2NF. The second part of the definition implies that if the key is composite, then none of the nonkey columns will depend upon just one of the columns that participates in the composite key.

The Faculty table in Table 2.9 is in first normal form. However, it has a composite primary key, made up of `faculty_id` and `office`. The phone number depends on a part of the primary key, the office, and not on the whole primary key. This can lead to update and deletion anomalies mentioned above.

By splitting the old Faculty table (Figure 2.8) into two new tables, Faculty and Office, we can remove the dependencies mentioned earlier. Now the faculty table has a primary key, `faculty_id` and the Office table has a primary key, `office`. The nonkey columns in both tables now depend only on the primary keys.

2.7.3 Third Normal Form (3NF)

A table is in third normal form if it is already in 2NF and every nonkey column is nontransitively dependent upon the primary key. In other words all nonkey columns are mutually independent, but at the same time they are fully dependent upon the primary key only.

Another way of stating this is that in order to achieve 3NF no column should depend upon any nonkey column. If column B depends on column A, then A is said to functionally determine column B; hence the term *determinant*. Another definition of 3NF says that the table should be in 2NF and only determinants it contains are candidate keys.

In the customer table in Figure 2.8, all nonkey columns depend on the primary key—the *course_id*. In addition, name and phone columns also depend on *faculty_id*. This table is in second normal form, but it suffers from update, addition, and deletion anomalies because of transitive dependencies. In order to conform to third normal form we can split this table into two tables, Course and Instructor (Tables 2.11 and 2.12). Now we have eliminated the transitive dependencies that are apparent in the Course table in Table 2.10.

Table 2.10 The Old Course Table

course_id	course	classroom	faculty_id	faculty_name	phone
CSC-131A	Computers in Society	TC-109	A52990	Black Anderson	750-378-9987
CSC-131B	Computers in Society	TC-114	B66750	Alice Brown	750-330-6650
CSC-131C	Computers in Society	TC-109	A52990	Black Anderson	750-378-9987
CSC-131D	Computers in Society	TC-109	B86590	Satish Bhalla	750-378-1061
CSC-131E	Computers in Society	TC-301	B66750	Alice Brown	750-330-6650
CSC-131I	Computers in Society	TC-109	A52990	Black Anderson	750-378-9987
CSC-132A	Introduction to Programming	TC-303	J33486	Steve Johnson	750-330-1116
CSC-132B	Introduction to Programming	TC-302	B78880	Ying Bai	750-378-1148

Table 2.11 The New Course Table

course_id	course	classroom
CSC-131A	Computers in Society	TC-109
CSC-131B	Computers in Society	TC-114
CSC-131C	Computers in Society	TC-109
CSC-131D	Computers in Society	TC-109
CSC-131E	Computers in Society	TC-301
CSC-131I	Computers in Society	TC-109
CSC-132A	Introduction to Programming	TC-303
CSC-132B	Introduction to Programming	TC-302

Table 2.12 The New Instructor Table

faculty_id	faculty_name	phone
A52990	Black Anderson	750-378-9987
B66750	Alice Brown	750-330-6650
A52990	Black Anderson	750-378-9987
B86590	Satish Bhalla	750-378-1061
B66750	Alice Brown	750-330-6650
A52990	Black Anderson	750-378-9987
J33486	Steve Johnson	750-330-1116
B78880	Ying Bai	750-378-1148
A77587	Debby Angles	750-330-2276

2.8 DATABASE COMPONENTS IN SOME POPULAR DATABASES

All databases allow for storage, retrieval, and management of the data. Simple databases provide basic services to accomplish these tasks. Many database providers, such as Microsoft SQL Server and Oracle, provide additional services that necessitate storing many components in the database other than data. These components such as views and stored procedures are collectively called database objects. In this section we will discuss various objects that make up MS Access, SQL Server, and Oracle databases.

There are two major types of databases, *File Server* and *Client Server*: In a File Server database, data is stored in a file, and each user of the database retrieves the data, displays the data, or modifies the data directly from or to the file. In a Client Server database the data is also stored in a file; however, all these operations are mediated through a master program called a server. MS Access is a File Server database, whereas Microsoft SQL Server and Oracle are Client Server databases. The Client Server databases have several advantages over the File Server databases. These include minimizing chances of crashes, provision of features for recovery, enforcement of security, better performance, and more efficient use of the network compared to the File Server databases.

2.8.1 Microsoft Access Databases

Microsoft Access Database Engine is a collection of information stored in a systematic way that forms the underlying component of a database. Also called a Jet (Joint Engine Technology), it allows the manipulation of relational databases. It offers a single interface that other software may use to access Microsoft databases. The supporting software is developed to provide security, integrity, indexing, record locking, and the like. By executing the MS Access program, MSACCESS.EXE, you can see the database engine at work and the user interface it provides. Figure 2.9 shows how a visual basic application accesses the MS Access database via Jet 4.0 OLE database provider.

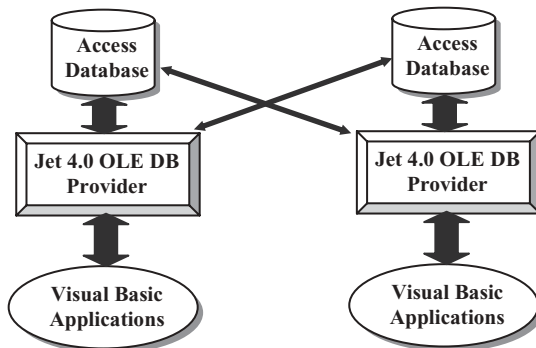


Figure 2.9 Microsoft Access database illustration.

2.8.1.1 Database File

Access database is made up of a number of components called objects that are stored in a single file referred to as *database file*. As new objects are created or more data is added to the database, this file gets bigger. This is a complex file that stores objects like tables, queries, forms, reports, macros, and modules. The Access files have a .mdb (Microsoft DataBase) extension. Some of these objects help users to work with the database; others are useful for displaying database information in a comprehensible and easy to read format.

2.8.1.2 Tables

Before you can create a table in Access, you must create a database container and give it a name with the extension .mdb. Database creation is a simple process and is explained in detail with an example, later in this chapter. Suffice it to say that a table is made up of columns and rows. Columns are referred to as fields, which are attributes of an entity. Rows are referred to as records also called tuples.

2.8.1.3 Queries

One of the main purposes of storing data in a database is that the data may be retrieved later as needed, without having to write complex programs. This purpose is accomplished in Access and other databases by writing SQL statements. A group of such statements is called a query. It enables you to retrieve, update, and display data in the tables. You may display data from more than one table by using a Join operation. In addition you may insert or delete data in the tables.

Access also provides a visual graphic user interface (GUI) to create queries. This bypasses writing SQL statements and makes it appealing to beginning and not so savvy users, who can use wizards or GUIs to create queries. Queries can extract information in a variety of ways. You can make them as simple or as complex as you like. You may specify various criteria to get desired information, perform comparisons, or you may want to perform some calculations and obtain the results. In essence, operators, functions, and expressions are the building blocks for Access operation.

2.8.2 SQL Server Databases

The Microsoft SQL Server database engine is a service for storing and processing data in either a relational (tabular) format or as extensible markup language (XML) documents. Various tasks performed by the database engine include:

- Designing and creating a database to hold the relational tables or XML documents
- Accessing and modifying the data stored in the database
- Implementing websites and applications
- Building procedures
- Optimizing the performance of the database

The SQL Server database is a complex entity, made up of multiple components. It is more complex than MS Access database, which can be simply copied and distributed. Certain procedures have to be followed for copying and distributing an SQL Server database.

The SQL Server is used by a diverse group of professionals with diverse needs and requirements. To satisfy different needs, SQL Server comes in five editions: Enterprise edition, Standard edition, Workgroup edition, Developer edition, and Express edition. The most common editions are Enterprise, Standard, and Workgroup. It is noteworthy that the database engine is virtually the same in all of these editions.

The SQL Server database can be stored on the disk using three types of files—primary data files, secondary data files, and transaction log files. Primary data files are created first and contain user-defined objects, such as tables and views, and system objects. These files have extensions of .mdf. If the database grows too big for a disk, it can be stored as secondary files with an extension .ndf. The SQL Server still treats these files as if they are together. The data file is made up of many objects. The transaction log files carry .ldf extension. All transactions to the database are recorded in this file.

Figure 2.10 illustrates the structure of the SQL Server database. Each visual C# application has to access the server, which in turn accesses the SQL database.

2.8.2.1 Data Files

A data file is a conglomeration of objects that includes tables, keys, views, stored procedures, and others. All these objects are necessary for the efficient operation of the database.

2.8.2.2 Tables

The data in a relational database resides in tables. These are the building blocks of the database. Each table consists of columns and rows. Columns represent various attributes or fields in a table. Each row represents one record. For example, one record in the

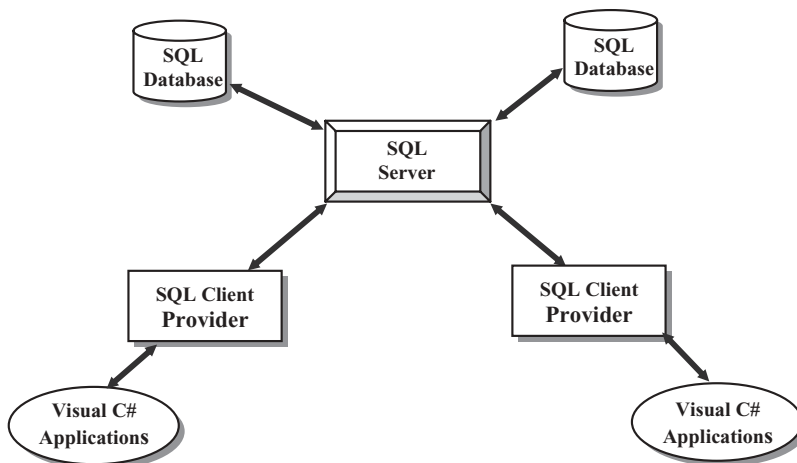


Figure 2.10 SQL Server database structure.

Faculty table consists of name, office, phone, college, title, and email. Each field has a distinct data type, meaning that it can contain only one type of data such as numeric or character. Tables are the first objects created in a database.

2.8.2.3 Views

Views are virtual tables, meaning that they do not contain any data. They are stored as queries in the database, which are executed when needed. A view can contain data from one or more tables. The views can provide database security. Sensitive information in a database can be excluded by including nonsensitive information in a view and providing user access to the views instead of all tables in a database. The views can also hide the complexities of a database. A user can be using a view that is made up of multiple tables, whereas it appears as a single table to the user. The user can execute queries against a view just like a table.

2.8.2.4 Stored Procedures

Users write queries to retrieve, display, or manipulate data in the database. These queries can be stored on the client machine or on the server. There are advantages associated with storing SQL queries on the server rather than on the client machine. It has to do with the network performance. Usually users use the same queries over and over again, frequently different users are trying to access the same data. Instead of sending the same queries on the network repeatedly, it improves the network performance and executes queries faster if the queries are stored on the server where they are compiled and saved as stored procedures. The users can simply call the stored procedure with a simple command like *execute stored_procedure A*.

2.8.2.5 Keys and Relationships

A *primary key* is created for each table in the database to efficiently access records and to ensure *entity integrity*. This implies that each record in a table is unique in some way. Therefore, no two records can have the same primary key. It is defined as a globally unique identifier. Moreover, a primary key may not have a null value, that is, missing data. The SQL Server creates a unique index for each primary key. This ensures fast and efficient access to data. One or columns can be combined to designate a primary key.

In a relational database, relationships between tables can be logically defined with the help of *foreign keys*. A foreign key of one record in a table points specifically to a primary key of a record in another table. This allows a user to join multiple tables and retrieve information from more than one table at a time. Foreign keys also enforce *referential integrity*, a defined relationship between the tables that does not allow insertion or deletion of records in a table unless the foreign key of a record in one table matches a primary key of a record in another table. In other words, a record in one table cannot have a foreign key that does not point to a primary key in another table. Additionally, a primary key may not be deleted if there are foreign keys in another table pointing to it. The foreign key values associated with a primary key must be deleted first. Referential integrity protects related data from corruption stored in different tables.

2.8.2.6 Indexes

The indexes are used to find records, quickly and efficiently, in a table just like one would use an index in a book. SQL Server uses two types of indexes to retrieve and update data—clustered and nonclustered.

The *clustered index* sorts the data in a table so that the data can be accessed efficiently. It is akin to a dictionary or a phone book where records are arranged alphabetically. Thus one can go directly to a specific alphabet and from there search sequentially for the specific record. The clustered indexes are like an inverted tree. The index structure is called a B tree for binary tree. You start with the root page at the top and find the location of other pages further down at the secondary level, following to the tertiary level, and so on until you find the desired record. The very bottom pages are the leaf pages and contain the actual data. There can be only one clustered index per table because clustered indexes physically rearrange the data.

Nonclustered indexes do not physically rearrange the data as do the clustered indexes. They also consist of a binary tree with various levels of pages. The major difference, however, is that the leaves do not contain the actual data as in the clustered indexes, instead they contain pointers that point to the corresponding records in the table. These pointers are called row locators.

The indexes can be unique where the duplicate keys are not allowed or not unique, which permits duplicate keys. Any column that can be used to access data can be used to generate an index. Usually the primary and the foreign key columns are used to create indexes.

2.8.2.7 Transaction Log Files

A transaction is a logical group of SQL statements that carry out a unit of work. Client server databases use log files to keep track of transactions that are applied to the database. For example, before an update is applied to a database, the database server creates an entry in the transaction log to generate a before picture of the data in a table and then applies a transaction and creates another entry to generate an after picture of the data in that table. This keeps track of all the operations performed on a database. Transaction logs can be used to recover data in case of crashes or disasters. Transaction logs are automatically maintained by the SQL Server.

2.8.3 Oracle Databases

Oracle was designed to be platform independent, making it architecturally more complex than the SQL Server database. The Oracle database contains more files than the SQL Server database.

The Oracle DBMS comes in three levels: Enterprise, Standard, and Personal. The Enterprise edition is the most powerful and is suitable for a large installations using a large number of transactions in a multiuser environment. The Standard edition is also used by high-level multi-user installations. It lacks some of the utilities available in the Enterprise edition. The Personal edition is used in a single-user environment for developing database applications. The database engine components are virtually the same for all three editions.

Oracle architecture is made up of several components including an Oracle server, Oracle instance, and an Oracle database. The Oracle server contains several files, processes, and memory structures. Some of these are used to improve the performance of the database and ensure database recovery in case of a crash. The Oracle server consists of an Oracle instance and an Oracle database. An Oracle instance consists of background processes and memory structures. Background processes perform input/output and monitor other Oracle processes for better performance and reliability. Oracle database consists of data files that provide the actual physical storage for the data.

2.8.3.1 Data Files

The main purpose of a database is to store and retrieve data. It consists of a collection of data that is treated as a unit. An Oracle database has a logical and physical structure. The logical layer consists of tablespaces, necessary for the smooth operation of an Oracle installation. Data files make up the physical layer of the database. These consist of three types of files: *data files*, which contain actual data in the database; *redo logfiles*, which contain records of modifications made to the database for future recovery in case of failure; and *control files*, which are used to maintain and verify database integrity. Oracle server uses other files that are not part of the database. These include *parameter file*, which defines the characteristics of an Oracle instance, *password file* used for authentication, and *archived redo log* files, which are copies of the redo log files necessary for recovery from failure. A partial list of some of the components follows.

2.8.3.2 Tables

Users can store data in a regular table, partitioned table, index-organized table, or clustered table. A *regular table* is the default table as in other databases. Rows can be stored in any order. A *partitioned table* has one or more partitions where rows are stored. Partitions are useful for large tables that can be queried by several processes concurrently. *Index-organized tables* provide fast key-based access for queries involving exact matches. The table may have index on one or more of its columns. Instead of using two storage spaces for the table and a B-tree index, a single storage space is used to store both the B tree and other columns. A *clustered table* or group of tables share the same block called a cluster. They are grouped together because they share common columns and are frequently used together. Clusters have a cluster key for identifying the rows that need to be stored together. Cluster keys are independent of the primary key and may be made up of one or more columns. Clusters are created to improve performance.

2.8.3.3 Views

Views are like virtual tables and are used in a similar fashion as in the SQL Server databases discussed above.

2.8.3.4 Stored Procedures

In Oracle, functions and procedures may be saved as stored program units. Multiple input arguments (parameters) may be passed as input to functions and procedures; however,

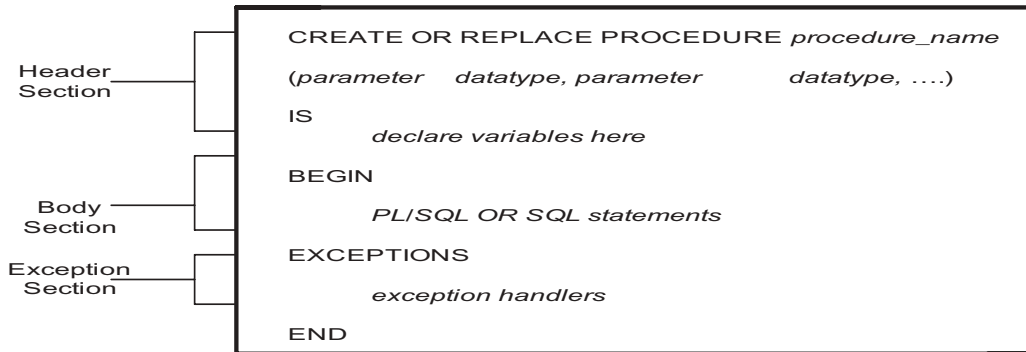


Figure 2.11 Syntax for creating a stored procedure in Oracle.

functions return only one value as output, whereas procedures may return multiple values as output. The advantages to creating and using stored procedures are the same as mentioned above for SQL Server. By storing procedures on the server, individual SQL statements do not have to be transmitted over the network, thus reducing the network traffic. In addition commonly used SQL statements are saved as functions or procedures and may be used again and again by various users, thus saving rewriting the same code over and over again. The stored procedures should be made flexible so that different users are able to pass input information to the procedure in the form of arguments or parameters and get the desired output.

Figure 2.11 shows the syntax to create a stored procedure in Oracle. It has three sections—a header, a body, and an exception section. The procedure is defined in the header section. Input and output parameters, along with their data types, are declared here and transmit information to or from the procedure. The body section of the procedure starts with a keyword `BEGIN` and consists of SQL statements. The exceptions section of the procedure begins with the keyword `EXCEPTION` and contains exception handlers that are designed to handle the occurrence of some conditions that change the normal flow of execution.

2.8.3.5 Indexes

Indexes are created to provide direct access to rows. An index is a tree structure. Indexes can be classified on their logic design or their physical implementation. Logical classification is based on application perspective, whereas physical classification is based on how the indexes are stored. Indexes can be partitioned or nonpartitioned. Large tables use partitioned indexes, which spreads an index to multiple table spaces thus decreasing contention for index lookup and increasing manageability. An index may consist of a single column or multiple columns; it may be unique or nonunique. Some of these indexes are outlined below.

Function-based indexes precompute the value of a function or expression of one or more columns and store it in an index. It can be created as a B tree or as a bit map. It can improve the performance of queries performed on tables that rarely change.

Domain indexes are application specific and are created and managed by the user or applications. Single-column indexes can be built on text, spatial, scalar, object, or LOB data types.

B-tree indexes store a list of row IDs for each key. The structure of a B-tree index is similar to the ones in the SQL Server described above. The leaf nodes contain indexes that point to rows in a table. The leaf blocks allow scanning of the index in either ascending or descending order. Oracle server maintains all indexes when insert, update, or delete operations are performed on a table.

Bitmap indexes are useful when columns have low cardinality and a large number of rows. For example, a column may contain few distinct values like Y/N for marital status or M/F for gender. A bitmap is organized like a B tree where the leaf nodes store a bitmap instead of row IDs. When changes are made to the key columns, bitmaps must be modified.

2.8.3.6 Initialization Parameter Files

Oracle server must read the initialization parameter file before starting an Oracle database instance. There are two types of initialization parameter files: a static parameter file and a persistent parameter file. An initialization parameter file contains a list of instance parameters, the name of the database with which the instance is associated, name and location of control files, and information about the undo segments. Multiple initialization parameter files can exist to optimize performance.

2.8.3.7 Control Files

A control file is a small binary file that defines the current state of the database. Before a database can be opened a control file is read to determine if the database is in a valid state or not. It maintains the integrity of the database. Oracle uses a single control file per database. It is maintained continuously by the server and can be maintained only by the Oracle server. It cannot be edited by a user or database administrator. A control file contains: database name and identifier, time stamp of database creation, tablespace name, names and location of data files and redo log files, current log files sequence number, and archive and backup information.

2.8.3.8 Redo Log Files

Oracle's redo log files provides a way to recover data in the event of a database failure. All transactions are written to a redo log buffer and passed on to the redo log files.

Redo log files record all changes to the data, provide a recovery mechanism, and can be organized into groups. A set of identical copies of online redo log files is called a redo log file group. The Oracle server needs a minimum of two online redo log-file groups for normal operations. The initial set of redo log-file groups and members are created during the database creation. Redo log files are used in a cyclic fashion. Each redo log-file group is identified by a log sequence number and is overwritten each time the log is reused. In other words, when a redo log file is full, then the log writer moves to the second redo log file. After the second one is full the first one is reused.

2.8.3.9 Password Files

Depending upon whether the database is administered locally or remotely, one can choose either operating system or password file authentication to authenticate database

administrators. Oracle provides a password utility to create password files. Administrators use the GRANT command to provide access to the database using the password file.

2.9 CREATE MICROSOFT ACCESS SAMPLE DATABASE

In this section, you will learn how to create a sample Microsoft Access database CSE_DEPT.accdb and its database file. As we mentioned in the previous sections, Access is a file-based database system, which means that the database is composed of a set of data tables that are represented in the form of files.

Open the Microsoft Office Access 2007. Select Blank Database item and enter **CSE_DEPT** into the File Name box as the database name and keep the extension **accdb** unchanged. Click on the small file-folder icon that is next to the File Name box to open the File New Database dialog to select the desired destination to save this new database. In our case, select the C:\Database and then click the OK button. Now click on the Create button to create this new database.

2.9.1 Create Login Table

After a new blank database is created, click on the drop-down arrow of the View button from the Toolbar, and select the Design View item to open the database in the design view. Enter **Login** into the Table Name box of the pop-up dialog as the name of our first table, Login. Click on OK to open this table in the design view. Enter the following data, which are shown in Figure 2.12, into this design view to build our Login table.



Starting from Office 2007, Microsoft released a new Access database format, **accdb**, which is different from old formats and contains quite a few new functionalities that the old Access formats do not have, such as allowing you to store file attachments as part of your database files, use multivalued fields, integrate with SharePoint and Outlook and perform encryption improvements. You can convert the old format such as Access 2000, Access 2002–2003 with the **.mdb** extension to this new format with the extension **.accdb** if you like.

Three columns are displayed in this Design view: Field Name, Data Type, and Description. The first table you want to create is the Login table with four columns: `user_name`, `pass_word`, `faculty_id`, and `student_id`. Enter `user_name` into the first Field Name box. The data type for this `user_name` should be Text, so click the drop-down arrow of the Data Type box and select Text. You can enter some comments in the Description box to indicate the purpose of this data. In this case, just enter: primary key for the Login table since you need this column as the primary key for this table.

In a similar way, enter the `pass_word`, `faculty_id`, and `student_id` into the second, third, and fourth fields with the data type as Text for those fields. Now you need to assign



Figure 2.12 Design view of the LogIn table.

the `user_name` column as the primary key for this table. In the previous versions of Microsoft Office Access, such as Office 2003 or XP, you needed to click and select the first row `user_name` from the table, and then go to the Toolbar and select the primary key tool that is displayed as a key. But starting from Office 2007, you do not need to do that since the first column has been selected as the primary key by default, which is represented as a key sign and is shown in Figure 2.12.

Click on the Save button on the Toolbar to save the design for this table. Your finished Design view of the LogIn table should match the one shown in Figure 2.12.

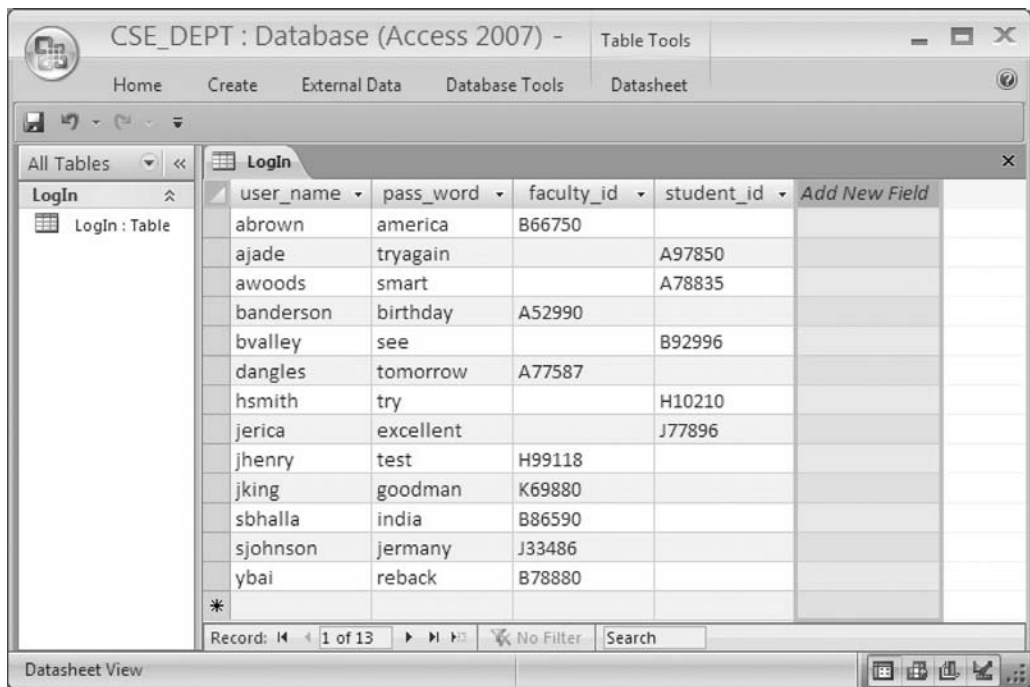
Next you need to add the data into this LogIn table. To do that, you need to open the DataSheet view of the table. You can open this view by clicking the drop-down arrow of the View tool on the Toolbar, which is the first tool located on the Toolbar, then select the DataSheet view.

Four data columns, `user_name`, `pass_word`, `faculty_id`, and `student_id`, are displayed when the DataSheet view of this LogIn table is opened. Enter the data shown in Table 2.13 into this table. Your finished LogIn table is shown in Figure 2.13.

Your finished LogIn table should match the one shown in Figure 2.13. Click on the Save button on the Toolbar to save this table. Then click on the Close button located on the upper-right corner of the table to close this LogIn table.

Table 2.13 Data in the LogIn Table

user_name	pass_word	faculty_id	student_id
abrown	america	B66750	
ajade	tryagain		A97850
awoods	smart		A78835
banderson	birthday	A52990	
bvalley	see		B92996
dangles	tomorrow	A77587	
hsmith	try		H10210
jerica	excellent		J77896
jhenry	test	H99118	
jking	goodman	K69880	
sbhalla	india	B86590	
sjohnson	jermany	J33486	
ybai	reback	B78880	

**Figure 2.13** Completed LogIn table.

2.9.2 Create Faculty Table

Now let's continue to create the second table Faculty. Click on the Create menu item from the menu bar and select the Table icon from the Toolbar to create a new table. Click on the Home menu item and select the DesignView by clicking on the drop-down

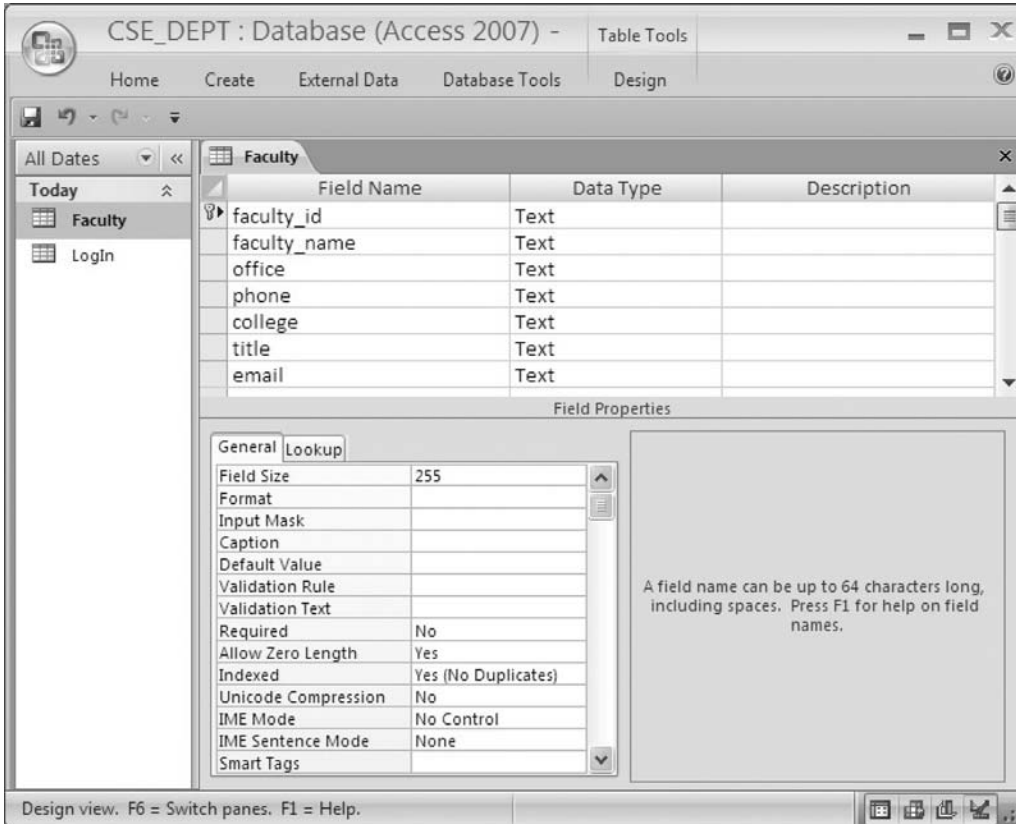


Figure 2.14 Design view of the Faculty table.

arrow from the View tool on the Toolbar. Enter **Faculty** into the TableName box of the pop-up dialog as the name for this new table, and click on OK.

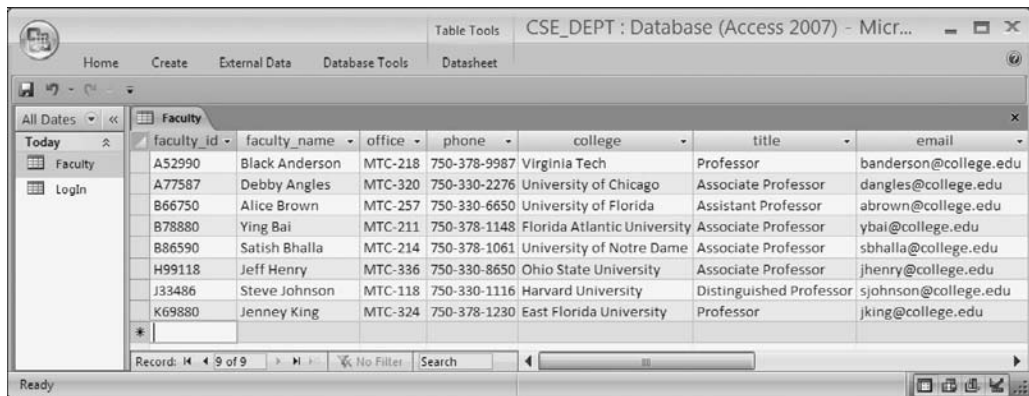
Seven columns are included in this table, they are: faculty_id, faculty_name, office, phone, college, title, and email. The data types for all columns in this table are Text since all of them are string variables. You can redefine the length of each Text string by modifying the FieldSize in the FieldProperties pane located below of the table, which is shown in Figure 2.14. The default length for each text string is 255.

Now you need to assign the primary key for this table. As we discussed in the last section, you do not need to do this in Office 2007 Access since the first column, faculty_id, has been selected as the primary key by default. Click on the Save tool on the Toolbar to save this table. The finished DesignView of the Faculty table is shown in Figure 2.14.

Now open the DataSheet view of the Faculty table by clicking on the Home menu item, and then the drop-down arrow of the View tool, and select the DataSheet View item. Enter the data that is shown in Table 2.14 into this opened Faculty table. The finished Faculty table should match the one is shown in Figure 2.15.

Table 2.14 Data in the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.edu
A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@college.edu
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

**Figure 2.15** Completed Faculty table.

2.9.3 Create Other Tables

In a similar way, you need to create the following three tables: Course, Student, and StudentCourse. Select the `course_id`, `student_id`, and `s_course_id` columns as the primary key for the Course, Student, and StudentCourse tables (refer to Tables 2.15, 2.16, and 2.17). For the data type selections, follow the directions below:

The data type selections for the Course table:

- `course_id`—Text
- `credit`—Number
- `enrollment`—Number
- all other columns—Text

The data type selections for the Student table:

- `student_id`—Text
- `credits`—Number
- all other columns—Text

The data type selections for the StudentCourse table:

Table 2.15 Data in the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-131A	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	28	A52990
CSC-131B	Computers in Society	3	TC-114	M-W-F: 9:00-9:55 AM	20	B66750
CSC-131C	Computers in Society	3	TC-109	T-H: 11:00-12:25 PM	25	A52990
CSC-131D	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	30	B86590
CSC-131E	Computers in Society	3	TC-301	M-W-F: 1:00-1:55 PM	25	B66750
CSC-131I	Computers in Society	3	TC-109	T-H: 1:00-2:25 PM	32	A52990
CSC-132A	Introduction to Programming	3	TC-303	M-W-F: 9:00-9:55 AM	21	J33486
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-230	Algorithms & Structures	3	TC-301	M-W-F: 1:00-1:55 PM	20	A77587
CSC-232A	Programming I	3	TC-305	T-H: 11:00-12:25 PM	28	B66750
CSC-232B	Programming I	3	TC-303	T-H: 11:00-12:25 PM	17	A77587
CSC-233A	Introduction to Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	18	H99118
CSC-233B	Introduction to Algorithms	3	TC-302	M-W-F: 11:00-11:55 AM	19	K69880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSC-234B	Data Structure & Algorithms	3	TC-114	T-H: 11:00-12:25 PM	15	J33486
CSC-242	Programming II	3	TC-303	T-H: 1:00-2:25 PM	18	A52990
CSC-320	Object Oriented Programming	3	TC-301	T-H: 1:00-2:25 PM	22	B66750
CSC-331	Applications Programming	3	TC-109	T-H: 11:00-12:25 PM	28	H99118
CSC-333A	Computer Arch & Algorithms	3	TC-301	M-W-F: 10:00-10:55 AM	22	A77587
CSC-333B	Computer Arch & Algorithms	3	TC-302	T-H: 11:00-12:25 PM	15	A77587
CSC-335	Internet Programming	3	TC-303	M-W-F: 1:00-1:55 PM	25	B66750
CSC-432	Discrete Algorithms	3	TC-206	T-H: 11:00-12:25 PM	20	B86590
CSC-439	Database Systems	3	TC-206	M-W-F: 1:00-1:55 PM	18	B86590
CSE-138A	Introduction to CSE	3	TC-301	T-H: 1:00-2:25 PM	15	A52990
CSE-138B	Introduction to CSE	3	TC-109	T-H: 1:00-2:25 PM	35	J33486
CSE-330	Digital Logic Circuits	3	TC-305	M-W-F: 9:00-9:55 AM	26	K69880
CSE-332	Foundations of Semiconductors	3	TC-305	T-H: 1:00-2:25 PM	24	K69880
CSE-334	Elec Measurement & Design	3	TC-212	T-H: 11:00-12:25 PM	25	H99118
CSE-430	Bioinformatics in Computer	3	TC-206	Thu: 9:30-11:00 AM	16	B86590
CSE-432	Analog Circuits Design	3	TC-309	M-W-F: 2:00-2:55 PM	18	K69880
CSE-433	Digital Signal Processing	3	TC-206	T-H: 2:00-3:25 PM	18	H99118
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-436	Automatic Control and Design	3	TC-305	M-W-F: 10:00-10:55 AM	29	J33486
CSE-437	Operating Systems	3	TC-303	T-H: 1:00-2:25 PM	17	A77587
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880
CSE-439	Special Topics in CSE	3	TC-206	M-W-F: 10:00-10:55 AM	22	J33486

- s_course_id—Number
- credit—Number
- all other columns—Text

Enter the data shown in Tables 2.15, 2.16, and 2.17 into each associated table, and save each table as Course, Student, and StudentCourse, respectively.

The finished Course table is shown in Figure 2.16. The completed Student and StudentCourse tables are shown in Figures 2.17 and 2.18.

Table 2.16 Data in the Student Table

student_id	student_name	gpa	credits	major	schoolYear	email
A78835	Andrew Woods	3.26	108	Computer Science	Senior	awoods@college.edu
A97850	Ashly Jade	3.57	116	Information System Engineering	Junior	ajade@college.edu
B92996	Blue Valley	3.52	102	Computer Science	Senior	bvalley@college.edu
H10210	Holes Smith	3.87	78	Computer Engineering	Sophomore	hsmith@college.edu
J77896	Erica Johnson	3.95	127	Computer Science	Senior	ejohnson@college.edu

Table 2.17 Data in the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1000	H10210	CSC-131D	3	CE
1001	B92996	CSC-132A	3	CS/IS
1002	J77896	CSC-335	3	CS/IS
1003	A78835	CSC-331	3	CE
1004	H10210	CSC-234B	3	CE
1005	J77896	CSC-234A	3	CS/IS
1006	B92996	CSC-233A	3	CS/IS
1007	A78835	CSC-132A	3	CE
1008	A78835	CSE-432	3	CE
1009	A78835	CSE-434	3	CE
1010	J77896	CSC-439	3	CS/IS
1011	H10210	CSC-132A	3	CE
1012	H10210	CSC-331	3	CE
1013	A78835	CSC-335	3	CE
1014	A78835	CSE-438	3	CE
1015	J77896	CSC-432	3	CS/IS
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE
1018	A97850	CSC-331	3	ISE
1019	A97850	CSC-335	3	ISE
1020	J77896	CSE-439	3	CS/IS
1021	B92996	CSC-230	3	CS/IS
1022	A78835	CSE-332	3	CE
1023	B92996	CSE-430	3	CE
1024	J77896	CSC-333A	3	CS/IS
1025	H10210	CSE-433	3	CE
1026	H10210	CSE-334	3	CE
1027	B92996	CSC-131C	3	CS/IS
1028	B92996	CSC-439	3	CS/IS

2.9.4 Create Relationships Among Tables

All five tables are completed and now we need to set up the relationships between these five tables by using the primary and foreign keys. Go to the Database Tools|Relationships menu item to open the Show Table dialog. Keep the default tab Tables selected, and select all five tables by pressing and holding the Shift key on the keyboard and clicking

course_id	course	cred	classroom	schedule	enrollment	faculty_id
CSC-131A	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	28	A52990
CSC-131B	Computers in Society	3	TC-114	M-W-F: 9:00-9:55 AM	20	B66750
CSC-131C	Computers in Society	3	TC-109	T-H: 11:00-12:25 PM	25	A52990
CSC-131D	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	30	B86590
CSC-131E	Computers in Society	3	TC-301	M-W-F: 1:00-1:55 PM	25	B66750
CSC-131I	Computers in Society	3	TC-109	T-H: 1:00-2:25 PM	32	A52990
CSC-132A	Introduction to Programming	3	TC-303	M-W-F: 9:00-9:55 AM	21	J33486
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-230	Algorithms & Structures	3	TC-301	M-W-F: 1:00-1:55 PM	20	A77587
CSC-232A	Programming I	3	TC-305	T-H: 11:00-12:25 PM	28	B66750
CSC-232B	Programming I	3	TC-303	T-H: 11:00-12:25 PM	17	A77587
CSC-233A	Introduction to Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	18	H99118
CSC-233B	Introduction to Algorithms	3	TC-302	M-W-F: 11:00-11:55 AM	19	K69880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSC-234B	Data Structure & Algorithms	3	TC-114	T-H: 11:00-12:25 PM	15	J33486
CSC-242	Programming II	3	TC-303	T-H: 1:00-2:25 PM	18	A52990
CSC-320	Object Oriented Programming	3	TC-301	T-H: 1:00-2:25 PM	22	B66750
CSC-331	Applications Programming	3	TC-109	T-H: 11:00-12:25 PM	28	H99118
CSC-333A	Computer Arch & Algorithms	3	TC-301	M-W-F: 10:00-10:55 AM	22	A77587
CSC-333B	Computer Arch & Algorithms	3	TC-302	T-H: 11:00-12:25 PM	15	A77587

Figure 2.16 Completed Course table.

student_id	student_name	gpa	credits	major	schoolYear	email
A78835	Andrew Woods	3.26	108	Computer Science	Senior	awoods@college.edu
A97850	Ashly Jade	3.57	116	Information System Engineering	Junior	ajade@college.edu
B92996	Blue Valley	3.52	102	Computer Science	Senior	bvalley@college.edu
H10210	Holes Smith	3.87	78	Computer Engineering	Sophomore	hsmith@college.edu
J77896	Erica Johnson	3.95	127	Computer Science	Senior	ejohnson@college.edu

Figure 2.17 Completed Student table.

on the last table—StudentCourse. Click on the Add button, and then the Close button to close this dialog box. All five tables are added and displayed in the Relationships dialog box. The relationships we want to add are shown in Figure 2.19.

The P.K. and F.K. in Figure 2.19 represent the primary and foreign keys, respectively. For example, the `faculty_id` in the Faculty table is a primary key and it can be connected with the `faculty_id` that is a foreign key in the LogIn table. The relationship between these two tables is one-to-many since the unique Primary key `faculty_id` in the Faculty table can be connected to the multiple Foreign key that is `faculty_id` located in the LogIn table.

s_course_id	student_id	course_id	credit	major
1000	H10210	CSC-131D	3	CE
1001	B92996	CSC-132A	3	CS/IS
1002	J77896	CSC-335	3	CS/IS
1003	A78835	CSC-331	3	CE
1004	H10210	CSC-234B	3	CE
1005	J77896	CSC-234A	3	CS/IS
1006	B92996	CSC-233A	3	CS/IS
1007	A78835	CSC-132A	3	CE
1008	A78835	CSE-432	3	CE
1009	A78835	CSE-434	3	CE
1010	J77896	CSC-439	3	CS/IS
1011	H10210	CSC-132A	3	CE
1012	H10210	CSC-331	3	CE
1013	A78835	CSC-335	3	CE
1014	A78835	CSE-438	3	CE
1015	J77896	CSC-432	3	CS/IS
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE
1018	A97850	CSC-331	3	ISE
1019	A97850	CSC-335	3	ISE
1020	J77896	CSE-439	3	CS/IS

Figure 2.18 Completed StudentCourse table.

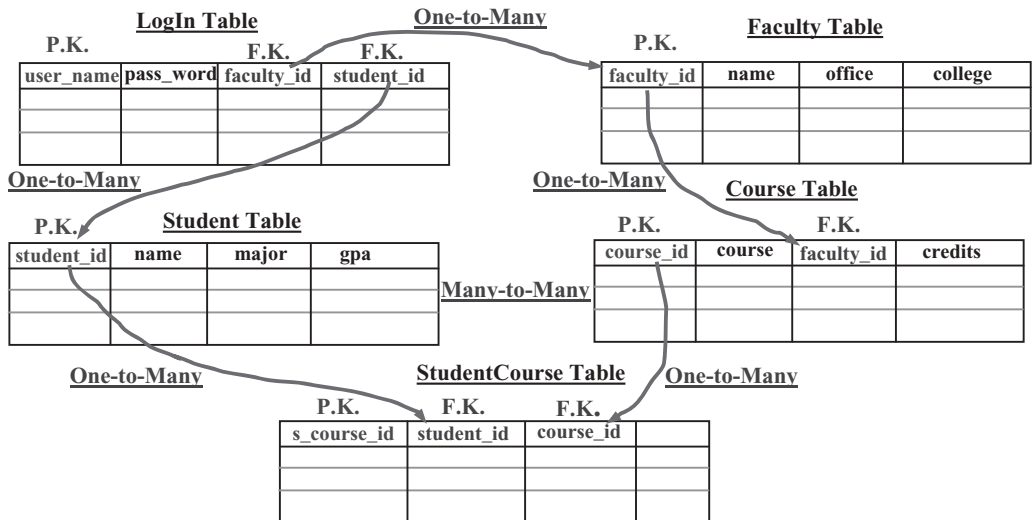


Figure 2.19 Relationships between tables.

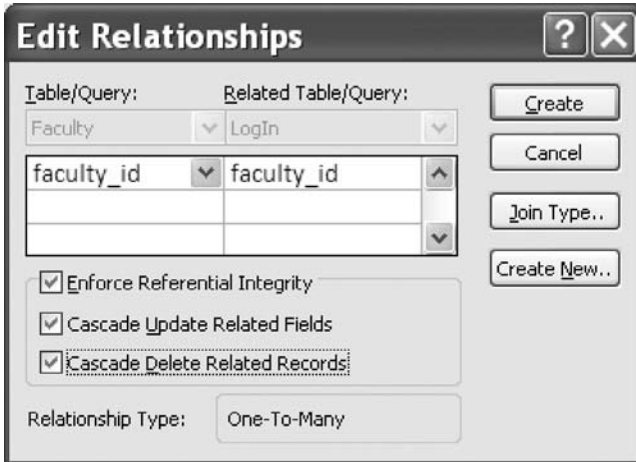


Figure 2.20 Edit Relationships dialog box.

To set this relationship between these two tables, click `faculty_id` from the Faculty table and drag to the `faculty_id` in the LogIn table. The Edit Relationships dialog box is displayed, which is shown in Figure 2.20.

Select the Enforce Referential Integrity checkbox to set up this reference integrity between these two fields. Also check the following two checkboxes:

- Cascade Update Related Fields
- Cascade Delete Related Records

The purpose of checking these two checkboxes is that all fields or records in the cascaded or child tables will be updated or deleted when the related fields or records in the parent tables are updated or deleted. This will greatly simplify the updating and deleting operations for a given relational database that contains a lot of related tables. Refer to Chapters 6 and 7 for more detailed discussions about the data updating and deleting actions.

Click on the Create button to create this relationship. In a similar way, you can create all other relationships between these five tables. One point you need to remember when you perform this dragging operation is to always start this drag from the primary key in the parent table and end it with the foreign key in the child table. As shown in Figure 2.20, the table located in the left of the Edit Relationships dialog is considered as the parent table, and the right of this dialog is the child table. Therefore, the `faculty_id` on the left is the primary key, and the `faculty_id` on the right is the foreign key, respectively.

The finished relationships dialog should match the one shown in Figure 2.21.

A completed Microsoft Access 2007 database file **CSE_DEPT.accdb** can be found in the folder **Access** located at the accompanying ftp site (see Chapter 1). Refer to Appendix F if you want to use this sample database in your applications.



During the process of creating relationships between tables, sometimes an error message may be displayed to indicate that some of tables are used by other users and are not locked to allow you to perform this relationship creation. In that case, just save your current result, close your database, and exit the Access. This error will be solved when you restart Access and open your database again. The reason for that is because some tables are considered to be used by you when you finished creating those tables and continue to perform the creating of the relationships between them.

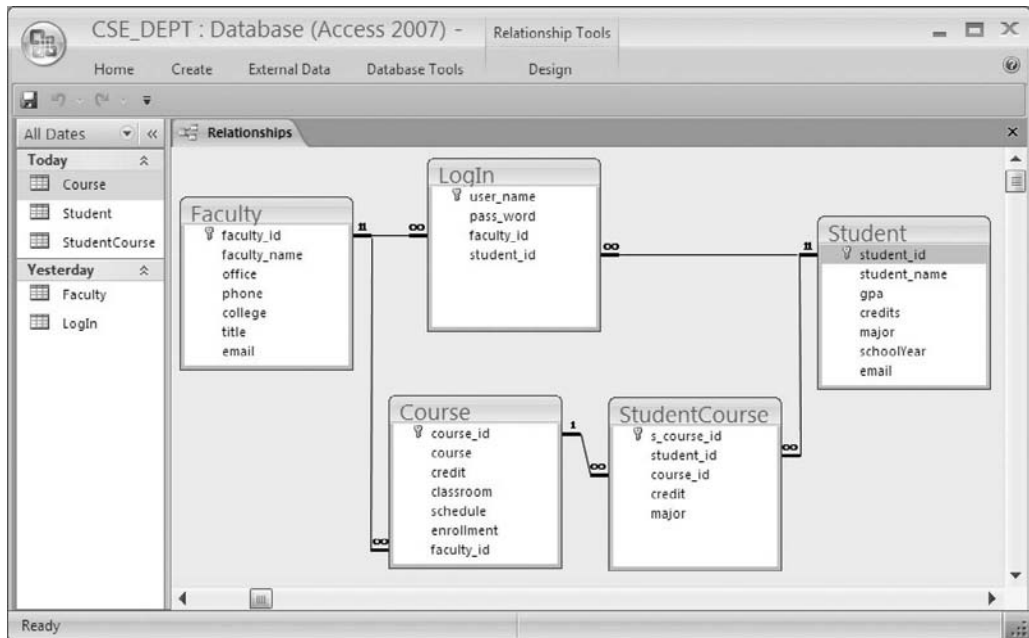


Figure 2.21 Completed relationships for tables.

2.10 CREATE MICROSOFT SQL SERVER 2005 SAMPLE DATABASE

After you finished the installation of SQL Server Management Studio Express (refer to Appendix B), you can begin to use it to connect to the server and build your database. To start, go to **Start|All Programs|Microsoft SQL Server 2005** and select **SQL Server Management Studio Express**. A connection dialog is opened as shown in Figure 2.22.

Your computer name followed by your server name should be displayed in the Server name: box. In this case, it is **YBAI\SQLEXPRESS5**. The Windows NT default security engine is used by selecting the **Windows Authentication** method from the Authentication box. The User name box contains the name you entered when you registered your computer. Click on the **Connect** button to connect your client to your server.



Figure 2.22 Connect to the SQL Server 2005.



Figure 2.23 Opened server management studio.

The Server Management Studio is opened when this connection is completed, which is shown in Figure 2.23.

To create a new database, right-click on the Databases folder from the Object Explorer window, and select the New Database item from the pop-up menu. Enter CSE_DEPT into the Database name box in the New Database dialog as the name of our database, keep all other settings unchanged and then click on the OK button. You can

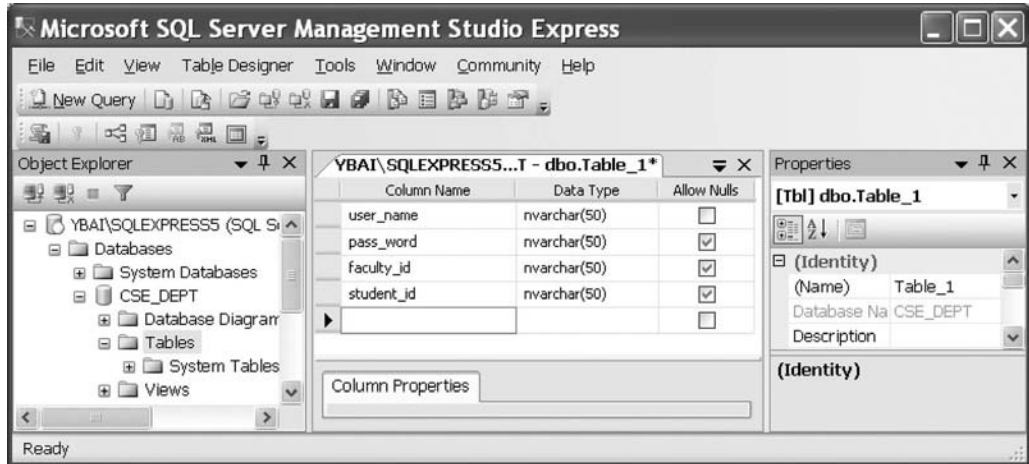


Figure 2.24 New table window.

find that a new database named CSE_DEPT is created, and it is located under the Database folder in the Object Explorer window.

Then you need to create data tables. For this sample database, you need to create five data tables: LogIn, Faculty, Course, Student, and StudentCourse. Expand the CSE_DEPT database folder by clicking on the plus symbol next to it. Right-click on the Tables folder and select the New Table item, a new table window is displayed, which is shown in Figure 2.24.

2.10.1 Create LogIn Table

A default data table named dbo.Table_1 is created as shown in Figure 2.24. Three columns are displayed in this new table: Column Name, Data Type, and Allow Nulls, which allow you to enter the name, the data type, and checkmark for each column. You can check the checkbox if you allow that column to be empty, otherwise do not check it if you want that column to contain a valid data. Generally for the column that has been selected to work as the primary key, you should not check the checkbox associated with that column.

The first table is LogIn table, which has four columns with the following column names: user_name, pass_word, faculty_id, and student_id. Enter those four names into the four Column Names columns. The data types for these four columns are all nvarchar(50), which means that this is a varied character type with a maximum letters of 50. Enter those data types into each Data Type column. The first column user_name is selected as the primary key, so leave the checkbox blank for that column and check the other three checkboxes.

To make the first column user_name as a primary key, click on the first row and then go to the Toolbar and select the primary key (displayed as a key) tool. In this way, a symbol of the primary key is displayed on the left of this row, which is shown in Figure 2.24.

Before we can continue to finish this LogIn table, we need first to save and name this table. Go to File|Save Table_1 and enter the **LogIn** as the name for this new table. Click on the OK button to finish this saving. A new table named **dbo.LogIn** is added into the new database under the Tables folder in the Object Explorer window.

To add data into this LogIn table, right-click on this table and select Open Table item from the pop-up menu. Enter all login data that is shown in Table 2.18 into this table. Your finished LogIn table should match the one shown in Figure 2.25.

Table 2.18 Data in the LogIn Table

user_name	pass_word	faculty_id	student_id
abrown	america	B66750	NULL
ajade	tryagain	NULL	A97850
awoods	smart	NULL	A78835
banderson	birthday	A52990	NULL
bvalley	see	NULL	B92996
dangles	tomorrow	A77587	NULL
hsmith	try	NULL	H10210
jerica	excellent	NULL	J77896
jhenry	test	H99118	NULL
jking	goodman	K69880	NULL
sbhalla	india	B86590	NULL
sjohnson	jermany	J33486	NULL
ybai	reback	B78880	NULL

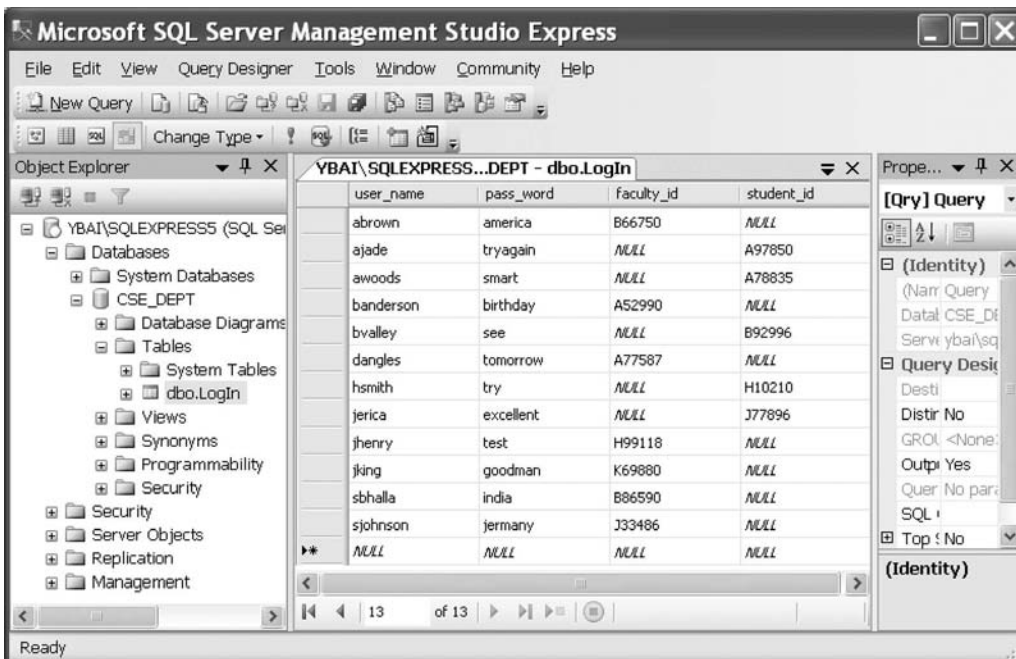


Figure 2.25 Finished LogIn table.

One point to note is that you must place a NULL for any field that has no value in this LogIn table since it is different for the blank field between the Microsoft Access and the SQL Server database. Go to the File/Save All item to save this table. Now let's continue to create the second table Faculty.

2.10.2 Create Faculty Table

Right-click on the Tables folder under the CSE_DEPT database folder and select the New Table item to open the design view of a new table, which is shown in Figure 2.26.

For this table, we have seven columns: faculty_id, faculty_name, office, phone, college, title, and email. The data type of the faculty_id is nvarchar(50), and all other data types are text since all of them are string variables. The reason we selected the nvarchar(50) as the data type for the faculty_id is that a primary key can work for this data type but it does not work for the text. The finished design view of the Faculty table should match the one shown in Figure 2.26.

Since we selected the faculty_id column as the primary key, click on that row and then go to the Toolbar and select the primary key tool. In this way, the faculty_id is chosen as the primary key for this table, which is shown in Figure 2.26.

Now go to the File menu item and select the Save Table_1, and enter Faculty into the box for the Choose Name dialog as the name for this table; click on OK to save this table.

Next you need to enter the data into this Faculty table. To do that, first open the table by right-clicking on the dbo.Faculty folder under the CSE_DEPT database folder in the Object Explorer window, and then select Open Table item to open this table. Enter the data shown in Table 2.19 into this Faculty table. Your finished Faculty table should match the one shown in Figure 2.27.

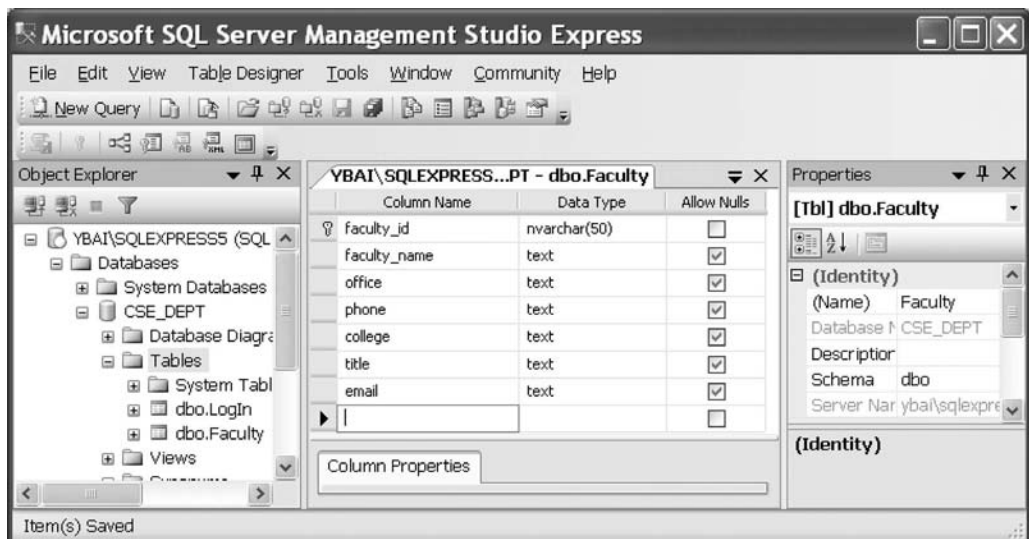
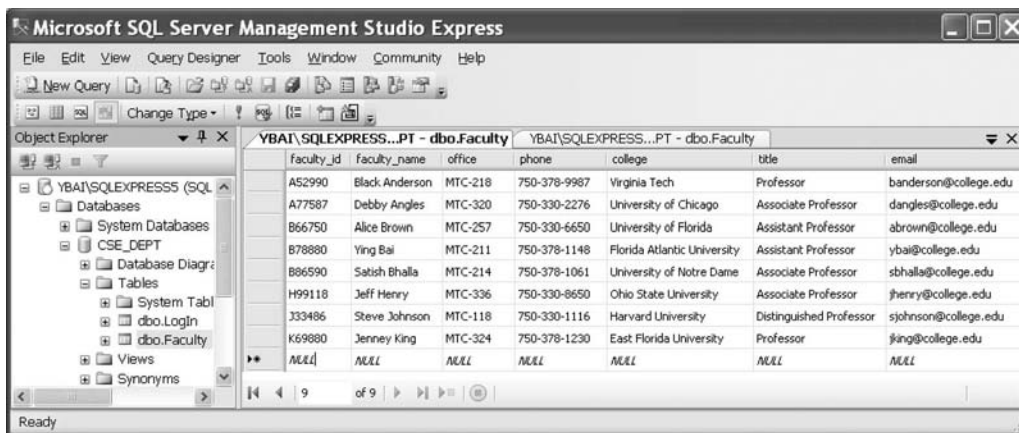


Figure 2.26 Design view of the Faculty table.

Table 2.19 Data in the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.edu
A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@college.edu
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

**Figure 2.27** Completed Faculty Table.

Now go to the File menu item and select Save All to save this completed Faculty data table. Your finished Faculty data table will be displayed as a table named `dbo.Faculty`, which has been added to the new database `CSE_DEPT` under the folder `Tables` in the Object Explorer window.

2.10.3 Create Other Tables

In a similar way, you need to create three more tables: `Course`, `Student`, and `StudentCourse`. Select `course_id`, `student_id`, and `s_course_id` as the Primary keys for these three tables (refer to Tables 2.20, 2.21, and 2.22). For the data type selections, follow the directions below:

The data type selections for the `Course` table:

- `course_id`—`nvarchar(50)` (primary key)
- `credit`—`float`
- `enrollment`—`int`
- `faculty_id`—`nvarchar(50)`
- all other columns—either `nvarchar(50)` or text

Table 2.20 Data in the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-131A	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	28	A52990
CSC-131B	Computers in Society	3	TC-114	M-W-F: 9:00-9:55 AM	20	B66750
CSC-131C	Computers in Society	3	TC-109	T-H: 11:00-12:25 PM	25	A52990
CSC-131D	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	30	B86590
CSC-131E	Computers in Society	3	TC-301	M-W-F: 1:00-1:55 PM	25	B66750
CSC-131I	Computers in Society	3	TC-109	T-H: 1:00-2:25 PM	32	A52990
CSC-132A	Introduction to Programming	3	TC-303	M-W-F: 9:00-9:55 AM	21	J33486
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-230	Algorithms & Structures	3	TC-301	M-W-F: 1:00-1:55 PM	20	A77587
CSC-232A	Programming I	3	TC-305	T-H: 11:00-12:25 PM	28	B66750
CSC-232B	Programming I	3	TC-303	T-H: 11:00-12:25 PM	17	A77587
CSC-233A	Introduction to Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	18	H99118
CSC-233B	Introduction to Algorithms	3	TC-302	M-W-F: 11:00-11:55 AM	19	K69880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSC-234B	Data Structure & Algorithms	3	TC-114	T-H: 11:00-12:25 PM	15	J33486
CSC-242	Programming II	3	TC-303	T-H: 1:00-2:25 PM	18	A52990
CSC-320	Object Oriented Programming	3	TC-301	T-H: 1:00-2:25 PM	22	B66750
CSC-331	Applications Programming	3	TC-109	T-H: 11:00-12:25 PM	28	H99118
CSC-333A	Computer Arch & Algorithms	3	TC-301	M-W-F: 10:00-10:55 AM	22	A77587
CSC-333B	Computer Arch & Algorithms	3	TC-302	T-H: 11:00-12:25 PM	15	A77587
CSC-335	Internet Programming	3	TC-303	M-W-F: 1:00-1:55 PM	25	B66750
CSC-432	Discrete Algorithms	3	TC-206	T-H: 11:00-12:25 PM	20	B86590
CSC-439	Database Systems	3	TC-206	M-W-F: 1:00-1:55 PM	18	B86590
CSE-138A	Introduction to CSE	3	TC-301	T-H: 1:00-2:25 PM	15	A52990
CSE-138B	Introduction to CSE	3	TC-109	T-H: 1:00-2:25 PM	35	J33486
CSE-330	Digital Logic Circuits	3	TC-305	M-W-F: 9:00-9:55 AM	26	K69880
CSE-332	Foundations of Semiconductors	3	TC-305	T-H: 1:00-2:25 PM	24	K69880
CSE-334	Elec. Measurement & Design	3	TC-212	T-H: 11:00-12:25 PM	25	H99118
CSE-430	Bioinformatics in Computer	3	TC-206	Thu: 9:30-11:00 AM	16	B86590
CSE-432	Analog Circuits Design	3	TC-309	M-W-F: 2:00-2:55 PM	18	K69880
CSE-433	Digital Signal Processing	3	TC-206	T-H: 2:00-3:25 PM	18	H99118
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-436	Automatic Control and Design	3	TC-305	M-W-F: 10:00-10:55 AM	29	J33486
CSE-437	Operating Systems	3	TC-303	T-H: 1:00-2:25 PM	17	A77587
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880
CSE-439	Special Topics in CSE	3	TC-206	M-W-F: 10:00-10:55 AM	22	J33486

Table 2.21 Data in the Student Table

student_id	student_name	gpa	credits	major	schoolYear	email
A78835	Andrew Woods	3.26	108	Computer Science	Senior	awoods@college.edu
A97850	Ashly Jade	3.57	116	Information System Engineering	Junior	ajade@college.edu
B92996	Blue Valley	3.52	102	Computer Science	Senior	bvalley@college.edu
H10210	Holes Smith	3.87	78	Computer Engineering	Sophomore	hsmith@college.edu
J77896	Erica Johnson	3.95	127	Computer Science	Senior	ejohnson@college.edu

Table 2.22 Data in the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1000	H10210	CSC-131D	3	CE
1001	B92996	CSC-132A	3	CS/IS
1002	J77896	CSC-335	3	CS/IS
1003	A78835	CSC-331	3	CE
1004	H10210	CSC-234B	3	CE
1005	J77896	CSC-234A	3	CS/IS
1006	B92996	CSC-233A	3	CS/IS
1007	A78835	CSC-132A	3	CE
1008	A78835	CSE-432	3	CE
1009	A78835	CSE-434	3	CE
1010	J77896	CSC-439	3	CS/IS
1011	H10210	CSC-132A	3	CE
1012	H10210	CSC-331	2	CE
1013	A78835	CSC-335	3	CE
1014	A78835	CSE-438	3	CE
1015	J77896	CSC-432	3	CS/IS
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE
1018	A97850	CSC-331	3	ISE
1019	A97850	CSC-335	3	ISE
1020	J77896	CSE-439	3	CS/IS
1021	B92996	CSC-230	3	CS/IS
1022	A78835	CSE-332	3	CE
1023	B92996	CSE-430	3	CE
1024	J77896	CSC-333A	3	CS/IS
1025	H10210	CSE-433	3	CE
1026	H10210	CSE-334	3	CE
1027	B92996	CSC-131C	3	CS/IS
1028	B92996	CSC-439	3	CS/IS

The data type selections for the Student table:

- student_id—nvarchar(50) (primary key)
- gpa—float
- credits—int
- all other columns—either nvarchar(50) or text

The data type selections for the StudentCourse table:

- s_course_id—int (primary key)
- student_id—nvarchar(50)
- course_id—nvarchar(50)
- credit—int
- major—either nvarchar(50) or text

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-131A	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	28	A52990
CSC-131B	Computers in Society	3	TC-114	M-W-F: 9:00-9:55 AM	20	B66750
CSC-131C	Computers in Society	3	TC-109	T-H: 11:00-12:25 PM	25	A52990
CSC-131D	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	30	B86590
CSC-131E	Computers in Society	3	TC-301	M-W-F: 1:00-1:55 PM	25	B66750
CSC-131I	Computers in Society	3	TC-109	T-H: 1:00-2:25 PM	32	A52990
CSC-132A	Introduction to Programming	3	TC-303	M-W-F: 9:00-9:55 AM	21	J33486
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-230	Algorithms & Structures	3	TC-301	M-W-F: 1:00-1:55 PM	20	A77587
CSC-232A	Programming I	3	TC-305	T-H: 11:00-12:25 PM	28	B66750
CSC-232B	Programming I	3	TC-303	T-H: 11:00-12:25 PM	17	A77587
CSC-233A	Introduction to Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	18	H99118
CSC-233B	Introduction to Algorithms	3	TC-302	M-W-F: 11:00-11:55 AM	19	K69880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSC-234B	Data Structure & Algorithms	3	TC-114	T-H: 11:00-12:25 PM	15	J33486
CSC-242	Programming II	3	TC-303	T-H: 1:00-2:25 PM	18	A52990
CSC-320	Object Oriented Programming	3	TC-301	T-H: 1:00-2:25 PM	22	B66750
CSC-331	Applications Programming	3	TC-109	T-H: 11:00-12:25 PM	28	H99118
CSC-333A	Computer Arch & Algorithms	3	TC-301	M-W-F: 10:00-10:55 AM	22	A77587
CSC-333B	Computer Arch & Algorithms	3	TC-302	T-H: 11:00-12:25 PM	15	A77587
CSC-335	Internet Programming	3	TC-303	M-W-F: 1:00-1:55 PM	25	B66750

Figure 2.28 Completed Course table.

Enter the data shown in Tables 2.20, 2.21, and 2.22 into each associated table, and save each table as Course, Student, and StudentCourse, respectively.

The finished Course table should match the one shown in Figure 2.28.

The finished Student table should match the one shown in Figure 2.29.

The finished StudentCourse table should match the one that in Figure 2.30.

One point you need to note is that you can copy the content of the whole table from the Microsoft Access database file to the associated data table opened in the Microsoft SQL Server environment if the Microsoft Access database has been developed.

To make these copies and pastes, first you must select a whole blank row from your destination table—the table in the Microsoft SQL Server database—and then select all data rows from your source table—the Microsoft Access database file—by highlighting them and choose the Copy menu item. Next you need to paste those rows by clicking the blank row in the Microsoft SQL Server database and then click on the Paste item from the Edit menu item. An error message may be displayed as shown in Figure 2.31.

Just click on the OK button and your data will be pasted to your destination table without problem. The reason for the error message is because of the primary key, which cannot be a NULL value. Before you can finish this paste operation, the table cannot identify whether you will have a non-null value in the source row that will be pasted in this column or not.

Microsoft SQL Server Management Studio Express

YBAI\SQLEXPRESS...PT - dbo.Student

student id	student name	apa	credits	major	schoolYear	email
A78835	Andrew Woods	3.26	108	Computer Science	Senior	awoods@college.edu
A97850	Ashly Jade	3.57	116	Information System Engineering	Junior	ajade@college.edu
B92996	Blue Valley	3.52	102	Computer Science	Senior	bvalley@college.edu
H10210	Holes Smith	3.87	76	Computer Engineering	Sophomore	hsmith@college.edu
J77896	Erica Johnson	3.95	127	Computer Science	Senior	ejohnson@college.edu
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ready

Figure 2.29 Completed Student table.

Microsoft SQL Server Management Studio Express

YBAI\SQLEXPRES...o.StudentCourse

s course id	student id	course id	credit	major
1012	H10210	CSC-331	2	CE
1013	A78835	CSC-335	3	CE
1014	A78835	CSE-438	3	CE
1015	J77896	CSC-432	3	CS/IS
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE
1018	A97850	CSC-331	3	ISE
1019	A97850	CSC-335	3	ISE
1020	J77896	CSE-439	3	CS/IS
1021	B92996	CSC-230	3	CS/IS
1022	A78835	CSE-332	3	CE
1023	B92996	CSE-430	3	CE
1024	J77896	CSC-333A	3	CS/IS
1025	H10210	CSE-433	3	CE
1026	H10210	CSE-334	3	CE
1027	B92996	CSC-131C	3	CS/IS
1028	B92996	CSC-439	3	CS/IS
NULL	NULL	NULL	NULL	NULL

Item(s) Saved

Figure 2.30 Completed StudentCourse table.

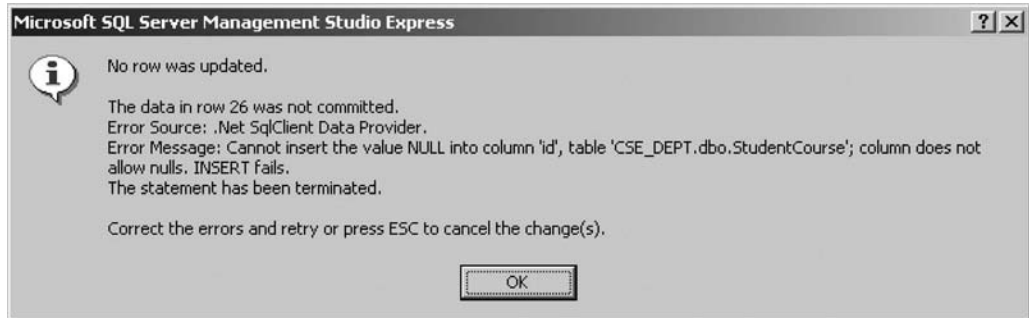


Figure 2.31 Error message when performing a paste job.

2.10.4 Create Relationships Among Tables

Next we need to set up relationships among these five tables using the primary and foreign keys. In the Microsoft SQL Server 2005 Express database environment, the relationship between tables can be set by using the Keys folder under each data table from the Object Explorer window. Now let's begin to set up the relationship between the LogIn and the Faculty tables.

2.10.4.1 Create Relationship Between LogIn and Faculty Tables

The relationship between the Faculty and the LogIn table is one-to-many, which means that the faculty_id is a primary key in the Faculty table, and it can be mapped to the many faculty_id that are foreign keys in the LogIn table. To set up this relationship, expand the LogIn table and the Keys folder that is under the LogIn table. Currently only one primary key, PK_LogIn, is existed under the Keys folder.

To add a new foreign key, right-click on the Keys folder and select New Foreign Key item from the pop-up menu to open the Foreign Key Relationships dialog, which is shown in Figure 2.32.

The default foreign relationship is FK_LogIn_LogIn*, which is displayed in the Selected Relationship box. Right now we want to create the foreign relationship between the LogIn and the Faculty tables, so change the name of this foreign relationship to FK_LogIn_Faculty by modifying its name in the (Name) box, which is under the Identity pane, and then press the Enter key on your keyboard. Then select two tables by clicking on the Tables And Columns Specification item that is under the General pane. Click the expansion button located on the right of the Tables And Columns Specification item to open the Tables and Columns dialog, which is shown in Figure 2.33.

Click on the drop-down arrow from the primary key table combobox and select the Faculty table since we need the primary key faculty_id from this table; then click on the blank row that is just below the primary key table combobox and select the faculty_id column. You can see that the LogIn table has been automatically selected and displayed in the foreign key table combobox. Click on the drop-down arrow from the box that is just under the foreign key table combobox and select the faculty_id as the foreign key for the LogIn table. Your finished Tables and Columns dialog should match the one shown in Figure 2.34. Click on OK to close this dialog.

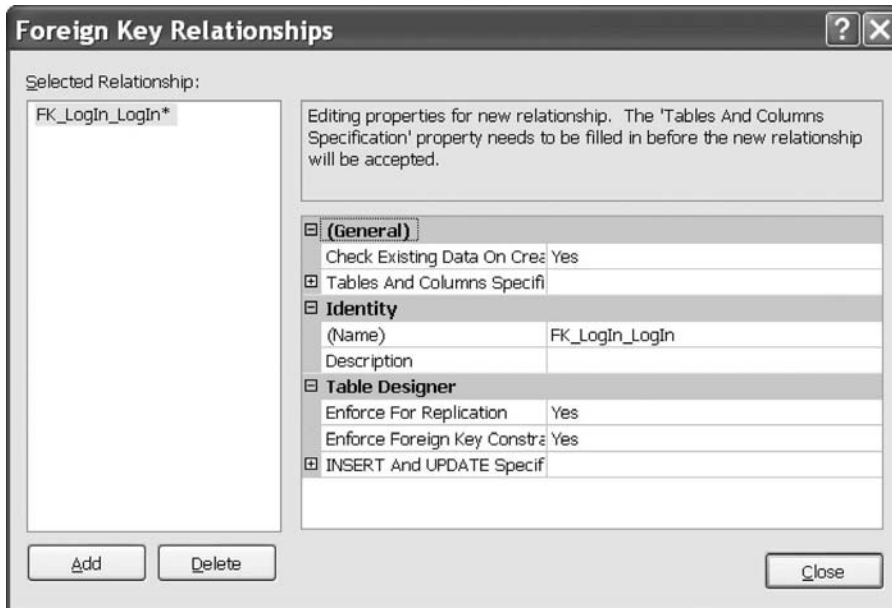


Figure 2.32 Opened Foreign Key Relationships dialog box.

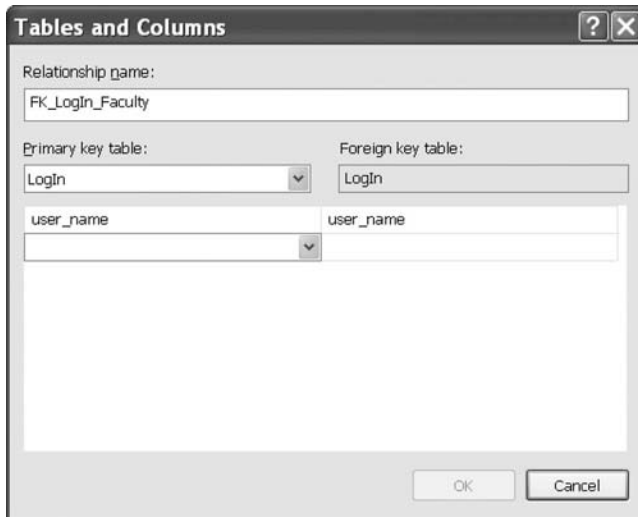


Figure 2.33 Opened Tables and Columns dialog box.

Before we can close this dialog, we need to do one more thing, which is to set up a cascaded relationship between the primary key (`faculty_id`) in the parent table `Faculty` and the foreign keys (`faculty_id`) in the child table `LogIn`. The reason we need to do this is because we want to simplify the data updating and deleting operations between these tables in a relational database such as `CSE_DEPT`. You will have a better understanding

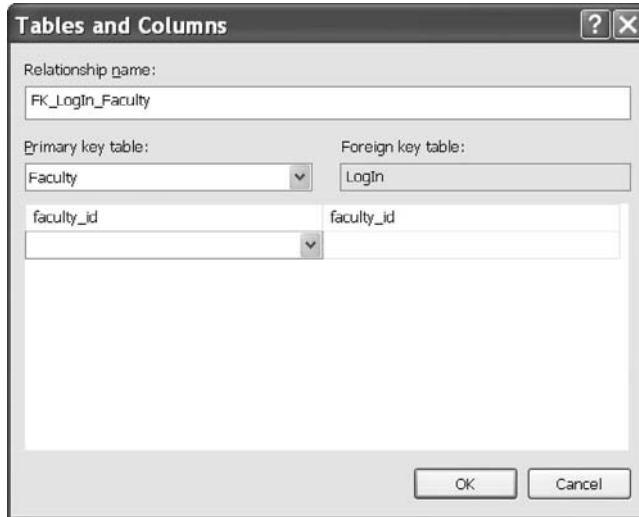


Figure 2.34 Finished Tables and Columns dialog box.

about this cascading later on when you learn how to update and delete data against a relational database in Chapters 7 and 8.

To do this cascading, scroll down along this Foreign Key Relationships dialog and expand the item Table Designer. You find the INSERT And UPDATE Specifications item. Expand this item by clicking on the small plus icon, two subitems are displayed, which are:

- Delete Rule
- Update Rule

The default value for both subitems are No Action. Click on the No Action box for the Delete Rule item and then click on the drop-down arrow, and select the Cascade item from the list. Perform the same operation for the Update Rule item. Your finished Foreign Key Relationships dialog should match the one shown in Figure 2.35.

In this way, we established the cascaded relationship between the primary key in the parent table and the foreign keys in the child table. Later on when you update or delete any primary key from a parent table, the related foreign keys in the child tables will also be updated or deleted without other additional operations. It is convenient! Click on the Close button to close this dialog.

Go to the File|Save LogIn menu item to open the Save dialog and click on the Yes button to save this relationship. You can select Yes or No to the Save Change Script dialog box if it appears.

Now right-click on the Keys folder under the LogIn table from the Object Explorer window, and select the Refresh item from the pop-up menu to refresh this Keys folder. Immediately you can find a new foreign key named FK_LogIn_Faculty that appears under this Keys folder. This is our new created foreign key that sets the relationship between our LogIn and Faculty tables. You can confirm and find this new created foreign key by right-clicking on the Keys folder that is under the Faculty table.

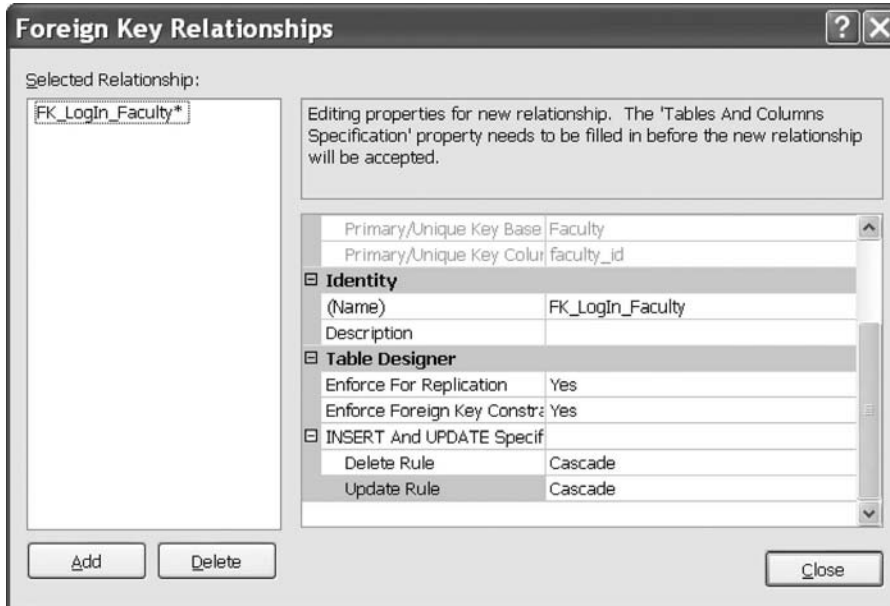


Figure 2.35 Finished Foreign Key Relationships dialog.

2.10.4.2 Create Relationship Between LogIn and Student Tables

In a similar way, you can create a foreign key for the LogIn table and set up a one-to-many relationship between the Student and the LogIn tables.

Right-click on the Keys folder that is under the dbo.LogIn table and select New Foreign Key item from the pop-up menu to open the Foreign Key Relationships dialog. Change the name to FK_LogIn_Student and press the Enter key on your keyboard. Go to the Tables And Columns Specification item to open the Tables and Columns dialog, then select the Student table from the primary key table combobox and student_id from the box that is under the primary key table combobox. Select the student_id from the box that is under the foreign key table combobox. Your finished Tables and Columns dialog should match the one shown in Figure 2.36.

Click on the OK to close this dialog box. Do not forget to establish the cascaded relationship for Delete Rule and Update Rule items by expanding the Table Designer and the INSERT And UPDATE Specifications items, respectively. Click on the Close button to close the Foreign Key Relationships dialog box.

Go to the File|Save LogIn menu item to save this relationship. Click on Yes for the following dialog box to finish this saving. Now right-click on the Keys folder that is under the dbo.LogIn table, and select Refresh item to show our new created foreign key FK_LogIn_Student.

2.10.4.3 Create Relationship Between Faculty and Course Tables

The relationship between the Faculty and the Course tables is one-to-many, and the faculty_id in the Faculty table is a primary key and the faculty_id in the Course table is a foreign key.

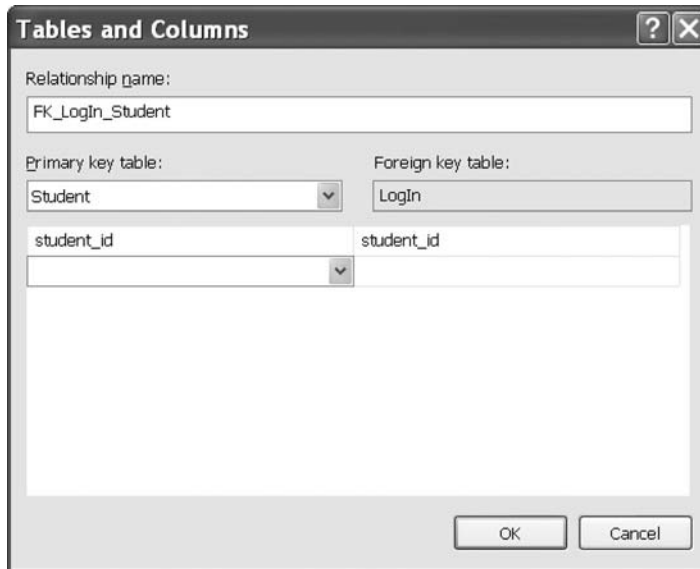


Figure 2.36 Completed Tables and Columns dialog.

Right-click on the Keys folder under the dbo.Course table from the Object Explorer window and select the New Foreign Key item from the pop-up menu. On the opened Foreign Key Relationships dialog, change the name of this new relationship to FK_Course_Faculty in the (Name) box and press the Enter key on the keyboard. In the opened Tables and Columns dialog box, select the Faculty table from the primary key table combobox and select the faculty_id from the box that is just under the primary key table combobox. Then select the faculty_id from the box that is just under the foreign key table combobox. Your finished Tables and Columns dialog should match the one shown in Figure 2.37.

Click on OK to close this dialog and set up the cascaded relationship for the Delete Rule and the Update Rule items, and then click on the Close button to close the Foreign Key Relationships dialog box. Go to the File|Save Course menu item and click on Yes for the following dialog box to save this setting.

Now right-click on the Keys folder under the dbo.Course table, and select the Refresh item. Immediately you can find our new created relationship key FK_Course_Faculty.

2.10.4.4 Create Relationship Between Student and StudentCourse Tables

The relationship between the Student and the StudentCourse tables is one-to-many, and the student_id in the Student table is a primary key and the student_id in the StudentCourse table is a foreign key.

Right-click on the Keys folder under the dbo.StudentCourse table from the Object Explorer window and select the New Foreign Key item from the pop-up menu. On the opened Foreign Key Relationships dialog, change the name of this new relationship to FK_StudentCourse_Student in the (Name) box and press the Enter key on the keyboard.

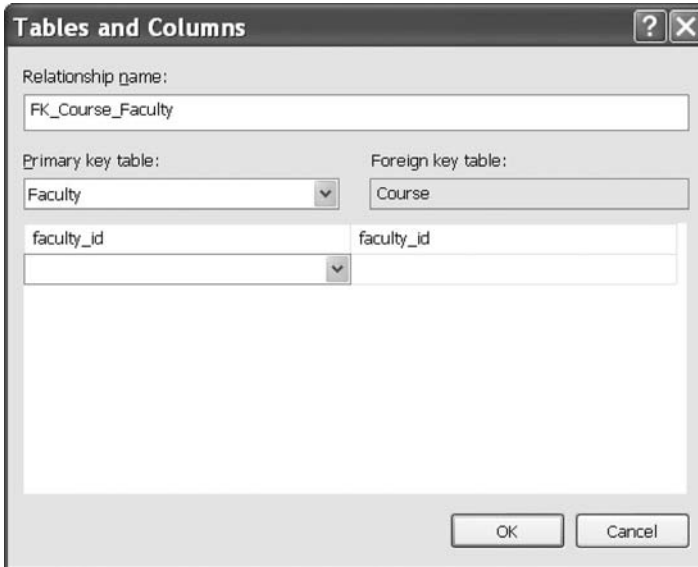


Figure 2.37 The finished Tables and Columns dialog.

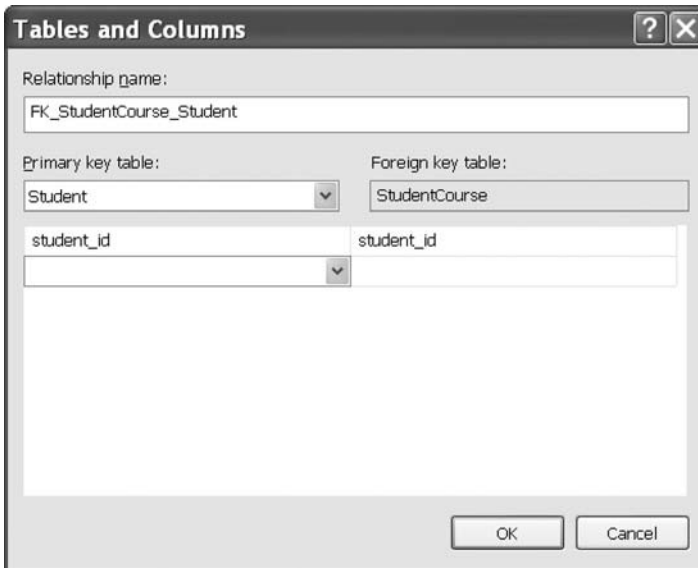


Figure 2.38 Finished Tables and Columns dialog.

In the opened Tables and Columns dialog box, select the Student table from the primary key table combobox and select the student_id from the box that is just under the primary key table combobox. Then select the student_id from the box that is just under the foreign key table combobox. The finished Tables and Columns dialog should match the one shown in Figure 2.38.

Click on OK to close this dialog and set up the cascaded relationship for Delete Rule and the Update Rule items, and then click on the Close button to close the Foreign Key Relationships dialog box. Go to the File|Save StudentCourse menu item and click on Yes for the following dialog box to save this relationship.

Now right-click on the Keys folder under the dbo.StudentCourse table, and select the Refresh item. Immediately you can find our new created relationship key FK_StudentCourse_Student.

2.10.4.5 Create Relationship Between Course and StudentCourse Tables

The relationship between the Course and the StudentCourse tables is one-to-many, and the course_id in the Course table is a primary key and the course_id in the StudentCourse table is a foreign key.

Right-click on the Keys folder under the dbo.StudentCourse table from the Object Explorer window and select the New Foreign Key item from the pop-up menu. On the opened Foreign Key Relationships dialog, change the name of this new relationship to FK_StudentCourse_Course in the (Name) box and press the Enter key on the keyboard. In the opened Tables and Columns dialog box, select the Course table from the Primary key table combobox and select the course_id from the box that is just under the Primary key table combobox. Then select the course_id from the box that is just under the Foreign key table combobox. Your finished Tables and Columns dialog should match the one shown in Figure 2.39.

Click on OK to close this dialog and do not forget to establish a cascaded relationship for the Delete Rule and the Update Rule items, and then click on the Close button to close the Foreign Key Relationships dialog box. Then go to the File|Save StudentCourse menu item and click on Yes for the following dialog box to save this relationship.

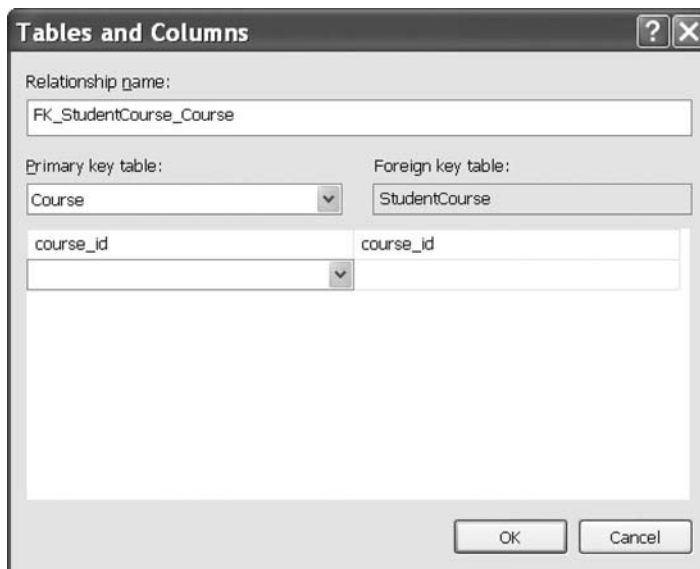


Figure 2.39 Finished Tables and Columns dialog.

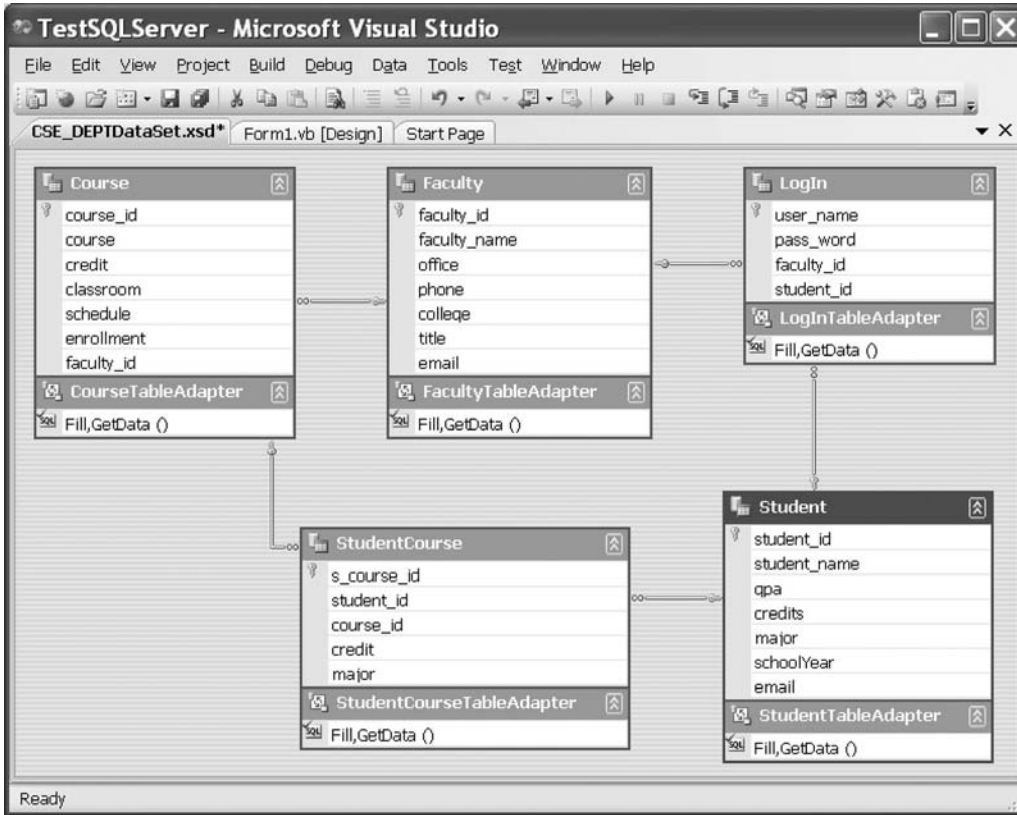


Figure 2.40 Relationships among tables.

Now right-click on the Keys folder under the `dbo.StudentCourse` table, and select the Refresh item. Immediately you can find our new created relationship key `FK_StudentCourse_Course`.

At this point, we complete setting the relationships among our five data tables. A completed Microsoft SQL Server 2005 sample database file `CSE_DEPT.mdf` can be found from the folder **SQLServer** located at the accompanying ftp site (see Chapter 1). Refer to Appendix F if you want to use this sample database in your C# applications. The completed relationships for these tables are shown in Figure 2.40.

2.11 CREATE ORACLE 10G XE SAMPLE DATABASE

After you download and installed Oracle Database 10g XE (refer to Appendix C), you need to create a customer Oracle database. To do that, you need to start this job from the Oracle home page in the server. To connect your computer to your Oracle server, go to `Start\All Programs\Oracle Database 10g Express Edition\Go To Database Home Page` to open the Login page, which is shown in Figure 2.41.

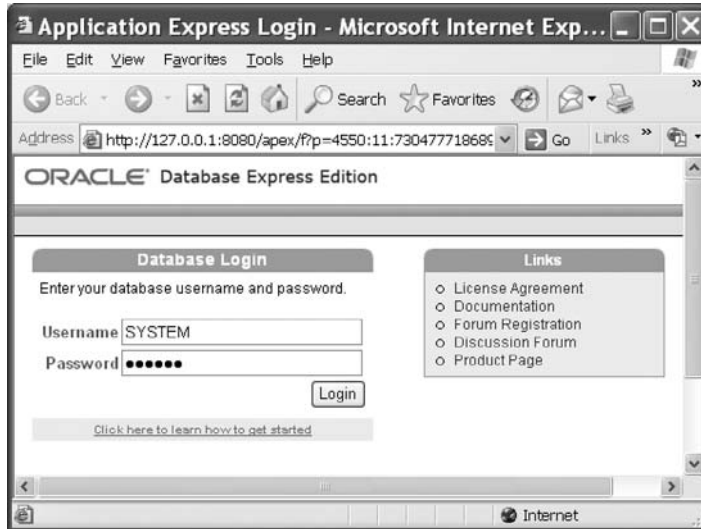


Figure 2.41 Opened Login page.

You can login as an Administrator by entering `SYSTEM` into the Username box and password you selected during your download and installation of the Oracle Database 10g XE into the Password box if you want to use the whole system. But if you want to create customer databases, you must create a new user and login as that user. We will concentrate on creating a customer database, `CSE_DEPT`, in this section only. If you want to work as an Administrator to create and manage all data sources, just log on yourself as an Administrator. There is no difference between the Administrator and a specific database user in creating and manipulating tables in Oracle Database 10g XE, and the only difference is that the Administrator has more control abilities than any specific database user.

In Oracle Database 10g XE, only a single database instance is allowed to be created and implemented for any database applications. To make the database simple and easy, each database object is considered as a schema, and each schema is related to a user or a user account. When you create a new user and assign a new account to that user, you create a new schema. A schema is a logical container for the database objects (such as tables, views, triggers, etc.) that the user creates. The schema name is the same as the username and can be used to unambiguously refer to objects owned by the user.

After you download and installed Oracle Database 10g XE, by default that database contains a lot of default utility tables, and most of them are not related to our special applications. In order to specify a database for our applications, we need to create a user database or a schema to meet our specific requirements in our applications. Following we will use the `CSE_DEPT` database as an example to illustrate how to create a user database in Oracle Database 10g XE.

2.11.1 Create Oracle User Database

To create a schema or a new user database, we need to create a new user account with the following steps:

1. Log on to Oracle Database 10g XE as the Administrator using the user id SYSTEM and your password.
2. Create a new user account using the Administration|Database Users|Create User items.
3. Enter the desired username and password.
4. Click on the Create button to create a new user account.

Now let's follow these four steps to create our new user account or user database named CSE_DEPT.

Open the Oracle Database 10g XE home page, log in as an Administrator, and click on the Administration button and go to Administration|Database Users|Create User. On the opened dialog box, enter CSE_DEPT into the Username and reback into the Password and the Confirm Password textboxes, respectively. Keep the Unlocked in the Account Status box unchanged and check the following checkboxes:

- CONNECT
- RESOURCE
- CREATE DATABASE LINK
- CREATE TABLE
- CREATE TYPE

The purpose of checking these checkboxes is to allow this new created user to be able to set up a connection (CONNECT) with this user database, use all data sources, set up database links, and create tables as well as the data types as a new instance of this database is created in an application.

Your finished creating new user dialog box should match the one shown in Figure 2.42. Click on the Create button to create this new user.

After a new user is created, a new schema or a database is also created with the same name as the user's name, CSE_DEPT. Next we can add new data tables into this new database. To do that, first we need to log out from the current Administrator account and log in as our new user account CSE_DEPT. Click on the Logout icon on the upper-right corner of this window and click on the Login icon to open the home page again.

Enter CSE_DEPT into the Username and reback into the Password textboxes to log in as a CSE_DEPT user.

Now we can create five data tables such as LogIn, Faculty, Course, Student, and StudentCourse in this CSE_DEPT user account or user database as we did in the last part. Also we need to define the constraint relationships between those five tables as we did in the last section to set up relationships between these five tables.

2.11.2 Add New Data Tables into Oracle User Database

You need to use the Object Browser to create, access, and manipulate your data in this database via the Oracle Database 10g Express server. After you log in as the CSE_DEPT user, click on the Object Browser icon to open the Object Browser window, which is shown in Figure 2.43.

Since we just created a new database CSE_DEPT without any table attached with this database, therefore a blank table list is displayed under the Tables item, as shown in

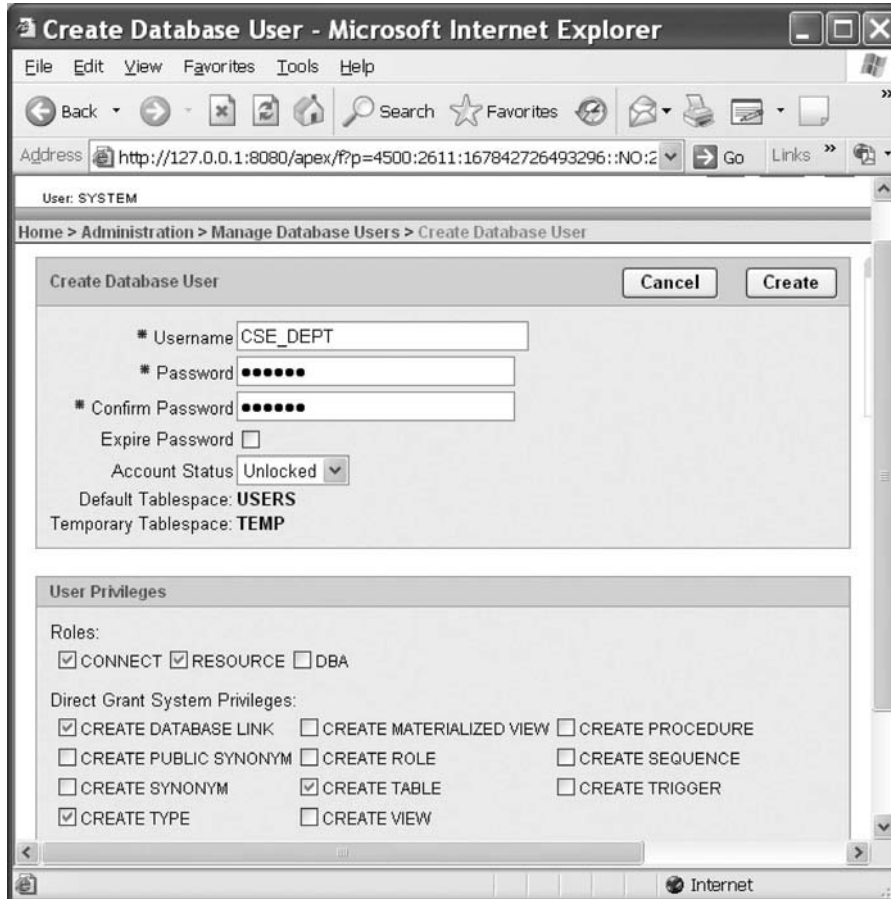


Figure 2.42 Create new user dialog box.

Figure 2.43. We need to create and add five new tables, LogIn, Faculty, Course, Student, and StudentCourse, into this new database.

2.11.2.1 Create LogIn Table

Make sure that the content of the textbox in the left column is Tables, which is the default value, click on the drop-down arrow of the Create button and select Tables to create our first new table—LogIn, which is shown in Figure 2.44.

A flowchart of developing the table is shown in the left pane. This means that you need to follow these five steps to finish creating your data table, and each step is mapped to one page. The middle pane contains the most components that allow us to create and edit our data table. Enter the LogIn as the table name into the Table Name box.

The first step in the flowchart is the Columns, which means that you need to create each column based on the information of your data table, such as the Column Name, Type, Precision, Scale, and Not Null. For our LogIn table, we have four columns: user_

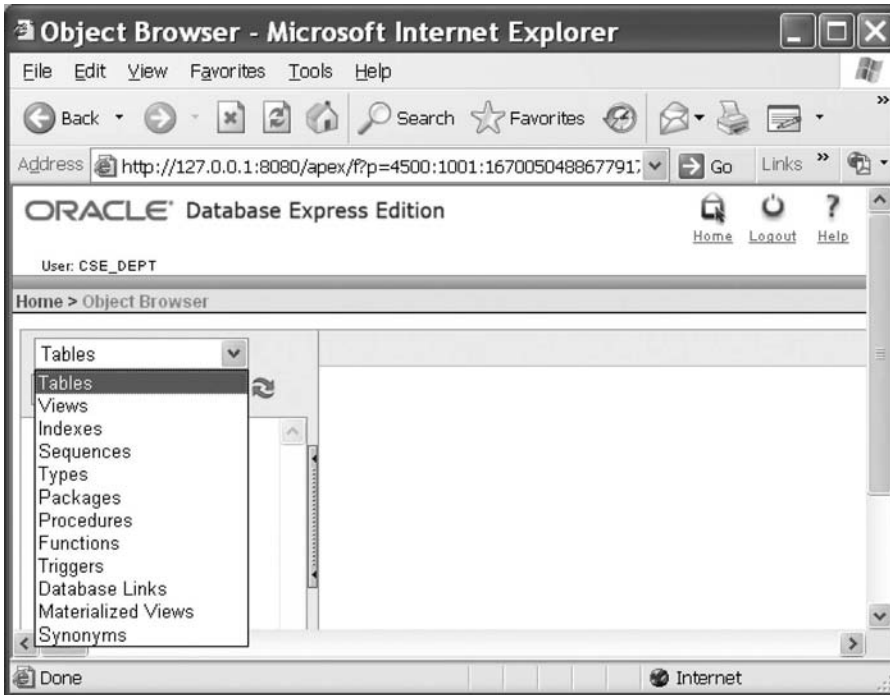


Figure 2.43 Opened Object Browser window.

name, pass_word, faculty_id, and student_id. The data type for all columns is VARCHAR2(15) since this data type is flexible and it can contain varying-length characters. The upper bound of the length is 15, which is determined by the number you entered in the Scale box, and it means that each column can contain up to 15 characters. Since the user_name is selected as the primary key for this table, check the Not Null checkbox next to this column to indicate that this column cannot contain a blank value. Your finished first step is shown in Figure 2.44.

Click on the Next button to go to the next page to assign the primary key for this table, which is shown in Figure 2.45.

To assign a primary key to our new LogIn table, select the Not Populated from the Primary key selection list because we don't want to use any Sequence object to assign any sequence to our primary key. The Sequence object in Oracle is used to automatically create a sequence of numeric numbers for the primary key. In our case, our primary key is a string and therefore we cannot use this object. Keep the Primary Key Name, LOGIN_PK, unchanged and select the USER_NAME(VARCHAR2) from the Primary key box. In this way, we select the user_name as the primary key for this table. Since we do not have any Composite Primary Key for this table, just keep this box unchanged. Your finished second step should match the one shown in Figure 2.45. Click on the Next button to continue to the next page—Set the foreign key page.

Since we have not created any other tables, we cannot select our foreign key for this LogIn table right now. We leave this job to be handled later. Click on the Next button

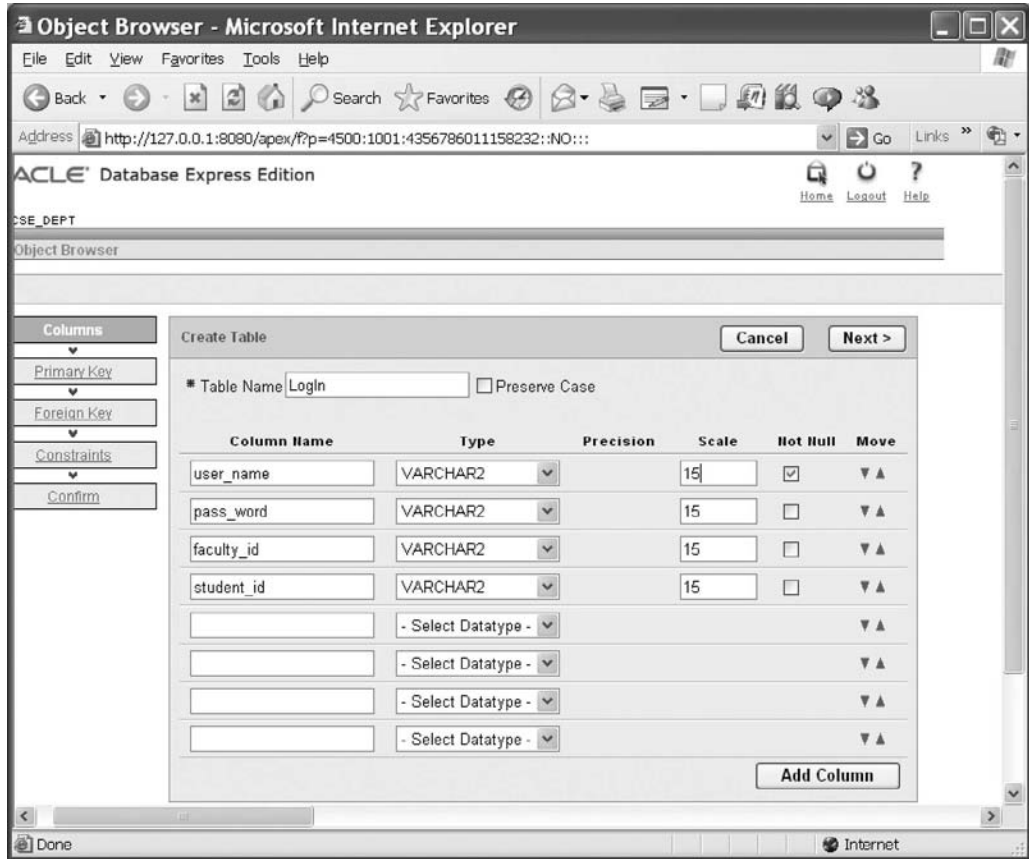


Figure 2.44 Create a new table.

to go to the next page. The next page allows you to set up some constraints on this table, which is shown in Figure 2.46.

No constraint is needed for this sample database at this moment, so you can click on the Finish button to go to the last page to confirm our LogIn table. The opened Confirm page is shown in Figure 2.47.

Click on the Create button to create and confirm this new LogIn table. Your created LogIn table should match the one shown in Figure 2.48 if it is successful. The new created LogIn table is also added into the left pane.

After the LogIn table is created, the necessary editing tools are attached with this table and displayed at the top of this table. The top row of these tools contains object editing tools, and the bottom line includes the actual editing methods. The editing methods include Add Column, Modify Column, Rename Column, and Drop Column, and these methods are straightforward in meaning without question.

To add data into this new LogIn table, you need to use and open the Data object tool in the top row. Click on the Data tool to open the Data page, which is shown in Figure 2.49.

Click on the Insert Row button to open the data sheet view of the LogIn table, which is shown in Figure 2.50.

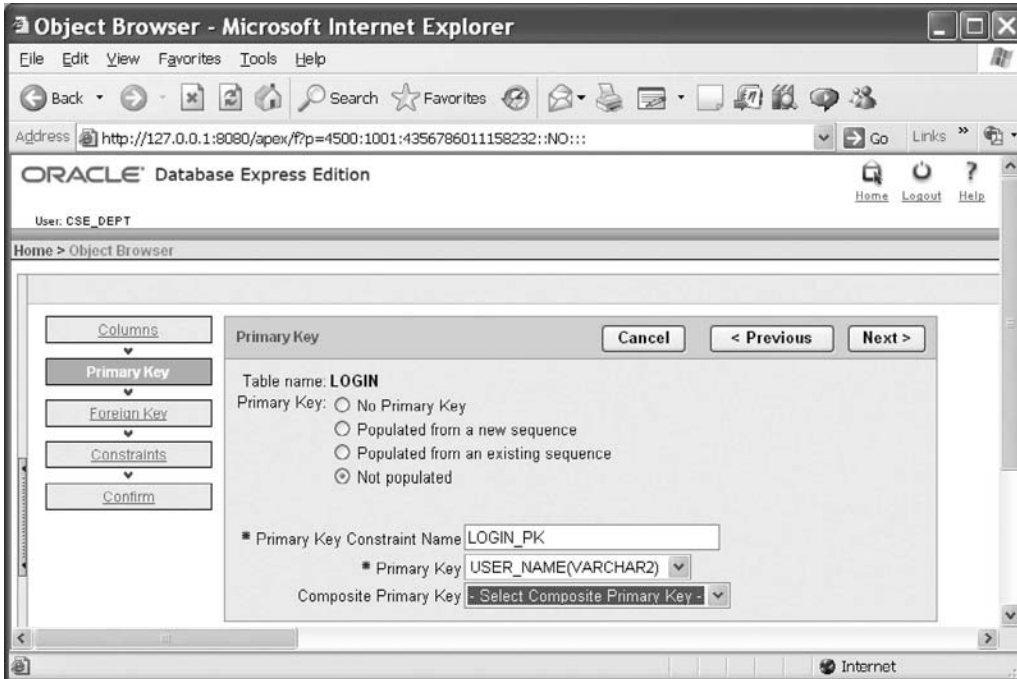


Figure 2.45 Second step—assign the primary key.

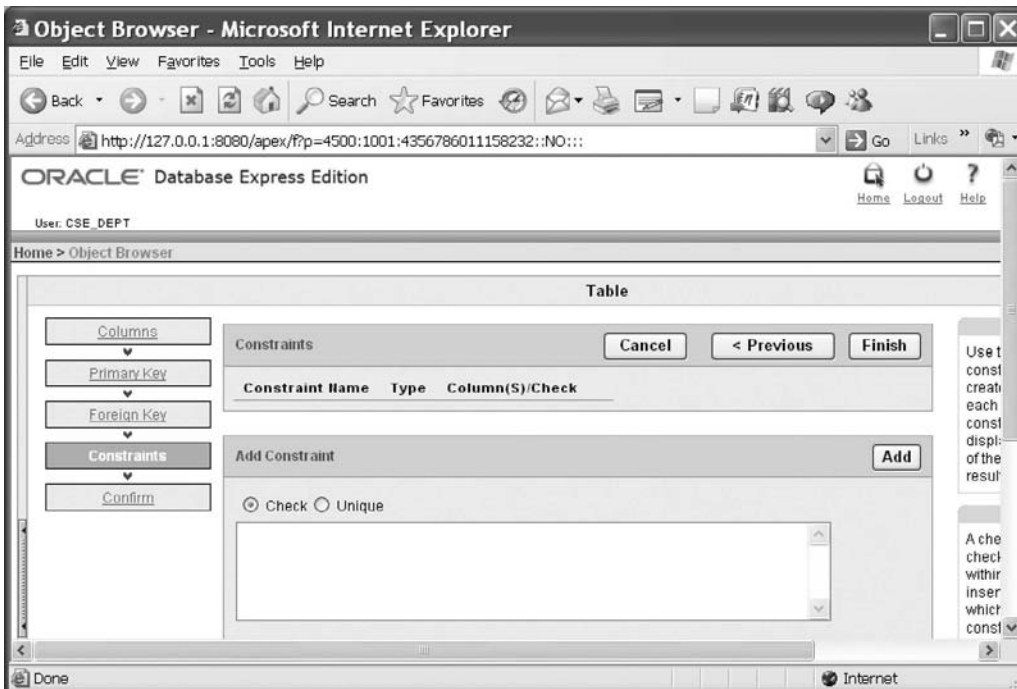


Figure 2.46 Fourth step—set up constraints.

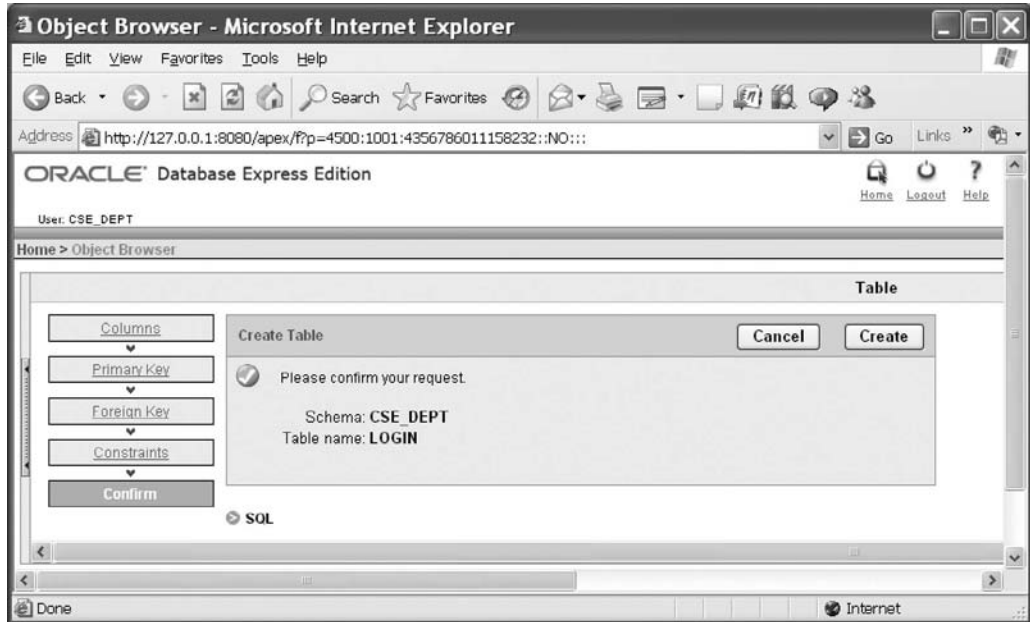


Figure 2.47 Last step—confirmation.

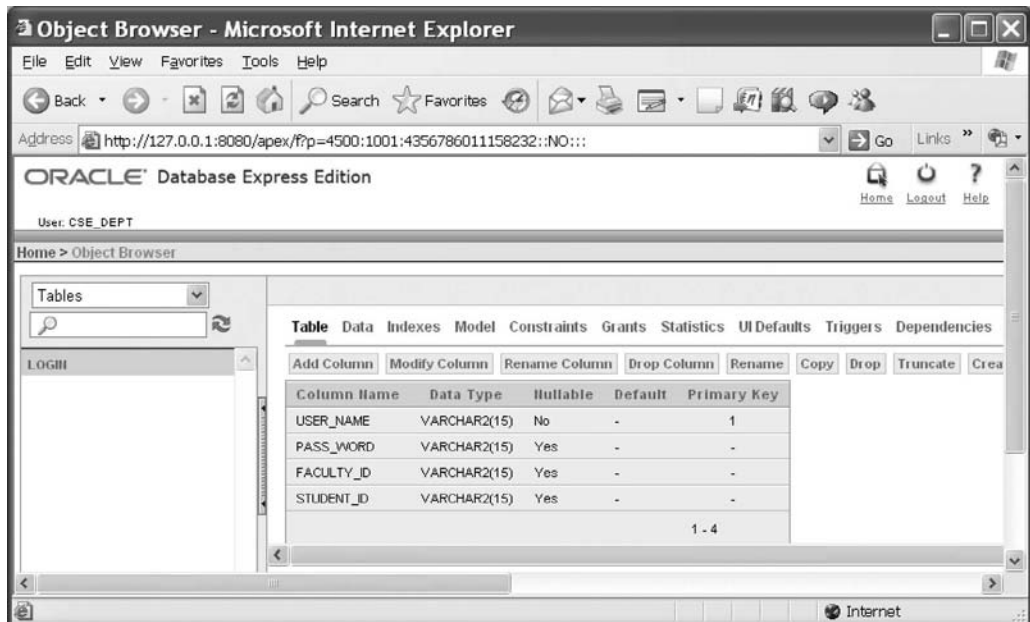


Figure 2.48 Created LogIn table.

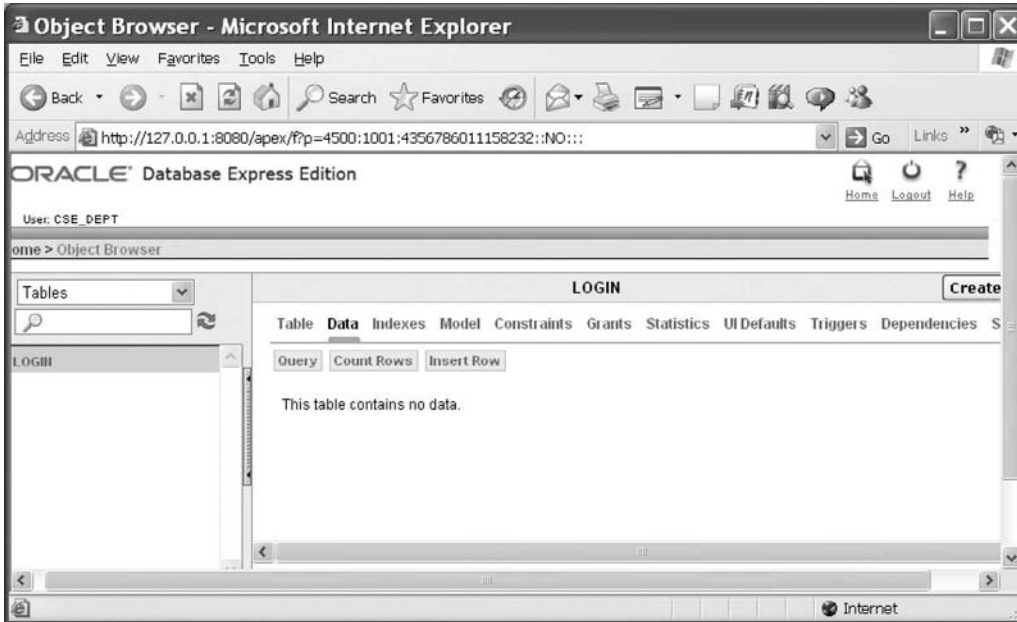


Figure 2.49 Opened Data page.

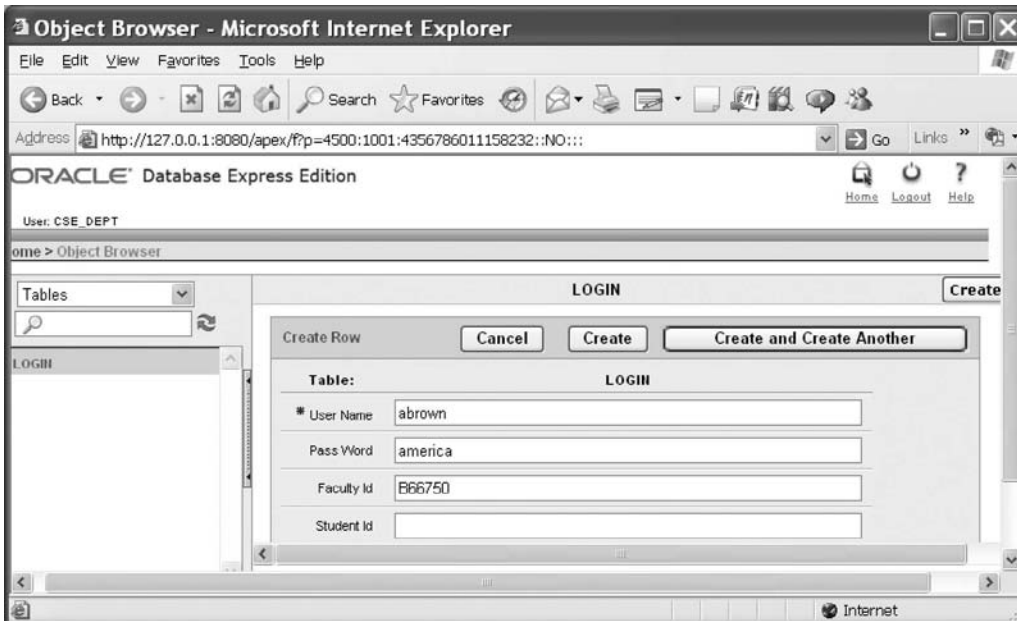


Figure 2.50 Opened data sheet view of the LogIn table.

Table 2.23 Data in the LogIn Table

user_name	pass_word	faculty_id	student_id
abrown	america	B66750	
ajade	tryagain		A97850
awoods	smart		A78835
banderson	birthday	A52990	
bvalley	see		B92996
dangles	tomorrow	A77587	
hsmith	try		H10210
jerica	excellent		J77896
jhenry	test	H99118	
jking	goodman	K69880	
sbhalla	india	B86590	
sjohnson	jermany	J33486	
ybai	reback	B78880	

Add the following data into the first row: User Name—abrown, Pass Word—america, Faculty Id—B66750. Since this user is a faculty, leave the Student Id row blank (**don't place a NULL in here, otherwise you will have trouble when you create a foreign key for this table later!**). Your finished first row is shown in Figure 2.50.

Click on the Create and Create Another button to create the next row. In a similar way, add each row that is shown in Table 2.23 into each row in the LogIn table.

You can click on the Create button after you add the final row into your table. Your finished LogIn table should match the one shown in Figure 2.51.

Next let's create our second table—Faculty table.

2.11.2.2 Create Faculty Table

Click on the Table tool on the top row and click on the Create button to create another new table. Select the Table item to open a new table page. Enter Faculty into the Table Name box as the name for this new table, and enter the following columns into this new table:

- faculty_id—VARCHAR2(10)
- faculty_name—VARCHAR2 (20)
- office—VARCHAR2 (10)
- phone—CHAR(12)
- college—VARCHAR2 (50)
- title—VARCHAR2 (30)
- email—VARCHAR2 (30)

The popular data types used in the Oracle database include NUMBER, CHAR, and VARCHAR2. Each data type has its upper bound and low bound. The difference between the CHAR and the VARCHAR2 is that the former is used to store a fixed-length string and the latter can provide a varying-length string, which means that the real

The screenshot shows a Microsoft Internet Explorer window titled 'Object Browser - Microsoft Internet Explorer'. The address bar contains the URL: `http://127.0.0.1:8080/apex/?p=4500:1001:4356786011158232::NO:::`. The main content area displays a table with the following data:

EDIT	USER_NAME	PASS_WORD	FACULTY_ID	STUDENT_ID
	abrown	america	B66750	-
	ajade	tryagain	-	A97850
	dangles	tomorrow	A77587	-
	hsmith	try	-	H10210
	jerica	excellent	-	J77896
	jhenry	test	H99118	-
	awoods	smart	-	A76835
	banderson	birthday	A52990	-
	bvalley	see	-	B92996
	ikling	goodman	K69880	-
	sbhalla	india	B86590	-
	sjohnson	jernany	J33486	-
	ybai	reback	B78880	-

Figure 2.51 Completed LogIn table.

length of the string depends on the number of real letters entered by the user. The data types for all columns are VARCHAR2 with one exception, which is the phone column that has a CHAR type with an upper bound of 12 letters since our phone numbers are composed of 10 digits, and we can extend this length to 12 with two dashes. For all other columns, the length varies with the different information, so the VARCHAR2 is selected for those columns.

The finished design view of your Faculty table is shown in Figure 2.52. You need to check the Not Null checkbox for the faculty_id column since we selected this column as the primary key for this table.

Click on the Next button to go to the next page to add the primary key for this table, which is shown in Figure 2.53.

Check the Not Populated from the primary key list since we don't want to use any Sequence object to automatically generate a sequence of numeric numbers as our primary key, and then select the FACULTY_ID(VARCHAR2) from the Primary Key textbox. In this way, the faculty_id column is selected as the primary key for this table. Keep the Composite Primary Key box untouched since we do not have that kind of key in this table, and click on the Next button to go to the next page.

Since we have not created all other tables to work as our reference tables for the foreign key, click on Next to continue, and we will do the foreign key for this table later. Click on the Finish button to go to the Confirm page. Finally click on the Create button to create this new Faculty table. Your completed columns in the Faculty table are shown in Figure 2.54.

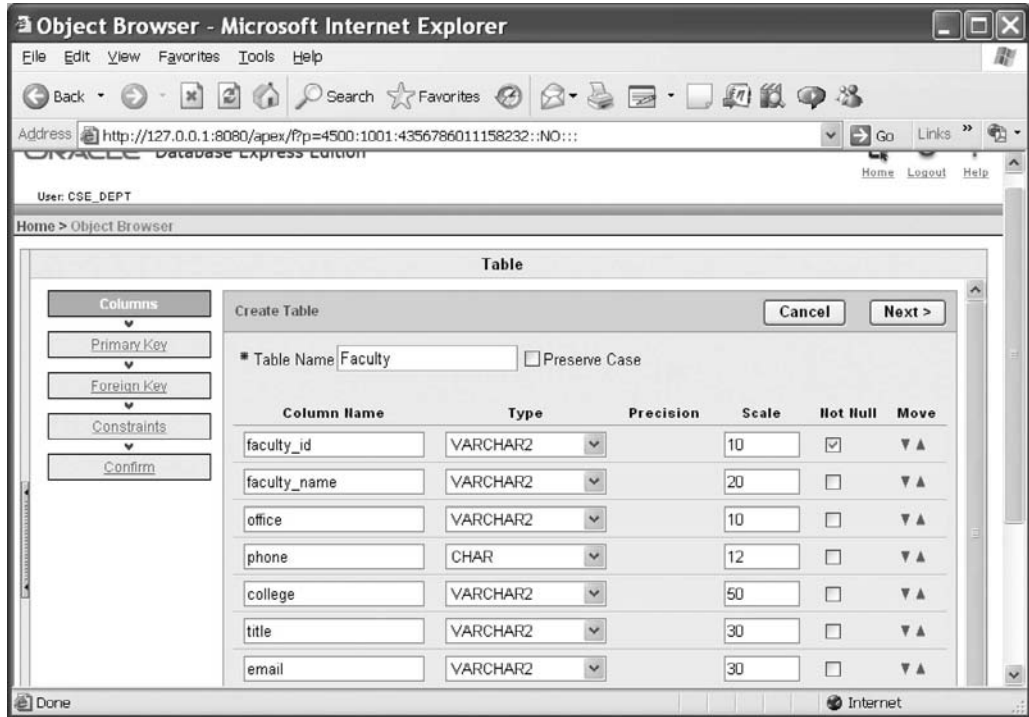


Figure 2.52 Finished design view of the Faculty table.



Figure 2.53 Opened Primary Key window.

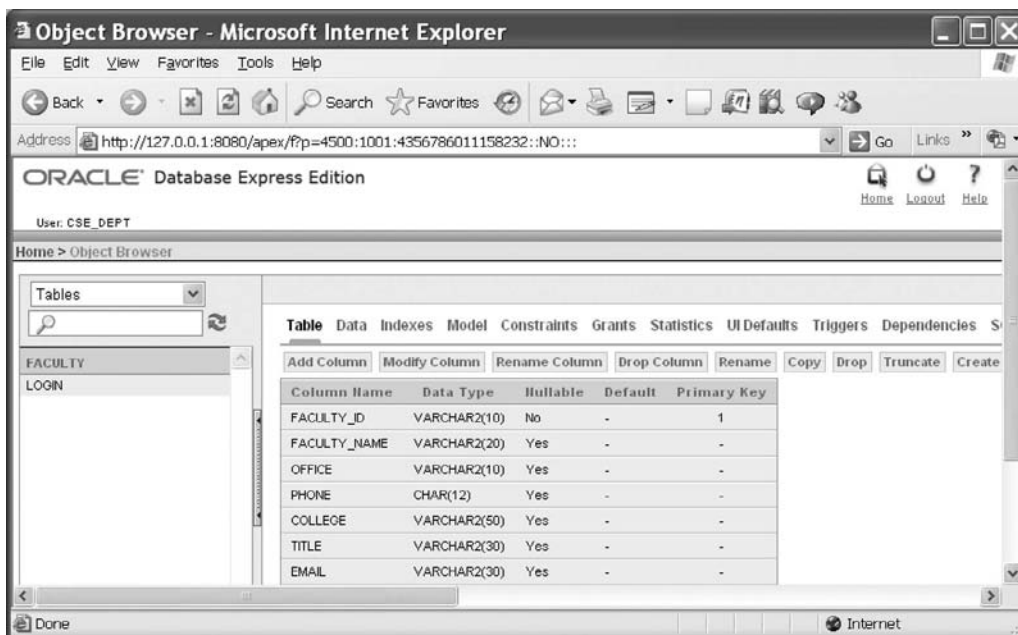


Figure 2.54 Completed columns in the Faculty table.

Table 2.24 Data in the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.edu
A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@college.edu
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

Now click on the Data object tool to add the data into this new table. Click on the Insert Row button to add all rows that are shown in Table 2.24 into this table.

Click on the Create and Create Another button when the first row is done, and continue to create all rows with the data shown in Table 2.24. You may click on the Create button for your last row. Your finished Faculty table should match the one shown in Figure 2.55.

2.11.2.3 Create Other Tables

In a similar way, you can continue to create the following three tables: Course, Student, and StudentCourse based on the data shown in Tables 2.25, 2.26, and 2.27.

The screenshot shows a web browser window titled 'Object Browser - Microsoft Internet Explorer'. The address bar shows a URL starting with 'http://127.0.0.1:8080/apex/f?p=4500:1001:4356796011158232::NO::'. The main content area displays a table with the following data:

EDIT	FACULTY_ID	FACULTY_NAME	OFFICE	PHONE	COLLEGE	TITLE	EMAIL
	B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu
	B96590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
	K89880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu
	A52990	Black Anderson	MTC-218	750-378-9997	Virginia Tech	Professor	banderson@college.edu
	H99118	Jeff Henry	MTC-338	750-330-8850	Ohio State University	Associate Professor	jhenry@college.edu
	A77507	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
	B66750	Alice Brown	MTC-257	750-330-8850	University of Florida	Assistant Professor	abrown@college.edu
	J33496	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu

Figure 2.55 Finished Faculty table.

The data types used in the Course table are:

- course_id: VARCHAR2(10)—primary Key
- course: VARCHAR2(40)
- credit: NUMBER(1, 0)—precision = 1, scale = 0 (1-bit integer)
- classroom: CHAR(6)
- schedule: VARCHAR2(40)
- enrollment: NUMBER(2, 0)—precision = 2, scale = 0 (2-bit integer)
- faculty_id: VARCHAR2(10)

The data types used in the Student table are:

- student_id: VARCHAR2(10)—primary Key
- student_name: VARCHAR2(20)
- gpa: NUMBER(3, 2)—precision = 3, scale = 2 (3-bit floating point data with 2-bit after the decimal point)
- credits: NUMBER(3, 0)—precision = 3, scale = 0 (3-bit integer)
- major: VARCHAR2(40)
- schoolYear: VARCHAR2(20)
- email: VARCHAR2(20)

The data types used in the StudentCourse table are:

- s_course_id: NUMBER(4, 0)—precision = 4, scale = 0 (4-bit integer) primary key
- student_id: VARCHAR2(10)

Table 2.25 Data in the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-131A	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	28	A52990
CSC-131B	Computers in Society	3	TC-114	M-W-F: 9:00-9:55 AM	20	B66750
CSC-131C	Computers in Society	3	TC-109	T-H: 11:00-12:25 PM	25	A52990
CSC-131D	Computers in Society	3	TC-119	M-W-F: 9:00-9:55 AM	30	B86590
CSC-131E	Computers in Society	3	TC-301	M-W-F: 1:00-1:55 PM	25	B66750
CSC-131F	Computers in Society	3	TC-109	T-H: 1:00-2:25 PM	32	A52990
CSC-132A	Introduction to Programming	3	TC-303	M-W-F: 9:00-9:55 AM	21	J33486
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-230	Algorithms & Structures	3	TC-301	M-W-F: 1:00-1:55 PM	20	A77587
CSC-232A	Programming I	3	TC-305	T-H: 11:00-12:25 PM	28	B66750
CSC-232B	Programming I	3	TC-303	T-H: 11:00-12:25 PM	17	A77587
CSC-233A	Introduction to Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	18	H99118
CSC-233B	Introduction to Algorithms	3	TC-302	M-W-F: 11:00-11:55 AM	19	K69880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSC-234B	Data Structure & Algorithms	3	TC-114	T-H: 11:00-12:25 PM	15	J33486
CSC-242	Programming II	3	TC-303	T-H: 1:00-2:25 PM	18	A52990
CSC-320	Object Oriented Programming	3	TC-301	T-H: 1:00-2:25 PM	22	B66750
CSC-331	Applications Programming	3	TC-109	T-H: 11:00-12:25 PM	28	H99118
CSC-333A	Computer Arch & Algorithms	3	TC-301	M-W-F: 10:00-10:55 AM	22	A77587
CSC-333B	Computer Arch & Algorithms	3	TC-302	T-H: 11:00-12:25 PM	15	A77587
CSC-335	Internet Programming	3	TC-303	M-W-F: 1:00-1:55 PM	25	B66750
CSC-432	Discrete Algorithms	3	TC-206	T-H: 11:00-12:25 PM	20	B86590
CSC-439	Database Systems	3	TC-206	M-W-F: 1:00-1:55 PM	18	B86590
CSE-138A	Introduction to CSE	3	TC-301	T-H: 1:00-2:25 PM	15	A52990
CSE-138B	Introduction to CSE	3	TC-109	T-H: 1:00-2:25 PM	35	J33486
CSE-330	Digital Logic Circuits	3	TC-305	M-W-F: 9:00-9:55 AM	26	K69880
CSE-332	Foundations of Semiconductors	3	TC-305	T-H: 1:00-2:25 PM	24	K69880
CSE-334	Elec Measurement & Design	3	TC-212	T-H: 11:00-12:25 PM	25	H99118
CSE-430	Bioinformatics in Computer	3	TC-206	Thu: 9:30-11:00 AM	16	B86590
CSE-432	Analog Circuits Design	3	TC-309	M-W-F: 2:00-2:55 PM	18	K69880
CSE-433	Digital Signal Processing	3	TC-206	T-H: 2:00-3:25 PM	18	H99118
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-436	Automatic Control and Design	3	TC-305	M-W-F: 10:00-10:55 AM	29	J33486
CSE-437	Operating Systems	3	TC-303	T-H: 1:00-2:25 PM	17	A77587
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880
CSE-439	Special Topics in CSE	3	TC-206	M-W-F: 10:00-10:55 AM	22	J33486

Table 2.26 Data in the Student Table

student_id	student_name	gpa	credits	major	schoolYear	email
A78835	Andrew Woods	3.26	108	Computer Science	Senior	awoods@college.edu
A97850	Ashly Jade	3.57	116	Information System Engineering	Junior	ajade@college.edu
B92996	Blue Valley	3.52	102	Computer Science	Senior	bvalley@college.edu
H10210	Holes Smith	3.87	78	Computer Engineering	Sophomore	hsmith@college.edu
J77896	Erica Johnson	3.95	127	Computer Science	Senior	ejohnson@college.edu

Table 2.27 Data in the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1000	H10210	CSC-131D	3	CE
1001	B92996	CSC-132A	3	CS/IS
1002	J77896	CSC-335	3	CS/IS
1003	A78835	CSC-331	3	CE
1004	H10210	CSC-234B	3	CE
1005	J77896	CSC-234A	3	CS/IS
1006	B92996	CSC-233A	3	CS/IS
1007	A78835	CSC-132A	3	CE
1008	A78835	CSE-432	3	CE
1009	A78835	CSE-434	3	CE
1010	J77896	CSC-439	3	CS/IS
1011	H10210	CSC-132A	3	CE
1012	H10210	CSC-331	3	CE
1013	A78835	CSC-335	3	CE
1014	A78835	CSE-438	3	CE
1015	J77896	CSC-432	3	CS/IS
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE
1018	A97850	CSC-331	3	ISE
1019	A97850	CSC-335	3	ISE
1020	J77896	CSE-439	3	CS/IS
1021	B92996	CSC-230	3	CS/IS
1022	A78835	CSE-332	3	CE
1023	B92996	CSE-430	3	CE
1024	J77896	CSC-333A	3	CS/IS
1025	H10210	CSE-433	3	CE
1026	H10210	CSE-334	3	CE
1027	B92996	CSC-131C	3	CS/IS
1028	B92996	CSC-439	3	CS/IS

- course_id: VARCHAR2(10)
- credit: NUMBER(1, 0)—precision = 1, scale = 0 (1-bit integer)
- major: VARCHAR2(40)

Your finished Course, Student, and StudentCourse tables are shown in Figures 2.56, 2.57 and 2.58, respectively.

2.11.3 Create Constraints Between Tables

Now it is the time for us to set up the relationships between our five tables using the primary and foreign keys. Since we have already selected the primary key for each table when we create and build those tables, therefore we only need to take care of the foreign keys and connect them with the associated primary keys in the related tables. Let's start from the first table, LogIn table.

The screenshot shows the Object Browser interface in Microsoft Internet Explorer. The left sidebar lists 'COURSE', 'FACULTY', and 'LOGIN'. The main window displays the 'Table Data' view for the 'COURSE' table. The table has columns: EDIT, COURSE_ID, COURSE, CREDIT, CLASSROOM, SCHEDULE, ENROLLMENT, and FACULTY_ID. The data is as follows:

EDIT	COURSE_ID	COURSE	CREDIT	CLASSROOM	SCHEDULE	ENROLLMENT	FACULTY_ID
	CSC-242	Programming II	3	TC-303	T-H 1:00-2:25 PM	18	A52990
	CSC-331	Applications Programming	3	TC-109	T-Ht 11:00-12:25 PM	26	H99118
	CSC-333A	Computer Arch & Algorithms	3	TC-301	M-W-F: 10:00-10:55 AM	22	A77587
	CSC-439	Database Systems	3	TC-206	M-W-F: 1:00-1:55 PM	18	B86590
	CSE-332	Foundations of Semiconductors	3	TC-305	T-H 1:00-2:25 PM	24	K69880
	CSC-131A	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	26	A52990
	CSC-131F	Computers in Society	3	TC-109	T-H 1:00-2:25 PM	32	A52990
	CSC-132A	Introduction to Programming	3	TC-303	M-W-F: 9:00-9:55 AM	21	J33486
	CSC-320	Object Oriented Programming	3	TC-301	T-Ht 1:00-2:25 PM	22	B66750
	CSC-131E	Computers in Society	3	TC-301	M-W-F: 1:00-1:55 PM	25	B66750
	CSC-233A	Introduction to Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	18	H99118
	CSE-436	Automatic Control and Design	3	TC-305	M-W-F: 10:00-10:55 AM	29	J33486
	CSE-437	Operating Systems	3	TC-303	T-Ht 1:00-2:25 PM	17	A77587
	CSC-131B	Computers in Society	3	TC-114	M-W-F: 9:00-9:55 AM	20	B66750
	CSC-131C	Computers in Society	3	TC-109	T-H 11:00-12:25 PM	25	A52990

row(s) 1 - 15 of 38

Figure 2.56 Completed Course table.

The screenshot shows the Object Browser interface in Microsoft Internet Explorer. The left sidebar lists 'FACULTY', 'LOGIN', and 'STUDENT'. The main window displays the 'Table Data' view for the 'STUDENT' table. The table has columns: EDIT, STUDENT_ID, STUDENT_NAME, GPA, CREDITS, MAJOR, SCHOOLYEAR, and EMAIL. The data is as follows:

EDIT	STUDENT_ID	STUDENT_NAME	GPA	CREDITS	MAJOR	SCHOOLYEAR	EMAIL
	J77896	Erica Johnson	3.95	127	Computer Science	Senior	ejohnson@college.edu
	A78835	Andrew Woods	3.26	108	Computer Science	Senior	awoods@college.edu
	A97650	Ashly Jade	3.57	116	Information System Engineering	Junior	ejade@college.edu
	B92996	Blue Valley	3.52	102	Computer Science	Senior	bvalley@college.edu
	H10210	Holes Smith	3.87	78	Computer Engineering	Sophomore	hsmith@college.edu

row(s) 1 - 5 of 5

Figure 2.57 Completed Student table.

The screenshot shows a web browser window titled "Object Browser - Microsoft Internet Explorer". The address bar shows a URL: `http://127.0.0.1:8080/apex/?p=4500:1001:4356786011158232:NO::`. The browser displays a table with the following data:

EDIT	S_COURSE_ID	STUDEHT_ID	COURSE_ID	CREDIT	MAJOR
	1001	B92996	CSC-132A	3	CS/S
	1002	J77896	CSC-335	3	CS/S
	1003	A78835	CSC-331	3	CE
	1010	J77896	CSC-439	3	CS/S
	1014	A78835	CSE-438	3	CE
	1016	A97850	CSC-132B	3	ISE
	1021	B92996	CSC-230	3	CS/S
	1026	H10210	CSE-334	3	CE
	1027	B92996	CSC-131C	3	CS/S
	1028	B92996	CSC-439	3	CS/S
	1008	A78835	CSE-432	3	CE
	1012	H10210	CSC-331	3	CE
	1017	A97850	CSC-234A	3	ISE
	1018	A97850	CSC-331	3	ISE

Figure 2.58 Completed StudentCourse table.

2.11.3.1 Create Constraints Between LogIn and Faculty Tables

Now let's create the constraints between the LogIn and the Faculty tables by using a foreign key. Create a foreign key for the LogIn table and connect it to the primary key in the Faculty table. The faculty_id is a foreign key in the LogIn table but it is a primary key in the Faculty table. A one-to-many relationship exists between the faculty_id in the Faculty table and the faculty_id in the LogIn table.

Log on the Oracle Database 10g XE using the customer user name, CSE_DEPT, and the customer database password, and then open the home page of the Oracle Database 10g XE. Click on the Object Browser icon and select Browse/Table to list all tables. Select the LogIn table from the left pane to open it, click the Constraints tab and then click the Create button that is the first button in the second row. Enter LOGIN_FACULTY_FK into the Constraint Name box, and select the Foreign Key from the Constraint Type box, which is shown in Figure 2.59. Check the On Delete Cascade checkbox. Then select the FACULTY_ID from the LogIn table as the foreign key column. Select the FACULTY

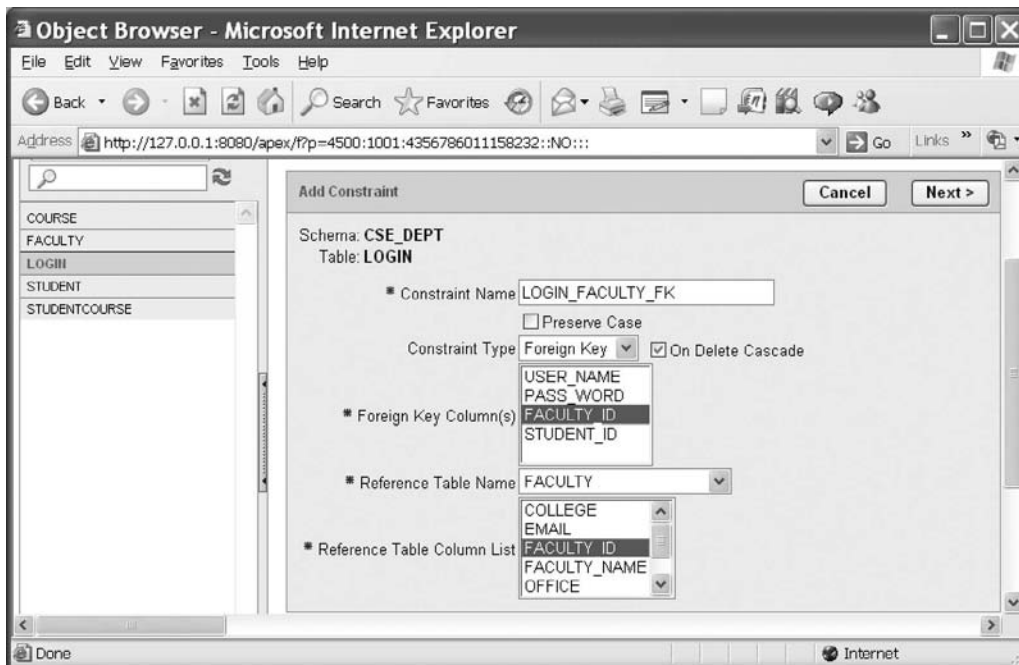


Figure 2.59 Create the foreign key between the LogIn and the Faculty table.

table from the Reference Table Name box as the reference table, and select the `FACULTY_ID` from the Reference Table Column List as the reference table column. Your finished Add Constraint window should match the one shown in Figure 2.59.

Click on Next to go to the next window, and then click on the Finish button to confirm this foreign key's creation.

2.11.3.2 Create Constraints Between LogIn and Student Tables

The relationship between the Student table and the LogIn table is a one-to-many relationship. The `student_id` in the Student table is a primary key, but the `student_id` in the LogIn table is a foreign key. Multiple `student_ids` can exist in the LogIn table, but only one or a unique `student_id` can be found from the Student table.

To create a foreign key from the LogIn table and connect it to the primary key in the Student table, open the LogIn table if it is not opened and click on the Constraints tab, and then click on the Create button that is the first button in the second row to open the Add Constraint window. Enter `LOGIN_STUDENT_FK` into the Constraint Name box, and select the Foreign Key from the Constraint Type box, which is shown in Figure 2.60. Check the On Delete Cascade checkbox. Then select the `STUDENT_ID` from the LogIn table as the foreign key column. Select the `STUDENT` table from the Reference Table Name box as the reference table, and select the `STUDENT_ID` from the Reference Table Column List as the reference table column. Your finished Add Constraint window should match the one shown in Figure 2.60.

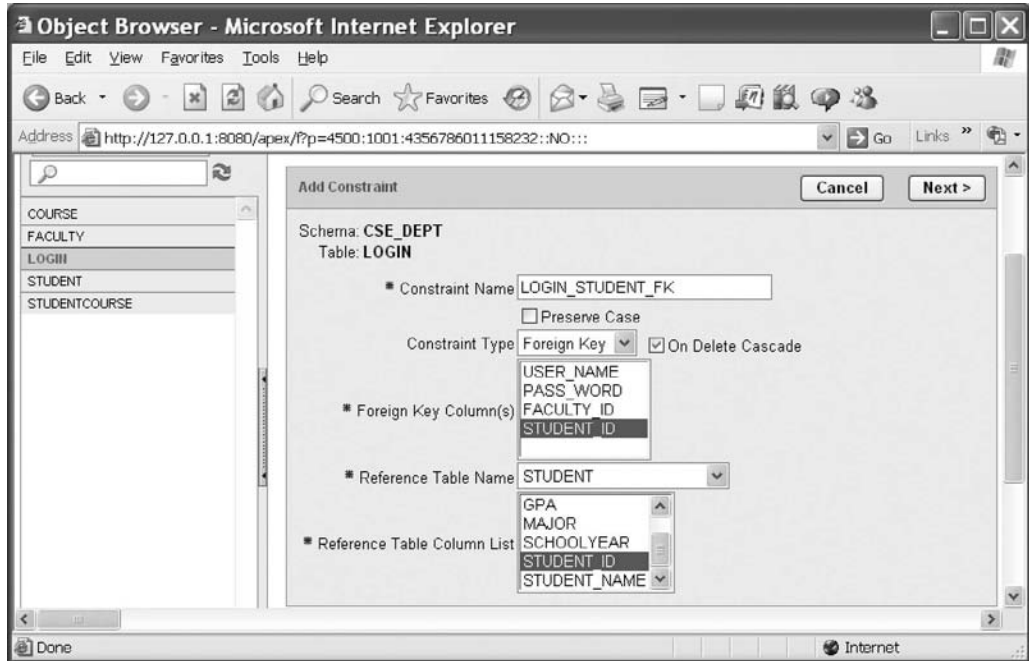


Figure 2.60 Create the foreign key between the LogIn and the Student table.

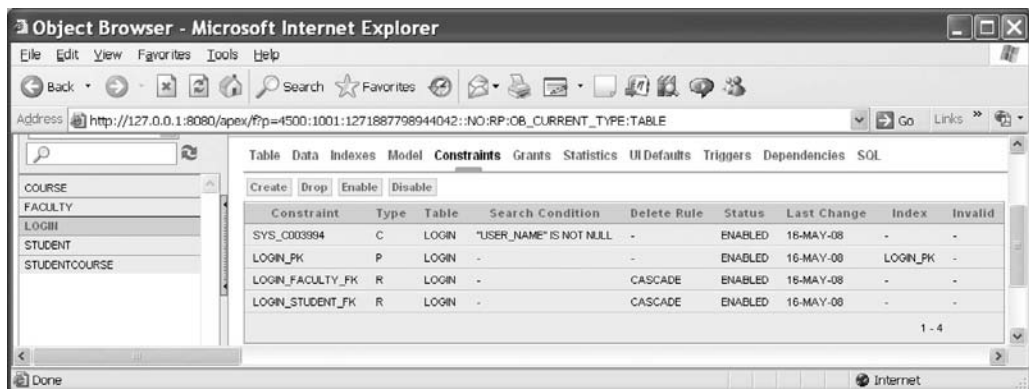


Figure 2.61 Finished foreign key creation window for the LogIn table.

Recall that when we created the LogIn table in Section 2.11.2.1, we emphasized that for the blank fields in both `faculty_id` and `student_id` columns, don't place a NULL into these fields and just leave those fields blank. The reason for this is that an ALTER TABLE command will be issued when you create a foreign key for the LogIn table, and the NULL cannot be recognized by this command, therefore an error ORA-02298 occurs and your creation of a foreign key will fail.

Click on the Next button to go to the next window, and then click on the Finish button to confirm this foreign key's creation. Your finished foreign key creation window for the LogIn table should match the one shown in Figure 2.61.

2.11.3.3 Create Constraints Between Course and Faculty Tables

The relationship between the Faculty table and the Course table is a one-to-many relationship. The `faculty_id` in the Faculty table is a primary key, but it is a foreign key in the Course table. This means that only a unique `faculty_id` exists in the Faculty table but multiple `faculty_ids` can exist in the Course table since one faculty can teach multiple courses.

Open the Course table by clicking it from the left panel. Click on the Constraints tab and then click the Create button. Enter `COURSE_FACULTY_FK` into the Constraint Name box, and select the Foreign Key from the Constraint Type box, which is shown in Figure 2.62. Check the On Delete Cascade checkbox. Then select the `FACULTY_ID` from the Course table as the foreign key column. Select the `FACULTY` table from the Reference Table Name box as the reference table, and select the `FACULTY_ID` from the Reference Table Column List as the reference table column. Your finished Add Constraint window should match the one shown in Figure 2.62.

Click on the Next button to go to the next window, and then click on the Finish button to confirm this foreign key's creation. Your finished foreign key creation window for the Course table should match the one shown in Figure 2.63.

2.11.3.4 Create Constraints Between StudentCourse and Student Tables

The relationship between the Student table and the StudentCourse table is a one-to-many relationship. The primary key `student_id` in the Student table is a foreign key in the

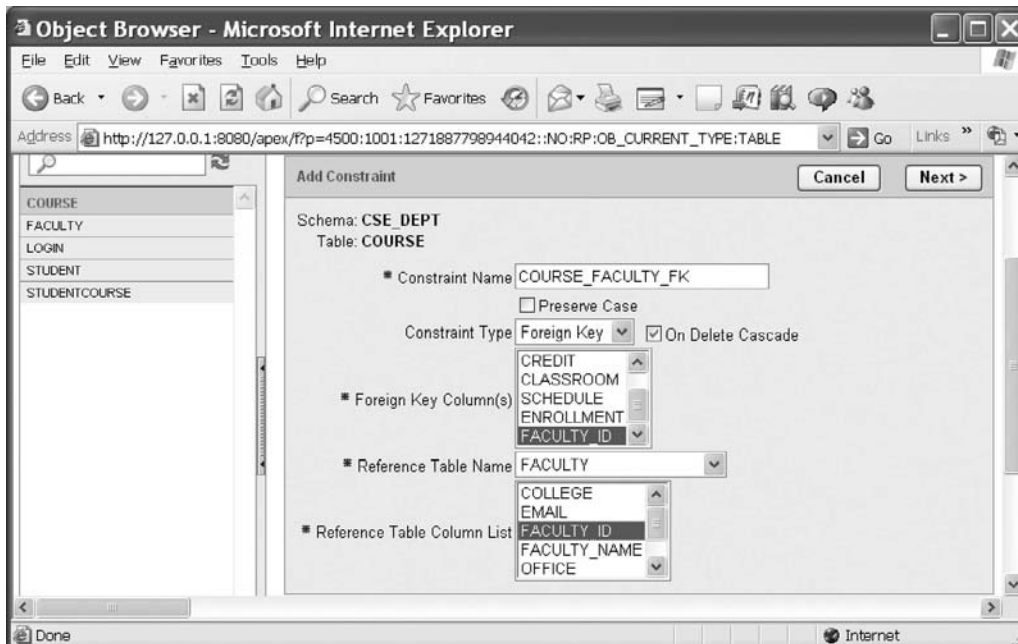


Figure 2.62 Create the foreign key between the Course and the Faculty table.

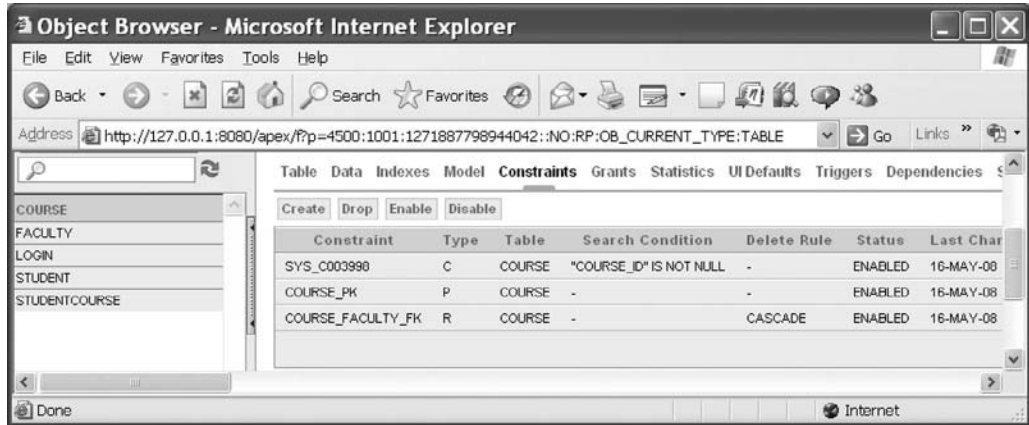


Figure 2.63 Finished foreign key creation window for the Course table.

StudentCourse table since one student can take multiple different courses. In order to create this relationship by using the foreign key, first let's open the StudentCourse table.

Click on the Constraints tab and then click the Create button that is the first button on the second row. Enter STUDENTCOURSE_STUDENT_FK into the Constraint Name box, and select the Foreign Key from the Constraint Type box, which is shown in Figure 2.64. Check the On Delete Cascade checkbox. Then select the STUDENT_ID from the StudentCourse table as the foreign key column. Select the STUDENT table from the Reference Table Name box as the reference table, and select the STUDENT_ID from the Reference Table Column List as the reference table column. Your finished Add Constraint window should match the one shown in Figure 2.64.

Click on the Next button to go to the next window, and then click on the Finish button to confirm this foreign key's creation.

2.11.3.5 Create Constraints Between StudentCourse and Course Tables

The relationship between the Course table and the StudentCourse table is a one-to-many relationship. The primary key `course_id` in the Course table is a foreign key in the StudentCourse table since one course can be taken by multiple different students. By using the StudentCourse table as an intermediate table, a many-to-many relationship can be built between the Student table and the Course table.

To create this relationship by using the foreign key, open the StudentCourse table by clicking on it from the left pane. Click the Constraints tab and then click on the Create button that is the first button on the second row. Enter STUDENTCOURSE_COURSE_FK into the Constraint Name box, and select the Foreign Key from the Constraint Type box, which is shown in Figure 2.65. Check the On Delete Cascade checkbox. Then select the COURSE_ID from the StudentCourse table as the foreign key column. Select the COURSE table from the Reference Table Name box as the reference table, and select the COURSE_ID from the Reference Table Column List as the

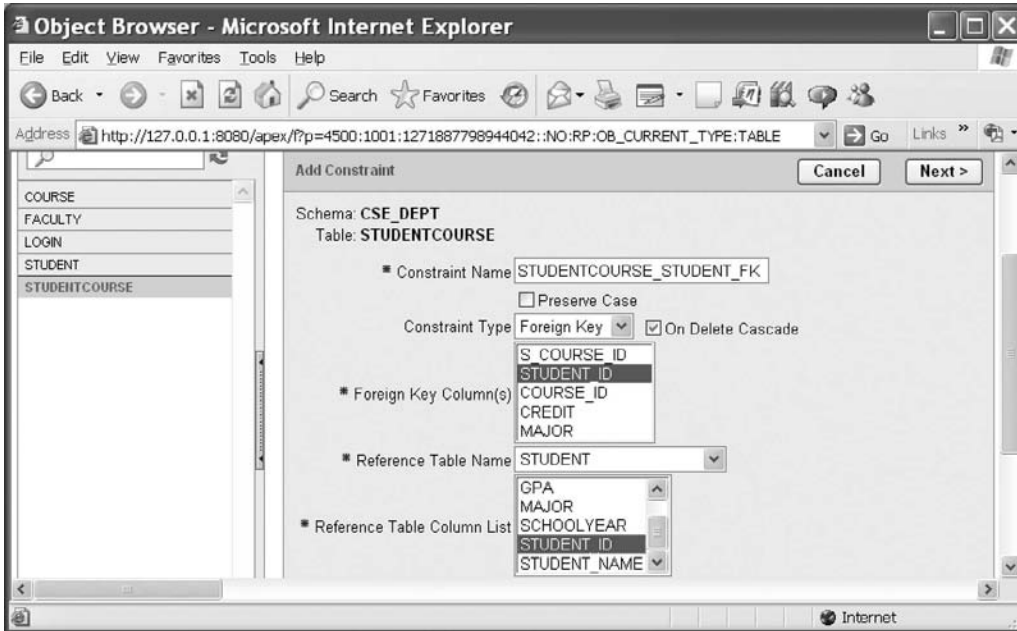


Figure 2.64 Create the foreign key between the StudentCourse and the Student table.

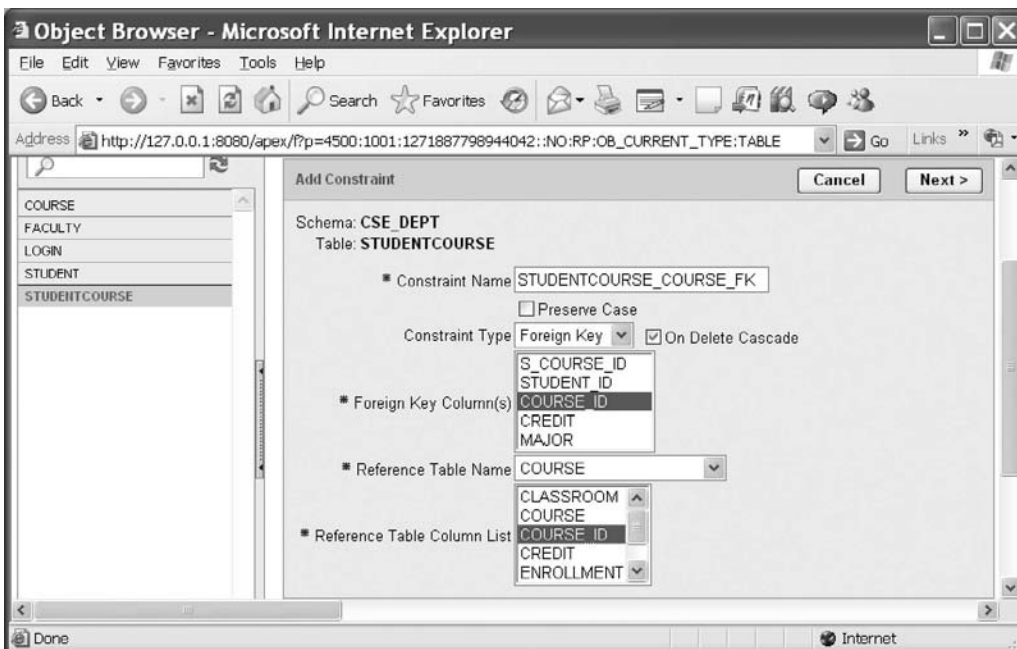


Figure 2.65 Create the foreign key between the StudentCourse and the Course table.

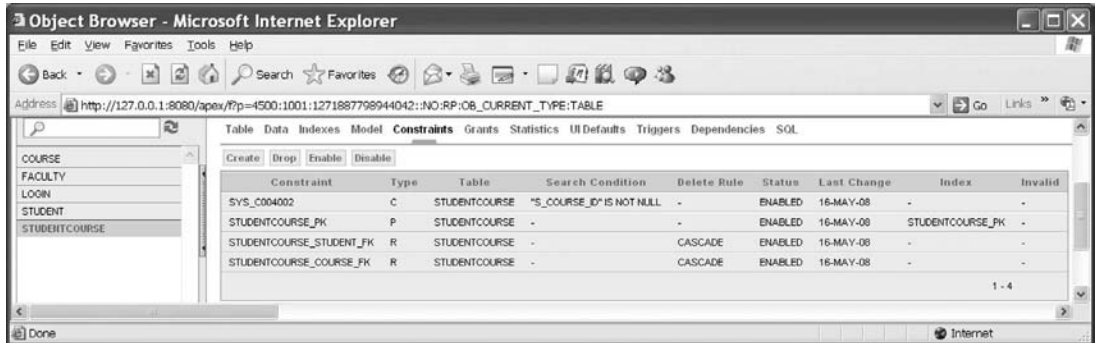


Figure 2.66 Finished foreign key creation window for the StudentCourse table.

reference table column. Your finished Add Constraint window should match the one shown in Figure 2.65.

Click on the Next button to go to the next window, and then click on the Finish button to confirm this foreign key's creation. Your finished foreign key creation window for the StudentCourse table should match the one shown in Figure 2.66.

Our customer database creation for Oracle Database 10g Express Edition is completed. A completed Oracle 10g XE sample database CSE_DEPT that is represented by a group of table files can be found in the folder **Oracle** located at the accompanying ftp web site (see Chapter 1). Refer to Appendix F to get more detailed information if you want to use this sample database in your Visual C# applications.

At this point, we have finished developing and creating all sample databases we need to use later. All of these sample databases will be utilized for different applications we will develop in this book.

Since the Oracle Database 10g XE is very different with other databases such as Microsoft Access and SQL Server 2005, you need to refer to Appendix F to get a clear picture about how to use the CSE_DEPT Oracle database files. Refer to Appendix D to get the knowledge on how to use the Utilities of Oracle Database 10g XE to Unload the five tables to five Text files, and how to Load those five table files into a new customer Oracle database to create a new customer Oracle database easily.

2.12 CHAPTER SUMMARY

A detailed discussion and analysis of the structure and components about databases were provided in this chapter. Some key technologies in developing and designing databases were also given and discussed. The procedures and components to develop a relational database were analyzed in detailed with some real data tables in our sample database CSE_DEPT. The process in developing and building a sample database was discussed in detail with the following points:

- Defining relationships
- Normalizing the data
- Implementing the relational database

In the second part of this chapter, three sample databases that were developed with three popular database management systems such as Microsoft Access, SQL Server 2005, and Oracle Database 10g XE were provided in detail. All of these three sample databases will be used throughout the whole book.

HOMWORK

I. True/False Selections

- ___ 1. Database development process involves project planning, problem analysis, logical design, physical design, implementation, and maintenance.
- ___ 2. Duplication of data creates problems with data integrity.
- ___ 3. If the primary key consists of a single column, then the table in 1NF is automatically in 2NF.
- ___ 4. A table is in first normal form if there are no repeating groups of data in any column.
- ___ 5. When a user perceives the database as made up of tables, it is called a Network Model.
- ___ 6. *Entity integrity rule* states that no attribute that is a member of the primary (composite) key may accept a null value.
- ___ 7. When creating data tables for the Microsoft Access database, a blank field can be kept as a blank without any letter in it.
- ___ 8. To create data tables in SQL Server database, a blank field can be kept as a blank without any letter in it.
- ___ 9. The name of each data table in SQL Server database must be prefixed by the keyword `dbo`.
- ___ 10. The Sequence object in Oracle database is used to automatically create a sequence of numeric numbers that work as the primary keys.

II. Multiple Choices

1. There are many advantages to using an integrated database approach over that of a file processing approach. These include
 - a. Minimizing data redundancy
 - b. Improving security
 - c. Data independence
 - d. All of the above
2. The entity integrity rule implies that no attribute that is a member of the primary key may accept _____.
 - a. Null value
 - b. Integer data type
 - c. Character data type
 - d. Real data type
3. Reducing data redundancy will lead to _____.
 - a. Deletion anomalies
 - b. Data consistency
 - c. Loss of efficiency
 - d. None of the above

4. _____ keys are used to create relationships among various tables in a database.
 - a. Primary keys
 - b. Candidate keys
 - c. Foreign keys
 - d. Composite keys
5. In a small university the Department of Computer Science has six faculty members. However, each faculty member belongs to only the Computer Science Department. This type of relationship is called _____.
 - a. One-to-one
 - b. One-to-many
 - c. Many-to-many
 - d. None of the above
6. The Client Server databases have several advantages over the File Server databases. These include _____.
 - a. Minimizing chances of crashes
 - b. Provision of features for recovery
 - c. Enforcement of security
 - d. Efficient use of the network
 - e. All of the above
7. One can create the foreign keys between tables _____.
 - a. Before any table can be created
 - b. When some tables are created
 - c. After all tables are created
 - d. With no limitations
8. To create foreign keys between tables, first one must select the table that contains a _____ key and then select another table that has a _____ key.
 - a. Primary, foreign
 - b. Primary, primary
 - c. Foreign, primary
 - d. Foreign, foreign
9. The data type VARCHAR2 in Oracle database is a string variable with _____.
 - a. Limited length
 - b. Fixed length
 - c. Certain number of letters
 - d. Varying length
10. For data tables in Oracle Database 10g XE, a blank field must be _____.
 - a. Indicated by NULL
 - b. Kept as a blank
 - c. Either by NULL or a blank
 - d. Avoided

III. Exercises

1. What are the advantages to using an integrated database approach over that of a file processing approach?
2. Define entity integrity and referential integrity. Describe the reasons for enforcing these rules.
3. Entities can have three types of relationships: *one-to-one*, *one-to-many*, and *many-to-many*. Define each type of relationship. Draw ER diagrams to illustrate each type of relationship.
4. List all steps to create foreign keys between data tables for SQL Server database in the SQL Server Management Studio Express. Illustrate those steps by using a real example. For instance, how to create foreign keys between the LogIn and the Faculty table.
5. List all steps to create foreign keys between data tables for Oracle database in the Oracle Database 10g XE. Illustrate those steps by using a real example. For instance, how to create foreign keys between the StudentCourse and the Course table.

Chapter 3

Introduction to ADO.NET

It has been a long road for software developers to generate and implement sophisticated data processing techniques to improve and enhance data operations. The evolution of data access Application Programming Interface (API) is also a long process focusing predominantly on how to deal with relational data in a more flexible way. The methodology development has been focused on Microsoft-based APIs such as Open Database Connectivity (ODBC), OLEDB, Microsoft Jet, Data Access Objects (DAO), and Remote Data Objects (RDO), in addition to many non-Microsoft-based APIs. These APIs did not bridge the gap between object-based and semistructured (XML) data programming needs. Combine this problem with the task of dealing with many different data stores, nonrelational data like XML and applications applying across multiple languages are challenging topics, and one should have a tremendous opportunity for complete rearchitecture. The ADO.NET is a good solution for these challenges.

ADO.NET 2.0 was considered as a full solution between the relational database API and object-oriented data access API, which was released with the .NET Framework 2.0 and Visual Studio.NET 2005. Today the updated version of the ADO.NET is 3.5, which is released with the .NET Framework 3.5 and Visual Studio.NET 2008. In this chapter, first we will provide a detailed review about the history of ADO.NET, and then a full discussion and description about the components and architectures of ADO.NET 2.0 is given since most data components used for today's database actions are still covered by the ADO.NET 2.0. Finally we introduce some new features and components included in the ADO.NET 3.5.

3.1 ADO AND ADO.NET

ActiveX Data Object (ADO) is developed based on Object Linking and Embedding (OLE) and Component Object Model (COM) technologies. COM is used by developers to create reusable software components, link components together to build applications, and take advantage of Windows services. For the last decade, ADO has been the pre-

ferred interface for Visual Basic programmers to access various data sources, with ADO 2.7 being the latest version of this technology. The development history of data accessing methods can be traced back to the mid-1990s with DAO and then followed by RDO, which was based on ODBC. In the late 1990s, the ADO that is based on OLEDB was developed. This technology is widely applied in most object-oriented programming and database applications during the last decade.

ADO.NET 3.5 is the updated version of ADO.NET, and it is based mainly on the Microsoft .NET Framework 3.5. The underlying technology applied in ADO.NET 3.5 is very different from the COM-based ADO. The ADO.NET Common Language Runtime provides bidirectional, transparent integration with COM. This means that COM and ADO.NET applications and components can use functionality from each system. But the ADO.NET 3.5 Framework provides developers with a significant number of benefits including a more robust, evidence-based security model, automatic memory management native Web services support, and Language Integrated Query (LINQ). For new development, the ADO.NET 3.5 is highly recommended as a preferred technology because of its powerful managed runtime environment and services.

This chapter will provide a detailed introduction to ADO.NET 2.0 and ADO.NET 3.5 and their components, and these components will be utilized for the rest of the book.

In this chapter, you will:

- Learn the basic classes in ADO.NET 2.0 and its architecture.
- Learn the different ADO.NET 2.0 data providers.
- Learn about the Connection and Command components.
- Learn about the Parameters collection component.
- Learn about the DataAdapter and DataReader components.
- Learn about the DataSet and DataTable components.
- Learn about the ADO.NET 3.5 Entity Framework (EF).
- Learn about the ADO.NET 3.5 Entity Framework Tools (EFT).
- Learn about the ADO.NET Entity Data Model (EDM).

First let's take a closer look at ADO.NET 2.0 and have a global picture of its components.

3.2 OVERVIEW OF ADO.NET 2.0

ADO.NET is a set of classes that expose data access services to the Microsoft .NET programmer. ADO.NET provides a rich set of components for creating distributed, data-sharing applications. It is an integral part of the Microsoft .NET Framework, providing access to relational, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, or Internet browsers.

All ADO.NET classes are located at the System.Data namespace with two files named System.Data.dll and System.Xml.dll. When compiling code that uses the System.Data namespace, reference both System.Data.dll and System.Xml.dll.

Basically speaking, ADO.NET provides a set of classes to support you to develop database applications and enable you to connect to a data source to retrieve, manipulate, and update data with your database. The classes provided by ADO.NET are core to develop a professional data-driven application, and they can be divided into the following three major components:

- Data Provider
- DataSet
- DataTable

These three components are located at the different namespaces. The DataSet and the DataTable classes are located at the System.Data namespace. The classes of the Data Provider are located at the different namespaces based on the types of Data Providers.

Data Provider contains four classes: Connection, Command, DataAdapter, and DataReader. These four classes can be used to perform the different functionalities to help you to:

1. Set a connection between your project and the data source using the Connection object.
2. Execute data queries to retrieve, manipulate, and update data using the Command object.
3. Move the data between your DataSet and your database using the DataAdapter object.
4. Perform data queries from the database (read-only) using the DataReader object.

The DataSet class can be considered as a table container and it can contain multiple data tables. These data tables are only a mapping to those real data tables in your database. But these data tables can also be used separately without connecting to the DataSet. In this case, each data table can be considered as a DataTable object.

The DataSet and DataTable classes have no direct relationship with the Data Provider class; therefore they are often called Data Provider-independent components. Four classes such as Connection, Command, DataAdapter, and DataReader that belong to Data Provider are often called Data Provider-dependent components.

To get a clearer picture of the ADO.NET, let's first take a look at the architecture of the ADO.NET.

3.3 ARCHITECTURE OF ADO.NET 2.0

The ADO.NET 2.0 architecture can be divided into two logical pieces: command execution and caching. Command execution requires features like connectivity, execution, and reading of results. These features are enabled with ADO.NET Data Providers. Caching of results is handled by the DataSet.

The Data Provider enables connectivity and command execution to underlying data sources. Note that these data sources do not have to be relational databases. Once a command has been executed the results can be read using a DataReader. A DataReader provides efficient forward-only stream-level access to the results. In addition, results can be used to render a DataSet a DataAdapter. This is typically called “filling the DataSet.”

Figure 3.1 shows a typical architecture of the ADO.NET 2.0.

In this architecture, the data tables are embedded in the DataSet as a DataTableCollection and the data transactions between the DataSet and the Data

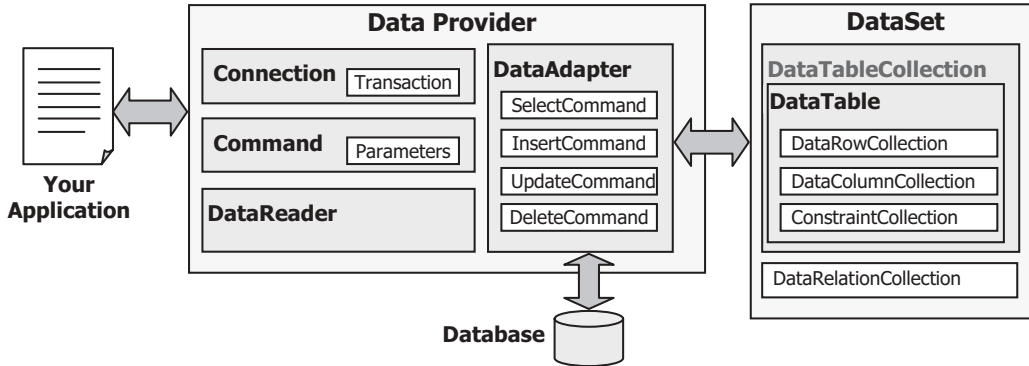


Figure 3.1 Typical architecture of ADO.NET 2.0.

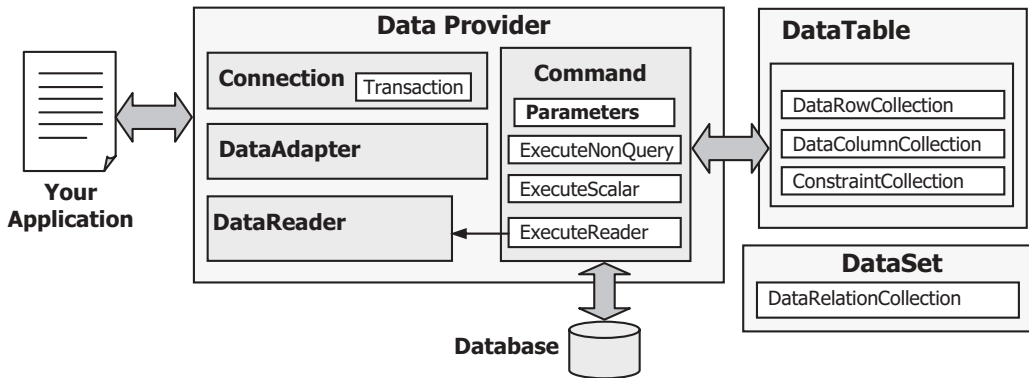


Figure 3.2 Another architecture of ADO.NET 2.0.

Provider such as SELECT, INSERT, UPDATE, and DELETE are made by using the DataAdapter via its own four different methods: SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand, respectively. The Connection object is only used to set a connection between your data source and your applications. The DataReader object is not used for this architecture. As you will see from the sample project in the following chapters, to execute the different methods under the DataAdapter to perform the data query is exactly to call the Command object with different parameters.

Another ADO.NET architecture is shown in Figure 3.2. In this architecture, the data tables are not embedded into the DataSet but treated as independent data tables, and each table can be considered as an individual DataTable object. The data transactions between the Data Provider and the DataTable are realized by executing the different methods of the Command object with the associated parameters. The ExecuteReader() method of the Command object is called when a data query is made from the data source, which is equivalent to executing an SQL SELECT statement, and the returned data should be stored to the DataReader object. When performing other data-accessing operations such as INSERT, UPDATE, or DELETE, the ExecuteNonQuery() method of the

Command object should be called with the suitable parameters attached to the Command object.

Keep these two ADO.NET 2.0 architectures in mind; we will have a more detailed discussion for each component of the ADO.NET below. The sample projects developed in the following sections utilized these two architectures to perform the data query from and the data accessing to the data source.

3.4 COMPONENTS OF ADO.NET 2.0

As we discussed in Section 3.2, ADO.NET 2.0 is composed of three major components: Data Provider, DataSet, and DataTable. First let's take a look at the Data Provider.

3.4.1 Data Provider

The Data Provider can also be called a data driver, and it can be used as a major component for your data-driven applications. The functionalities of the Data Provider, as its name means, are to:

- Connect your data source with your applications.
- Execute different methods to perform the associated data query and data-accessing operations between your data source and your applications.
- Disconnect the data source when the data operations are done.

The Data Provider is physically composed of a binary library file, and this library is in the DLL file format. Sometimes this DLL file depends on other DLL files, so in fact a Data Provider can be made up of several DLL files. Based on the different kinds of databases, Data Provider can have several versions and each version is matched to each kind of database. The popular versions of the Data Provider are:

- **Open DataBase Connectivity (Odbc)** Data Provider (ODBC.NET)
- **Object Linking and Embedding DataBase (OleDb)** Data Provider (OLEDB.NET)
- **SQL Server (Sql)** Data Provider (SQL Server.NET)
- **Oracle (Oracle)** Data Provider (Oracle.NET)

Each Data Provider can be simplified by using an associated keyword, shown in bold in the parentheses above. For instance, the keyword for the ODBC Data Provider is **Odbc**; the keyword for an SQL Server Data Provider is **Sql**, and so on.

In order to distinguish from the older Data Providers such as Microsoft ODBC, Microsoft OLE DB, Microsoft SQL Server, and Oracle, in some books all different Data Providers included in the ADO.NET are extended by a suffix .NET, such as OLE DB.NET, ODBC.NET, SQL Server.NET, and Oracle.NET. Since most Data Providers discussed in this book belong to ADO.NET, generally we do not need to add the .NET suffix but we will add this suffix if the old Data Providers are used.

The different Data Providers are located at the different namespaces, and these namespaces hold the various data classes that you must import into your code in order to use those classes in your project.

Table 3.1 Namespaces for Different Data Providers, DataSet, and DataTable

Namespaces	Descriptions
System.Data	Holds the DataSet and DataTable classes.
System.Data.OleDb	Holds the class collection used to access an OLEDB data source.
System.Data.SqlClient	Holds the classes used to access an SQL Server 7.0 data source or later.
System.Data.Odbc	Holds the class collection used to access an ODBC data source.
System.Data.OracleClient	Holds the classes used to access an Oracle data source.

Table 3.1 lists the most popular namespaces used by the different Data Providers and used by the DataSet and the DataTable.

Since a different Data Provider is located at the different namespace as shown in Table 3.1, you must first add the appropriate namespace into your Visual C#.NET 2008 project, exactly into each form's code window, whenever you want to use that Data Provider. Also all classes provided by that Data Provider must be prefixed by the associated keyword. For example, you must use "using System.Data.OleDb" to add the namespace of the OLEDB.NET Data Provider if you want to use this Data Provider in your project, and also all classes belonging to that Data Provider must be prefixed by the associated keyword OleDb, such as OleDbConnection, OleDbCommand, OleDbDataAdapter, and OleDbDataReader. The same thing holds true for all other Data Providers.

Although different Data Providers are located at the different namespaces and have the different prefixes, the classes of these Data Providers have the similar methods or properties with the same name. For example, no matter what kind of Data Provider you are using such as an OleDb, an Sql, or an Oracle, they have the methods or properties with the same name, such as Connection String property, Open() and Close() methods, as well as the ExecuteReader() method. This provides flexibility for the programmers and allows them to use different Data Providers to access the different data sources by only modifying the prefix applied before each class.

The following sections provide a more detailed discussion for each specific Data Provider. These discussions will give you a direction or guideline to help you to select the appropriate Data Provider when you want to use them to develop the different data-driven applications.

3.4.1.1 ODBC Data Provider

The .NET Framework Data Provider for ODBC uses native ODBC Driver Manager (DM) through COM interop to enable data access. The ODBC Data Provider supports both local and distributed transactions. For distributed transactions, the ODBC Data Provider, by default, automatically enlists in a transaction and obtains transaction details from Windows 2000 Component Services.

The ODBC.NET Data Provider provides access to ODBC data sources with the help of native ODBC drivers in the same way that the OleDb.NET Data Provider accesses native OLE DB providers.

The ODBC.NET supports the following Data Providers:

- SQL Server
- Microsoft ODBC for Oracle
- Microsoft Access Driver (*.mdb)

Some older database systems only support ODBC as the data access technique, which include older versions of SQL Server and Oracle as well as some third-party database such as Sybase.

3.4.1.2 OLEDB Data Provider

The System.Data.OleDb namespace holds all classes used by the .NET Framework 2.0 Data Provider for OLE DB. The .NET Framework Data Provider for OLE DB describes a collection of classes used to access an OLE DB data source in the managed space. Using the OleDbDataAdapter, you can fill a memory-resident DataSet that you can use to query and update the data source. The OLE DB.NET data access technique supports the following Data Providers:

- Microsoft Access
- SQL Server (7.0 or later)
- Oracle (9i or later)

One advantage of using the OLEDB.NET Data Provider is to allow users to develop a generic data-driven application. The so-called generic application means that you can use the OLEDB.NET Data Provider to access any data source such as Microsoft Access, SQL Server, Oracle, and other data sources that support the OLEDB.

Table 3.2 shows the compatibility between the OLEDB Data Provider and the OLE DB.NET Data Provider.

3.4.1.3 SQL Server Data Provider

This Data Provider provides access to a SQL Server version 7.0 or later database using its own internal protocol. The functionality of the Data Provider is designed to be similar to that of the .NET Framework Data Providers for OLE DB, ODBC, and Oracle. All classes related to this Data Provider are defined in a DLL file and located at the System.Data.SqlClient namespace. Although Microsoft provides different Data Providers to access the data in the SQL Server database, such as the ODBC and OLE DB, for the sake of optimal data operations, it is highly recommended to use this Data Provider to access the data in an SQL Server data source.

Table 3.2 Compatibility between OLEDB and OLEDB.NET

Provider Name	Descriptions
SQLOLEDB	Used for Microsoft SQL Server 6.5 or earlier
Microsoft.Jet.OLEDB.4.0	Used for Microsoft JET database (Microsoft Access)
MSDAORA	Used for Oracle version 7 and later

As shown in Table 3.2, this Data Provider is a new version and it can only work for the SQL Server version 7.0 and later. If an old version of SQL Server is used, you need to use either an OLE DB.NET or a SQLOLEDB Data Provider.

3.4.1.4 Oracle Data Provider

This Data Provider is an add-on component to the .NET Framework that provides access to the Oracle database. All classes related to this Data Provider are located in the System.Data.OracleClient namespace. This provider relies upon Oracle Client Interfaces provided by the Oracle Client software. You need to install the Oracle Client software on your computer to use this Data Provider.

Microsoft provides multiple ways to access the data stored in an Oracle database, such as Microsoft ODBC for Oracle and OLE DB. You should use this Oracle Data Provider to access the data in an Oracle data source since this one provides the most efficient way to access the Oracle database.

This Data Provider can only work for the recent versions of the Oracle database such as 8.1.7 and later. For old versions of the Oracle database, you need to use either MSDAORA or an OLE DB.NET.

As we mentioned in the previous parts, all different Data Providers use the similar objects, properties, and methods to perform the data operations for the different databases. In the following sections, we will make a detailed discussion for these similar objects, properties, and methods used for the different Data Providers.

3.4.2 Connection Class

As shown in Figures 3.1 and 3.2, the Data Provider contains four subclasses and the Connection component is one of them. This class provides a connection between your applications and the database you selected to connect to your project. To use this class to set up a connection between your application and the desired database, you need first to create an instance or an object based on this class. Depending on your applications, you can create a global connection instance for your entire project or you can create some local connection objects for each of your form windows. Generally a global instance is a good choice since you do not need to perform multiple open and close operations for connection objects. A global connection instance is used in all sample projects in this book.

The Connection object you want to use depends on the type of the data source you selected. Data Provider provides four different Connection classes and each one is matched to one different database. Table 3.3 lists these popular Connection classes used for the different data sources.

The new keyword is used to create a new instance or object of the Connection class. Although different Connection classes provide different overloaded constructors, two popular constructors are utilized widely for Visual C#.NET 2008. One of them does not accept any argument, but another one accepts a connection string as the argument, and this constructor is the most commonly used for data connections.

The connection string is a property of the Connection class, and it provides all necessary information to connect to your data source. Regularly this connection string contains

Table 3.3 Connection Classes and Databases

Connection Class	Associated Database
OdbcConnection	ODBC Data Source
OleDbConnection	OLE DB Database
SqlConnection	SQL Server Database
OracleConnection	Oracle Database

quite a few parameters to define a connection, but only five of them are popularly utilized for most data-driven applications:

1. Provider
2. Data source
3. Database
4. User ID
5. Password

For different databases, the parameters contained in the connection string may have little difference. For example, both OLE DB and ODBC databases need all of these five parameters to be included in a connection string to connect to an OleDb or Odbc data source. But for the SQL Server database connection, you may need to use the Server to replace the Provider parameter, and for the Oracle database connection, you do not need the Provider and Database parameters at all for your connection string. You can find these differences in Section 5.20.2.

The parameter names in a connection string are case insensitive, but some of parameters such as the Password or PWD may be case sensitive. Many of the connection string properties can be read out separately. For example, one of properties, state, is one of the most useful property for your data-driven applications. By checking this property, you can get to know what is the current connection status between your database and your project, and this checking is necessary for you to make the decision about which way your program is supposed to go. Also you can avoid the unnecessary errors related to the data source connection by checking this property. For example, you cannot perform any data operation if your database has not been connected to your application. By checking this property, you can get a clear picture whether your application is connected to your database or not.

A typical data connection instance with a general connection string can be expressed by the following codes:

```
xxxConnection Connection
= new xxxConnection("Provider=MyProvider;" +
                    "Data Source=MyServer;" +
                    "Database=MyDatabase;" +
                    "User ID=MyUserID;" +
                    "Password=MyPassWord;")
```

where **xxx** should be replaced by the selected Data Provider in your real application, such as OleDb, Sql, or Oracle. You need to use the real parameter values implemented in

your applications to replace those nominal values such as MyServer, MyDatabase, MyUserID, and MyPassWord in your application.

The Provider parameter indicates the database driver you selected. If you installed a local SQL server and client such as the SQL Server 2005 Express on your computer, the Provider should be localhost. If you are using a remote SQL Server instance, you need to use that remote server's network name. If you are using the default named instance of SQLX on your computer, you need to use `.\SQLEXPRESS` as the value for your Provider parameter. For the Oracle server database, you do not need to use this parameter.

The Data Source parameter indicates the name of the network computer on which your SQL server or Oracle server is installed and running. The Database parameter indicates your database name. The User ID and Password parameters are used for the security issue for your database. In most cases, the default Windows NT Security Authentication is utilized. Some typical Connection instances used for the different databases are listed below:

OLE DB Data Provider for Microsoft Access Database:

```
OleDbConnection Connection
= new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
    "Data Source=C:\database\CSE_DEPT.mdb;" +
    "User ID=MyUserID;" +
    "Password=MyPassWord;")
```

SQL Server Data Provider for SQL Server Database:

```
SqlConnection Connection
= new SqlConnection("Server=localhost;" +
    "Data Source=YBAI\SQLEXPRESS;" +
    "Database=CSE_DEPT;" +
    "Integrated Security=SSPI")
```

Oracle Data Provider for Oracle Database:

```
OracleConnection Connection
= new OracleConnection("Data Source=XE;" +
    "User ID=system;" +
    "Password=reback")
```

Besides these important properties such as the connection string and state, the Connection class contains some important methods, such as the Open() and Close() methods. To make a real connection between your data source and your application, the Open() method is needed, and the Close() method is also needed when you finished the data operations and you want to exit your application.

3.4.2.1 Open() Method of Connection Class

To create a real connection between your database and your applications, the Open() method of the Connection class is called, and it is used to open a connection to a data source with the property settings specified by the connection string. An important issue for this connection is that you must make sure that this connection is a bug-free connection, in other words, the connection is successful and you can use this connection to access data from your application to your desired data source without any problem. One of the

```

string strConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;" +
                             "Data Source=C:\\database\\Access\\CSE_DEPT.accdb;";
accConnection = new OleDbConnection(strConnectionString);
try
{
    accConnection.Open();
}
catch (OleDbException e)
{
    MessageBox.Show("Access Error");
    MessageBox.Show("Error Code = " + e.ErrorCode);
    MessageBox.Show("Error Message = " + e.Message);
}
catch (InvalidOperationException e)
{
    MessageBox.Show("Invalid Message = " + e.Message);
}
if (accConnection.State != ConnectionState.Open)
{
    MessageBox.Show("Database connection is Failed");
    Application.Exit();
}

```

Figure 3.3 Example coding of the opening a connection.

efficient ways to do this is to use the try ... catch block to embed this Open() operation to try to find and catch the typical possible errors caused by this connection. An example coding of opening an OLEDB connection is shown in Figure 3.3.

The Microsoft.ACE.OLEDB.12.0 driver, which is a driver for the Microsoft Access 2007, is used as the data provider and the Microsoft Access 2007 database file CSE_DEPT.accdb is located at the database\Access folder at our local computer. The Open() method, which is embedded inside the try ... catch block, is executed after a new OleDbConnection object is created to open this connection. Two possible typical errors, either an OleDbException or an InvalidOperationException, could be happened after this Open() method is executed. A related message would be displayed if any one of those errors occurred and was caught.

To make sure that the connection is bug-free, one of the properties of the Connection class, State, is used. This property has two possible values: Open or Closed. By checking this property, we can inspect whether this connection is successful or not. The project will be exited if this connection fails since we can do nothing without this connection object being set up and our database connection being successful.

3.4.2.2 Close() Method of Connection Class

The Close() method is a partner of the Open() method, and it is used to close a connection between your database and your applications when you have finished your data operations to the data source. You should close any connection object you connected to your data source after you finished the data access to any data source, otherwise a possible error may be encountered when you try to reopen that connection the next time you run your project.

```
' clean up the objects used
accConnection.Close()
accConnection.Dispose()
```

Figure 3.4 Example coding for the cleanup of resources.

Unlike the `Open()` method, which is a key to your data access and operation to your data source, the `Close()` method does not throw any exceptions when you try to close a connection that has already been closed. So you do not need to use a try ... catch block to catch any error for this method.

3.4.2.3 *Dispose()* Method of Connection Class

The `Dispose()` method of the `Connection` class is an overloaded method and it is used to release the resources used by the `Connection` object. You need to call this method after the `Close()` method is executed to perform a cleanup job to release all resources used by the `Connection` object during your data access and operations to your data source. Although it is unnecessary for you to have to call this `Dispose()` method to do the cleanup job since one of system tools, Garbage Collection, can periodically check and clean all resources used by unused objects in your computer, it is highly recommended for you to make this kind of coding to make your program more professional and efficient. A piece of example code is shown in Figure 3.4.

Now that we finished the discussion for the first component defined in a Data Provider, the `Connection` object, let's take a look at the next object, the `Command` object. Since a close relationship exists between the `Command` and the `Parameter` object, we discuss these two objects in one section.

3.4.3 Command and Parameter Classes

`Command` objects are used to execute commands against your database such as a data query, an action query, and even a stored procedure. In fact, all data accesses and data operations between your data source and your applications are achieved by executing the `Command` object with a set of parameters.

`Command` class can be divided into different categories, and these categories are based on the different Data Providers. For the popular Data Providers, such as OLE DB, ODBC, SQL Server, and Oracle, each one has its own `Command` class. Each `Command` class is identified by a different prefix such as `OleDbCommand`, `OdbcCommand`, `SqlCommand`, and `OracleCommand`. Although these different `Command` objects belong to the different Data Providers, they have similar properties and methods, and they are equivalent in functionalities.

Depending on the architecture of the ADO.NET, the `Command` object can have two different roles when you are using it to perform a data query or a data action. Refer to Figures 3.1 and 3.2. In Figure 3.1, if a `TableAdapter` is utilized to perform a data query and all data tables are embedded into the `DataSet` as a data-catching unit, the `Command` object is embedded into the different data query methods of the `TableAdapter`, such as `SelectCommand`, `InsertCommand`, `UpdateCommand`, and `DeleteCommand`, and is exe-

cuted based on the associated query type. In this case, the Command object can be executed indirectly, which means that you do not need to use any executing method to run the Command object directly, instead you can run it by executing the associated method of the TableAdapter.

In Figure 3.2, each data table can be considered as an individual table. The Command object can be executed directly based on the attached parameter collection that is created and initialized by the user.

No matter which role you want to use for the Command object in your application, you should first create, initialize, and attach the Parameters collection to the Command object before you can use it. Also you must initialize the Command object by assigning the suitable properties to it in order to use the Command object to access the data source to perform any data query or data action. Some popular properties of the Command class are discussed below.

3.4.3.1 Properties of Command Class

The Command class contains more than 10 properties, but only 4 of them are used popularly in most applications:

- Connection property
- CommandType property
- CommandText property
- Parameters property

The Connection property is used to hold a valid Connection object, and the Command object can be executed to access the connected database based on this Connection object. The CommandType property is used to indicate what kind of command stored in the CommandText property should be executed. In other words, the CommandType property specifies how the CommandText property can be interpreted. Three CommandType properties are available: Text, TableDirect, and StoredProcedure. The default value of this property is Text.

The content of the CommandText property is determined by the value of the CommandType property. It contains a complete SQL statement if the value of the CommandType property is Text. It may contain a group of SQL statements if the value of the CommandType property is StoredProcedure.

The Parameters property is used to hold a collection of the Parameter objects. You need to note that Parameters is a collection but Parameter is an object, which means that the former contains a group of objects and you can add the latter to the former.

You must first create and initialize a Parameter object before you can add that object to the Parameters collection for a Command object.

3.4.3.2 Constructors and Properties of Parameter Class

The Parameter class has four popular constructors, which are shown in Figure 3.5 (an SQL Server Data Provider is used as an example).

The first constructor is a blank one, and you need to initialize each property of the Parameter object one by one if you want to use this constructor to represent a new

```

SqlParameter sqlParameter = new SqlParameter();
SqlParameter sqlParameter = new SqlParameter(ParamName, objValue);
SqlParameter sqlParameter = new SqlParameter(ParamName, sqlDbType);
SqlParameter sqlParameter = new SqlParameter(ParamName, sqlDbType, intSize);

```

Figure 3.5 Four constructors of the Parameter class.

Table 3.4 Data Types and Associated Data Providers

Data Type	Associated Data Provider
OdbcType	ODBC Data Provider
OleDbType	OLE DB Provider
SqlDbType	SQL Server Data Provider
OracleType	Oracle Data Provider

Parameter object. Three popular properties of a Parameter object are:

- ParameterName
- Value
- DbType

The first property ParameterName contains the name of the selected parameter. The second property Value is the value of the selected parameter and it is an object. The third property DbType is used to define the data type of the selected parameter.

All parameters in the Parameter object must have a data type, and you can indicate a data type for a selected parameter by specifying the DbType property. ADO.NET and ADO.NET Data Provider have different definitions for the data types they provided. DbType is the data type used by ADO.NET, but ADO.NET Data Provider has another four different popular data types and each one is associated with a Data Provider. Table 3.4 lists these data types as well as the associated Data Providers.

Even the data types provided by ADO.NET and ADO.NET Data Provider are different, but they have a direct connection between them. As a user, you can use any data type you like, and the other one will be automatically changed to the corresponding value if you set one of them. For example, if you set the DbType property of an SqlParameter object to String, the SqlDbType parameter will be automatically set to Char. In this book, we always use the data types defined in the ADO.NET Data Provider since all parameters discussed in this section are related to the different Data Providers. The default data type for the DbType property is String.

3.4.3.3 Parameter Mapping

When you add a Parameter object to the Parameters collection of a Command object by attaching that Parameter object to the Parameters property of the Command class, the Command object needs to know the relationship between that added parameter and the parameters you used in your SQL query string such as a SELECT statement. In other words, the Command object needs to identify which parameter used in your SQL state-

Table 3.5 Different Parameter Mappings

Parameter Mapping	Associated Data Providers
Positional Parameter Mapping	ODBC Data Provider
Positional Parameter Mapping	OLE DB Provider
Named Parameter Mapping	SQL Server Data Provider
Named Parameter Mapping	Oracle Data Provider

ment should be mapped to this added parameter. Different parameter mappings are used for different Data Providers. Table 3.5 lists these mappings.

Both OLE DB and ODBC Data Providers used so-called positional parameter mapping, which means that the relationship between the parameters defined in an SQL statement and the added parameters into a Parameters collection is one-to-one in the order. In other words, the order in which the parameters appear in an SQL statement and the order in which the parameters are added into the Parameters collection should be exactly identical. Positional Parameter Mapping is indicated with a question mark.

For example, the following SQL statement is used for an OLE DB Data Provider as a query string:

```
SELECT user_name, pass_word FROM LogIn
WHERE (user_name=?) AND (pass_word=?)
```

The user_name and pass_word are mapped to two columns in the LogIn data table. Two dynamic parameters are represented by two question marks in this SQL statement. To add a Parameter object to the Parameters collection of a Command object accCommand, you need to use the Add() method as shown below:

```
accCommand.Parameters.Add("user_name",
OleDbType.Char).Value = txtUserName.Text;
accCommand.Parameters.Add("pass_word",
OleDbType.Char, 8).Value = txtPassWord.Text;
```

You must be careful with the order in which you add these two parameters, user_name and pass_word, and make sure that this order is identical with the order in which those two dynamic parameters (?) appear in the above SQL statement.

Both SQL Server and Oracle Data Provider used the Named Parameter Mapping, which means that each parameter, either defined in an SQL statement or added into a Parameters collection, is identified by the name. In other words, the name of the parameter appearing in an SQL statement or a stored procedure must be identical with the name of the parameter you added into a Parameters collection.

For example, the following SQL statement is used for an SQL Server Data Provider as a query string:

```
SELECT user_name,
pass_word FROM LogIn WHERE (user_name LIKE @Param1)
AND (pass_word LIKE @Param2)
```

The user_name and pass_word are mapped to two columns in the LogIn data table. Compared with the above SQL statement, two dynamic parameters are represented by two nominal parameters @Param1 and @Param2 in this SQL statement. The equal opera-

```

SqlParameter paramUserName = new SqlParameter();
SqlParameter paramPassWord = new SqlParameter();

paramUserName.ParameterName = "@Param1";
paramUserName.Value = txtUserName.Text;
paramPassWord.ParameterName = "@Param2";
paramPassWord.Value = txtPassWord.Text;

```

Figure 3.6 Example of initializing the property of a Parameter object.

tor is replaced by the keyword LIKE for two parameters. This change is required by the SQL Server Data Provider.

Then you need two Parameter objects associated with your Command object. An example of initializing these two Parameter objects is shown in Figure 3.6.

In this example, two ParameterName properties are assigned with two dynamic parameters, “@Param1” and “@Param2”, respectively. Both Param1 and Param2 are nominal names of the dynamic parameters and an @ symbol is prefixed before each parameter since this is the requirement of the SQL Server database when a dynamic parameter is utilized in an SQL statement.

You can see from this piece of code, the name of each parameter you used for each Parameter object must be identical with the name you defined in your SQL statement. Since the SQL Server and Oracle Data Provider use Named Parameter Mapping, you do not need to worry about the order in which you added Parameter objects into the Parameters collection of the Command object.

To add Parameter objects into a Parameters collection of a Command object, you need to use some methods defined in the ParameterCollection class.

3.4.3.4 Methods of ParameterCollection Class

Each ParameterCollection class has more than 10 methods, but only two of them are most often utilized in the data-driven applications, which are Add() and AddWithValue() methods. Each Parameter object must be added into the Parameters collection of a Command object before you can execute that Command object to perform any data query or data action.

As we mentioned in the last section, you do not need to worry about the order in which you add the parameter into the Parameter object if you are using a Named Parameter Mapping Data Provider such as an SQL Server or an Oracle. But you must pay attention to the order in which you add the parameter into the Parameter object if you are using a Positional Parameter Mapping Data Providers such as an OLE DB or an ODBC.

To add Parameter objects to an Parameters collection of a Command object, two popular ways are generally adopted, Add() method and AddWithValue() method.

The Add() method is an overloaded method, and it has five different protocols, but only two of them are widely used. The protocols of these two methods are shown below.

```

ParameterCollection.Add(SqlParameter value);
ParameterCollection.Add(string parameterName, Object Value);

```

The first method needs a Parameter object as the argument, and that Parameter object should have been created and initialized before you call this Add() method to add it into the collection if you want to use this method.

```

SqlParameter paramUserName = new SqlParameter();
SqlParameter paramPassWord = new SqlParameter();

paramUserName.ParameterName = "@Param1";
paramUserName.Value = txtUserName.Text;
paramPassWord.ParameterName = "@Param2";
paramPassWord.Value = txtPassWord.Text;

sqlCommand.Parameters.Add(paramUserName);
sqlCommand.Parameters.Add(paramPassWord);
.....
sqlCommand.Parameters.AddWithValue("@Param1", txtUserName.Text);
sqlCommand.Parameters.AddWithValue("@Param2", txtPassWord.Text);

```

Figure 3.7 Two methods to add Parameter objects.

The second method contains two arguments. The first one is a String that contains the ParameterName and the second is an object that includes the value of that parameter.

The AddWithValue() method is similar to the second Add() method with the following protocol:

```
ParameterCollection.AddWithValue(string parameterName, Object Value);
```

An example of using these two methods to add Parameter objects into a Parameters collection is shown in Figure 3.7. The top section is used to create and initialize the Parameter objects, which we have discussed in previous sections.

First the Add() method is executed to add two Parameter objects, paramUserName and paramPassWord to the Parameters collection of the Command object sqlCommand. To use this method, two Parameter objects should have been initialized.



The Parameters property in the Command class is a collection of a set of Parameter objects. You need first to create and initialize a Parameter object, and then you can add that Parameter object to the Parameters collection. In this way, you can assign that Parameter object to a Command object.

The second way to do this job is to use the AddWithValue() method to add these two Parameter objects, which is similar to the second protocol of the Add() method.

3.4.3.5 Constructor of Command Class

The constructor of the Command class is an overloaded method and it has multiple protocols. Four popular protocols are listed in Figure 3.8 (an SQL Server Data Provider is used as an example).

The first constructor is a blank one without any argument. You have to create and assign each property to the associated property of the Command object separately if you want to use this constructor to represent a new Command object.

```

SqlCommand sqlCommand = new SqlCommand();
SqlCommand sqlCommand = new SqlCommand (connString);
SqlCommand sqlCommand = new SqlCommand (connString, SqlConnection);
SqlCommand sqlCommand = new SqlCommand (connString, SqlConnection, SqlTransaction);

```

Figure 3.8 Three popular protocols of the constructor of the Command class.

```

string cmdString = "SELECT user_name, pass_word FROM LogIn ";
cmdString += "WHERE (user_name LIKE @Param1 ) AND (pass_word LIKE @Param2)";
SqlParameter paramUserName = new SqlParameter();
SqlParameter paramPassWord = new SqlParameter();
SqlCommand sqlCommand As New SqlCommand();

paramUserName.ParameterName = "@Param1";
paramUserName.Value = txtUserName.Text;
paramPassWord.ParameterName = "@Param2";
paramPassWord.Value = txtPassWord.Text;
sqlCommand.Connection = sqlConnection;
sqlCommand.CommandType = CommandType.Text;
sqlCommand.CommandText = cmdString;
sqlCommand.Parameters.Add(paramUserName);
sqlCommand.Parameters.Add(paramPassWord);

```

Figure 3.9 Example of creating a SqlCommand object.

The second constructor contains two arguments; the first one is the parameter name, which is a string variable, and the second is the value, which is an object. The following two constructors are similar to the second one, and the difference is that a data type and a data size argument are included.

An example of creating an SqlCommand object is shown in Figure 3.9. This example contains the following functionalities:

1. Create a SqlCommand object.
2. Create two SqlParameter objects.
3. Initialize two SqlParameter objects.
4. Initialize the SqlCommand object.
5. Add two Parameter objects into the Parameters collection of the Command object sqlCommand.

The top two lines of the coding create an SQL statement with two dynamic parameters, user_name and pass_word. Then two strings are concatenated to form a complete string. Two SqlParameter and an SqlCommand objects are created in Figure 3.9.

Then two SqlParameter objects are initialized with nominal parameters and the associated textbox's contents. After this, the SqlCommand object is initialized with four properties of the Command class.

Now let's take care of the popular methods used in the Command class.

3.4.3.6 Methods of Command Class

In the last section, we discussed how to create an instance of the Command class and how to initialize the Parameters collection of a Command object by attaching Parameter

Table 3.6 Methods of the Command Class

Method Name	Functionality
ExecuteReader()	Executes commands that return rows, such as a SQL SELECT statement. The returned rows are located in an OdbcDataReader, an OleDbDataReader, a SqlDataReader, or an OracleDataReader, depending on which Data Provider you are using.
ExecuteScalar()	Retrieves a single value from the database.
ExecuteNonQuery()	Executes a nonquery command such as SQL INSERT, DELETE, UPDATE, and SET statements.
ExecuteXmlReader (SqlCommand only)	Similar to the ExecuteReader method, but the returned rows must be expressed using XML. This method is only available for the SQL Server Data Provider.

```
string cmdString = "SELECT user_name, pass_word FROM LogIn ";
SqlCommand sqlCommand = new SqlCommand();

sqlCommand.Connection = sqlConnection;
sqlCommand.CommandType = CommandType.Text;
sqlCommand.CommandText = cmdString;
sqlDataReader = sqlCommand.ExecuteReader();
```

Figure 3.10 Example coding of running of ExecuteReader method.

objects to that Command object. Those steps are prerequisite to execute a Command object. The actual execution of a Command object is to run one of methods of the Command class to perform the associated data queries or data actions. Four popular methods are widely utilized for most data-driven applications and Table 3.6 lists these methods.

As we mentioned in the last section, the Command object is a Data Provider-dependent object, so four different versions of the Command object are developed, and each version is determined by the Data Provider the user selected and used in the application, such as the OleDbCommand, OdbcCommand, SqlCommand, and an OracleCommand. Although each Command object is dependent on the Data Provider, all methods of the Command object are similar in functionality and have the same roles in a data-driven application.

3.4.3.6.1 ExecuteReader Method The ExecuteReader() method is a data query method and it can only be used to execute a readout operation from a database. The most popular matched operation is to execute an SQL SELECT statement to return rows to a DataReader by using this method. Depending on which Data Provider you are using, the different DataReader objects should be utilized as the data receiver to hold the returned rows. Remember, the DataReader class is a read-only class and it can only be used as a data holder. You cannot perform any data updating by using the DataReader.

The example coding shown in Figure 3.10 can be used to execute an SQL SELECT statement.

As shown in Figure 3.10, as the `ExecuteReader()` method is called, an SQL `SELECT` statement is executed to retrieve the `id`, `user_name` and `pass_word` from the `LogIn` table. The returned rows are assigned to the `sqlDataReader` object. Please note that the `SqlCommand` object should already be created and initialized before the `ExecuteReader()` method can be called.

3.4.3.6.2 ExecuteScalar Method The `ExecuteScalar()` method is used to retrieve a single value from a database. This method is faster and has substantially less overhead than the `ExecuteReader()` method. You should use this method whenever a single value needs to be retrieved from a data source. A sample coding of using this method is shown in Figure 3.11.

In this sample, the SQL `SELECT` statement is used to try to pick up a password based on the username `ybai`, from the `LogIn` data table. This password can be considered as a single value. The `ExecuteScalar()` method is called after an `SqlCommand` object is created and initialized. The returned single value is a `String` and it is assigned to a `String` variable `passWord`.

Section 5.9 provides an example of using this method to pick up a single value, which is a password, from the `LogIn` data table in the `CSE_DEPT` database.

3.4.3.6.3 ExecuteNonQuery Method As we mentioned, the `ExecuteReader()` method is a read-only method and it can only be used to perform a data query job. To execute the different SQL statements such as `INSERT`, `UPDATE`, or `DELETE` commands, the `ExecuteNonQuery()` method is needed.

Figure 3.12 shows an example coding of using this method to insert and to delete a record from the `LogIn` data table.

```
string cmdString = "SELECT pass_word FROM LogIn WHERE (user_name = ybai)";
SqlCommand sqlCommand = new SqlCommand();
string passWord = string.Empty;

sqlCommand.Connection = sqlConnection;
sqlCommand.CommandType = CommandType.Text;
sqlCommand.CommandText = cmdString;
passWord = sqlCommand.ExecuteScalar();
```

Figure 3.11 Sample coding of using the `ExecuteScalar` method.

```
string cmdString1 = "INSERT INTO LogIn (pass_word) VALUES ('reback')";
string cmdString2 = "DELETE FROM LogIn WHERE (user_name = ybai)";
SqlCommand sqlCommand = new SqlCommand();

sqlCommand.Connection = sqlConnection;
sqlCommand.CommandType = CommandType.Text;
sqlCommand.CommandText = cmdString1;
sqlCommand.ExecuteNonQuery();
sqlCommand.CommandText = cmdString2;
sqlCommand.ExecuteNonQuery();
```

Figure 3.12 Example coding of using the `ExecuteNonQuery` method.

As shown in Figure 3.12, the first SQL statement is trying to insert a new password into the LogIn data table with a value reback. After an SqlCommand object is created and initialized, the ExecuteNonQuery() method is called to execute this INSERT statement. A similar procedure is performed for the DELETE statement.

Now let's look at the next class in the Data Provider, the DataAdapter.

3.4.4 DataAdapter Class

The DataAdapter serves as a bridge between a DataSet and a data source for retrieving and saving data. The DataAdapter provides this bridge by mapping Fill(), which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet.

The DataAdapter connects to your database using a Connection object, and it uses Command objects to retrieve data from the database and populate those data to the DataSet and related classes such as DataTables; also the DataAdapter uses Command objects to send data from your DataSet to your database.

To perform data query from your database to the DataSet, the DataAdapter uses the suitable Command objects and assigns them to the appropriate DataAdapter properties, such as SelectCommand, and executes that Command. To perform other data manipulations, the DataAdapter uses the same Command objects but assign them with different properties such as InsertCommand, UpdateCommand, and DeleteCommand to complete the associated data operations.

As we mentioned in the previous section, the DataAdapter is a subcomponent of the Data Provider, so it is a Data Provider–dependent component. This means that the DataAdapter has different versions based on the used Data Provider. Four popular DataAdapters are OleDbDataAdapter, OdbcDataAdapter, SqlDataAdapter, and OracleDataAdapter. Different DataAdapters are located at the different namespaces.

If you are connecting to an SQL Server database, you can increase overall performance by using the SqlDataAdapter along with its associated SqlCommand and SqlConnection objects. For OLE DB–supported data sources, use the OleDbDataAdapter with its associated OleDbCommand and OleDbConnection objects. For ODBC-supported data sources, use the OdbcDataAdapter with its associated OdbcCommand and OdbcConnection objects. For Oracle databases, use the OracleDataAdapter with its associated OracleCommand and OracleConnection objects.

3.4.4.1 Constructor of DataAdapter Class

The constructor of the DataAdapter class is an overloaded method and it has multiple protocols. Two popular protocols are listed in Table 3.7 (an SQL Server Data Provider is used as an example).

The first constructor is most often used in most data-driven applications.

3.4.4.2 Properties of DataAdapter Class

Some popular properties of the DataAdapter class are listed in Table 3.8.

Table 3.7 Constructors of the DataAdapter Class

Constructor	Descriptions
SqlDataAdapter()	Initializes a new instance of a DataAdapter class.
SqlDataAdapter(from)	Initializes a new instance of a DataAdapter class from an existing object of the same type.

Table 3.8 Public Properties of the DataAdapter Class

Properties	Descriptions
AcceptChangesDuringFill	Gets or sets a value indicating whether AcceptChanges is called on a DataRow after it is added to the DataTable during any of the Fill operations.
MissingMappingAction	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction	Determines the action to take when existing DataSet schema does not match incoming data.
TableMappings	Gets a collection that provides the master mapping between a source table and a DataTable.

Table 3.9 Public Methods of the DataAdapter Class

Methods	Descriptions
Dispose	Releases the resources used by the DataAdapter.
Fill	Adds or refreshes rows in the DataSet to match those in the data source using the DataSet name and creates a DataTable.
FillSchema	Adds a DataTable to the specified DataSet.
GetFillParameters	Gets the parameters set by the user when executing an SQL SELECT statement.
ToString	Returns a string containing the name of the Component, if any. This method should not be overridden.
Update	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet from a named DataTable.

3.4.4.3 Methods of DataAdapter Class

The DataAdapter has more than 10 methods available to help users to develop professional data-driven applications. Table 3.9 lists some of the most often used methods. Among these methods, the Dispose(), Fill(), FillSchema(), and Update() are most often used methods. The Dispose() method should be used to release the used DataAdapter after the DataAdapter completes its job. The Fill() method should be used to populate a DataSet after the Command object is initialized and ready to be used. The FillSchema() method should be called if you want to add a new DataTable into the DataSet, and the

Table 3.10 Events of the DataAdapter Class

Events	Descriptions
Disposed	Occurs when the component is disposed of by a call to the Dispose method.
FillError	Returned when an error occurs during a fill operation.

```

A string cmdString = "SELECT name, office, title, college FROM Faculty";
    SqlCommand sqlCommand = new SqlCommand();
    SqlDataAdapter sqlDataAdapter;
    DataSet sqlDataSet;

B   sqlCommand.Connection = sqlConnection;
    sqlCommand.CommandType = CommandType.Text;
    sqlCommand.CommandText = cmdString;

C   sqlDataAdapter = new SqlDataAdapter(cmdString, sqlConnection);
D   sqlDataAdapter.SelectCommand = sqlCommand;
    sqlDataSet = new DataSet();
    sqlDataSet.Clear();

E   int intValue = sqlDataAdapter.Fill(sqlDataSet);
    if (intValue == 0)
        MessageBox.Show("No valid faculty found!");

F   sqlDataSet.Dispose();
    sqlDataAdapter.Dispose();
    sqlCommand.Dispose();

```

Figure 3.13 Example of using the SqlDataAdapter to fill the DataSet.

Update() method should be used if you want to perform some data manipulations such as Insert, Update, and Delete with the database and the DataSet.

3.4.4.4 Events of DataAdapter Class

Two popular events are widely used in the DataAdapter class, and these events are listed in Table 3.10.

Before we can complete this section, an example coding is provided to show readers how to use the DataAdapter to perform some data access and data actions between your DataSet and your database. Figure 3.13 shows an example of using an SQL Server DataAdapter (assuming that a Connection object sqlConnection has been created).

Starting from step **A**, an SQL SELECT statement string is created with some other new object declarations, such as a new instance of SqlCommand class, a new object of SqlDataAdapter class, and a new instance of the DataSet class. The DataSet class will be discussed in the following section, and it is used as a table container to hold a collection of data tables. The Fill() method of the DataAdapter class can be used to populate the data tables embedded in the DataSet later.

In step **B**, the SqlCommand object is initialized with the Connection object, CommandType, and the command string.

The instance of the SqlDataAdapter, sqlDataAdapter, is initialized with the command string and the SqlConnection object in step **C**.

In step **D**, the initialized `SqlCommand` object, `sqlCommand`, is assigned to the `SelectCommand` property of the `sqlDataAdapter`. Also the `DataSet` is initialized and cleared to make it ready to be filled by executing the `Fill()` method of the `sqlDataAdapter` to populate the data table in the `DataSet` later.

The `Fill()` method is called to execute a population of data from the Faculty data table into the mapping of that table in the `DataSet` in step **E**.

An integer variable `Index` is used to hold the returned value of calling this `Fill()` method. This value is equal to the number of rows filled into the Faculty table in the `DataSet`. If this value is 0, which means that no matched row has been found from the Faculty table in the database and the 0 row has been filled into the Faculty table in the `DataSet`, an error message is displayed. Otherwise, this fill is successful.

In step **F**, all components used for this piece of code are released by using the `Dispose()` method.

3.4.5 DataReader Class

The `DataReader` class is a read-only class, and it can only be used to retrieve and hold the data rows returned from a database executing an `ExecuteReader()` method. This class provides a way of reading a forward-only stream of rows from a database. Depending on the Data Provider you are using, four popular `DataReaders` are provided by four Data Providers. They are `OdbcDataReader`, `OleDbDataReader`, `SqlDataReader`, and `OracleDataReader`.

To create a `DataReader` instance, you must call the `ExecuteReader()` method of the `Command` object, instead of directly using a constructor since the `DataReader` class does not have any public constructor. The following code that is used to create an instance of the `SqlDataReader` is incorrect:

```
SqlDataReader sqlDataReader = new SqlDataReader();
```

While the `DataReader` object is being used, the associated `Connection` is busy serving the `DataReader`, and no other operations can be performed on the `Connection` other than closing it. This is the case until the `Close()` method of the `DataReader` is called. For instance, you cannot retrieve output parameters until after you call the `Close()` method to close the connected `DataReader`.

The `IsClosed` property of the `DataReader` class can be used to check if the `DataReader` has been closed or not, and this property returns a `Boolean` value. A *true* means that the `DataReader` has been closed. It is a good habit to call the `Close()` method to close the `DataReader` each time when you finished data query using that `DataReader` to avoid the troubles caused by the multiple connections to the database.

Table 3.11 lists most public properties of the `SqlDataReader` class. All other `DataReader` classes have similar properties.

The `DataReader` class has more than 50 public methods. Table 3.12 lists the most useful methods of the `SqlDataReader` class. All other `DataReader` classes have similar methods.

When you run the `ExecuteReader()` method to retrieve data rows from a database and assign them to a `DataReader` object, each time the `DataReader` can only retrieve and hold one row. So if you want to read out all rows from a data table, a loop should be used to sequentially retrieve each row from the database.

Table 3.11 Popular Properties of the SqlDataReader Class

Property Name	Value Type	Functionality
FieldCount	Integer	Gets the number of columns in the current row.
HasRows	Boolean	Gets a value that indicates whether the SqlDataReader contains one or more rows.
IsClosed	Boolean	Retrieves a Boolean value that indicates whether the specified SqlDataReader instance has been closed.
Item(Int32)	Native	Gets the value of the specified column in its native format given the column ordinal.
Item(String)	Native	Gets the value of the specified column in its native format given the column name.
RecordsAffected	Integer	Gets the number of rows changed, inserted, or deleted by execution of the Transact-SQL statement.
VisibleFieldCount	Integer	Gets the number of fields in the SqlDataReader that are not hidden.

Table 3.12 Popular Methods of the SqlDataReader Class

Method Name	Functionality
Close	Closes the opened SqlDataReader object.
Dispose	Releases the resources used by the DbDataReader.
GetByte	Gets the value of the specified column as a byte.
GetName	Gets the name of the specified column.
GetString	Gets the value of the specified column as a string.
GetValue	Gets the value of the specified column in its native format.
IsDBNull	Gets a value that indicates whether the column contains nonexistent or missing values.
NextResult	Advances the data reader to the next result, when reading the results of batch Transact-SQL statements.
Read	Advances the SqlDataReader to the next record.
ToString	Returns a String that represents the current Object .

The DataReader object provides the most efficient ways to read data from the database, and you should use this object whenever you just want to read the data from the database from start to finish to populate a list on a form or to populate an array or collection. It can also be used to populate a DataSet or a DataTable.

Figure 3.14 shows an example coding of using the SqlDataReader object to continuously retrieve all records (rows) from the Faculty data table supposing a Connection object sqlConnection has been created.

The functionality of this piece of coding is explained below. Starting from section **A**, a new SqlCommand and an SqlDataReader object is created with a SQL SELECT statement string object. The Command object is initialized in section **B**. In section **C**, the ExecuteReader() method is called to retrieve the data row from the Faculty data table and assign the resulting row to the SqlDataReader object. By checking the HasRows

```

A string cmdString = "SELECT name, office, title, college FROM Faculty";
    SqlCommand sqlCommand = new SqlCommand();
    SqlDataReader sqlDataReader;

B sqlCommand.Connection = sqlConnection;
    sqlCommand.CommandType = CommandType.Text;
    sqlCommand.CommandText = cmdString;

C sqlDataReader = sqlCommand.ExecuteReader();
    if (sqlDataReader.HasRows == true)
    {
        while (FacultyReader.Read())
        {
            for (int intIndex = 0; intIndex <= FacultyReader.FieldCount - 1; intIndex++)
                FacultyLabel(intIndex).Text = FacultyReader.GetString(intIndex);
        }
    }
    else
D     MessageBox.Show("No matched faculty found!");

E sqlDataReader.Close();
    sqlCommand.Dispose();

```

Figure 3.14 Example coding of using the SqlDataReader object.

Table 3.13 Popular Exceptions of the DataReader Class

Exception Name	Functionality
IndexOutOfRangeException	If an index does not exist within the range, array, or collection, this exception occurs.
InvalidCastException	If you try to convert a database value using one of the Get methods to convert a column value to a specific data type, this exception occurs.
InvalidOperationException	If you perform an invalid operation, either a property or a method, this exception occurs.
NotSupportedException	If you try to use any property or method on a DataReader object that has not been opened or connected, this exception occurs.

property (refer to Table 3.11), one can determine whether a valid row has been collected or not. If a valid row has been retrieved, a *while* and a *for* loop is utilized to sequentially read out all rows one by one using the Read() method (refer to Table 3.12). The GetString() method (refer to Table 3.12) is used to populate the retrieved row to a Label control collection object. The FieldCount property (refer to Table 3.11) is used as the termination condition for the for loop, and its termination value is FieldCount-1 since the loop starts from 0, not 1. If the HasRows property returns a *false*, which means that no row has been retrieved from the Faculty table, an error message will be displayed in section **D**. Finally, before we can finish this data query job, we need to clean up the sources we used. In section **E**, the Close() and Dispose() (refer to Table 3.12) methods are utilized to finish this cleaning job.

Before we can finish this section and move to the next one, we need to discuss one more item, which is the DataReader Exceptions. Table 3.13 lists often used Exceptions.

You can use the try ... catch block to handle those Exceptions in your applications to avoid an unnecessary debug process as your project runs.

3.4.6 DataSet Component

The DataSet, which is an in-memory cache of data retrieved from a database, is a major component of the ADO.NET architecture. The DataSet consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. In other words, a DataSet object can be considered as a table-container that contains a set of data tables with the DataRelation as a bridge to relate all tables together. The relationship between a DataSet and a set of DataTable objects can be defined as follows:

- A DataSet class holds a data table collection, which contains a set of data tables or DataTable objects, and the Relations collection, which contains a set of DataRelation objects. This Relations collection sets up all relationships among those DataTable objects.
- A DataTable class holds the Rows collection, which contains a set of data rows or DataRow objects, and the Columns collection, which contains a set of data columns or DataColumn objects. The Rows collection contains all data rows in the data table and the Columns collection contains the actual schema of the data table.

The definition of the DataSet class is a generic idea, which means that it is not tied to any specific type of database. Data can be loaded into a DataSet by using a TableAdapter from many different databases such as Microsoft Access, Microsoft SQL Server, Oracle, Microsoft Exchange, Microsoft Active Directory, or any OLE DB or ODBC-compliant database.

Although not tied to any specific database, the DataSet class is designed to contain relational tabular data as one would find in a relational database.

Each table included in the DataSet is represented in the DataSet as a DataTable. The DataTable can be considered as a direct mapping to the real table in the database. For example, the LogIn data table, LogInDataTable, is a data table component or DataTable that can be mapped to the real table LogIn in the CSE_DEPT database. The relationship between any tables is realized in the DataSet as a DataRelation object. The DataRelation object provides the information that relates a child table to a parent table via a foreign key. A DataSet can hold any number of tables with any number of relationships defined between tables. From this point of view, a DataSet can be considered as a mini-database engine, so it can contain all information of tables it holds such as the column name and data type, all relationships between tables, and more important, it contains most management functionalities of the tables such as browse, select, insert, update, and delete data from tables.

A DataSet is a container and it keeps its data or tables in memory as XML files. In Visual Studio.NET 2003, when one wants to edit the structure of a DataSet, one must do that by editing an XML Schema or XSD file. Although there is a visual designer, the terminology and user interface are not consistent with a DataSet and its constituent objects.

With the Visual Studio.NET 2008, one can easily edit the structure of a DataSet and make any changes to the structure of that DataSet by using the DataSet Designer in the Data Source window. More important, one can graphically manipulate the tables and

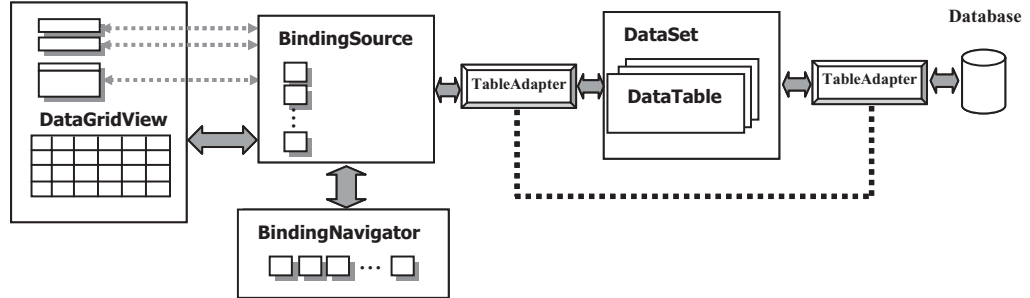
C# Window Form

Figure 3.15 Global representation of the DataSet and other data objects.

queries in a manner more directly tied to the DataSet rather than having to deal with an XML Schema (XSD).

Summarily, the DataSet object is a very powerful component that can contain multiple data tables with all information related to those tables. By using this object, one can easily browse, access, and manipulate data stored in it. We will explore this component in more detail in the following sections when a real project is built.

As we mentioned before, when you build a data-driven project and set up a connection between your project and a database by using ADO.NET, the data tables in the DataSet can be populated with data coming from your database by using the data query methods or the Fill method. From this point of view, you can consider the DataSet as a *data source*, and it contains all mapped data tables from the database you connected to your project. In some books, the terminology *data source* means the DataSet. Figure 3.15 shows a global relationship between the DataSet object, other data objects, and the Visual C# 2008 application.

A DataSet can be typed or untyped, and the difference between them is that the typed DataSet object has a schema and the untyped DataSet does not. In your data-driven applications, it is highly recommended to use the typed DataSet if that is possible because the typed DataSet has more support in Visual Studio 2008.

A typed DataSet object provides an easier way to access the content of the data table fields through strongly typed programming. The so-called strongly typed programming uses information from the underlying data scheme, which means that you can directly access and manipulate those data objects related to data tables. Another point is that a typed DataSet has a reference to an XML schema file, and this file has an extension of the .xsd. A complete description of the structure of all data tables included in the DataSet is provided in this schema file.

3.4.6.1 DataSet Constructor

The DataSet class has four public overloaded constructors, and Table 3.14 lists two most often used constructors.

The first constructor is used to create a new instance of the DataSet class with a blank parameter. The second constructor is used to create a new instance of the DataSet with the specific name of the new instance.

Table 3.14 Popular Constructors of the DataSet Class

Constructor	Functionality
DataSet()	Initializes a new instance of the DataSet class.
DataSet(String)	Initializes a new instance of a DataSet class with the given name.

Table 3.15 Public Properties of the DataSet Class

Property Name	Type	Functionality
DataSetName	String	Gets or sets the name of the current DataSet.
DefaultViewManager	DataViewManager	Gets a custom view of the data contained in the DataSet to allow filtering, searching, and navigating using a custom DataViewManager.
HasErrors	Boolean	Gets a value indicating whether there are errors in any of the DataTable objects within this DataSet .
IsInitialized	Boolean	Gets a value that indicates whether the DataSet is initialized.
Namespace	String	Gets or sets the namespace of the DataSet .
Tables	DataTableCollection	Gets the collection of tables contained in the DataSet .

3.4.6.2 DataSet Properties

The DataSet class has more than 15 public properties. Table 3.15 lists the most often used properties. Among these properties, the DataSetName, IsInitialized, and Tables are the most often used properties in your data-driven applications.

3.4.6.3 DataSet Methods

The DataSet class has more than 30 public methods. Table 3.16 lists the most often used methods.

Among those methods, the Clear(), Dispose(), and Merge() methods are often used. Before you can fill a DataSet, execute the Clear() method to clean up the DataSet to avoid any possible old data. Often in your applications, you need to merge other DataSets or data arrays into the current DataSet object by using the Merge() method. After you finished your data query or data action using the DataSet, you need to release it by executing the Dispose() method.

3.4.6.4 DataSet Events

The DataSet class has three public events and Table 3.17 lists these events.

The Disposed event is used to trigger the Dispose() method as this event occurs. The Initialized event is used to make a mark to indicate that the DataSet has been initialized

Table 3.16 Public Methods of the DataSet Class

Method Name	Functionality
BeginInit	Begins the initialization of a DataSet that is used on a form or used by another component. The initialization occurs at runtime.
Clear	Clears the DataSet of any data by removing all rows in all tables.
Copy	Copies both the structure and data for this DataSet .
Dispose	Releases the resources used by the MarshalByValueComponent.
GetChanges	Gets a copy of the DataSet containing all changes made to it since it was last loaded, or since AcceptChanges was called.
HasChanges	Gets a value indicating whether the DataSet has changes, including new, deleted, or modified rows.
Load	Fills a DataSet with values from a data source using the supplied IDataReader .
Merge	Merges a specified DataSet , DataTable , or array of DataRow objects into the current DataSet or DataTable .
Reset	Resets the DataSet to its original state. Subclasses should override Reset to restore a DataSet to its original state.
ToString	Returns a String containing the name of the Component, if any. This method should not be overridden.
WriteXml	Writes XML data, and optionally the schema, from the DataSet .
WriteXmlSchema	Writes the DataSet structure as an XML schema.

Table 3.17 Public Events of the DataSet Class

Event Name	Descriptions
Disposed	Adds an event handler to listen to the Disposed event on the component.
Initialized	Occurs after the DataSet is initialized.
Mergefailed	Occurs when a target and source DataRow have the same primary key value, and EnforceConstraints is set to true .

to your applications. The **Mergefailed** event is triggered when a conflict occurs and the **EnforceConstraints** property is set to *true* as you want to merge a **DataSet** with an array of **DataRow** objects, another **DataSet**, or a **DataTable**.

Before we can finish this section, we need to show how to create, initialize, and implement a real **DataSet** object in a data-driven application. A piece of code shown in Figure 3.16 is used to illustrate these issues, and an SQL Server Data Provider is utilized for this example. Assuming that an **SqlConnection** object, `sqlConnection`, has been created and initialized for this example.

Starting from step **A**, some initialization jobs are performed. An SQL **SELECT** statement is created, an **SqlCommand** object, an **SqlDataAdapter** object, and a **DataSet** object are also created. The integer variable `intValue` is used to hold the returned value from calling the `Fill()` method.


```

A string cmdString = "SELECT name, office, title, college FROM Faculty";
    SqlCommand sqlCommand = new SqlCommand();
    SqlDataAdapter sqlDataAdapter;
    DataSet sqlDataSet;
    int intValue = 0;

B sqlCommand.Connection = sqlConnection;
    sqlCommand.CommandType = CommandType.Text;
    sqlCommand.CommandText = cmdString;

C sqlDataAdapter.SelectCommand = sqlCommand;
    sqlDataSet = new DataSet();
    sqlDataSet.Clear();

D intValue = sqlDataAdapter.Fill(sqlDataSet);
    if (intValue == 0)
        MessageBox.Show("No valid faculty found!");

E sqlDataSet.Dispose();
    sqlDataAdapter.Dispose();
    sqlCommand.Dispose();

```

Figure 3.16 Example using DataSet.

In section **B**, the `SqlCommand` object is initialized by assigning the `SqlConnection` object to the `Connection` property, the `CommandType.Text` to the `CommandType` property, and `cmdString` to the `CommandText` property of the `SqlCommand` object.

The initialized `SqlCommand` object is assigned to the `SelectCommand` property of the `SqlDataAdapter` object in step **C**. Then a new `DataSet` object `sqlDataSet` is initialized and the `Clear()` method is called to clean up the `DataSet` object before it can be filled.

In step **D**, the `Fill()` method of the `SqlDataAdapter` object is executed to fill the `sqlDataSet`. If this fill is successful, which means that the `sqlDataSet` (exactly the `DataTable` in the `sqlDataSet`) has been filled by some data rows, the returned value should be greater than 0. Otherwise it means that some errors occurred for this fill and an error message will be displayed to warn the user.

Before the project can be completed, all resources used in this piece of code should be released and cleaned up. These cleaning jobs are performed in step **E** by executing some related method such as `Dispose()`.

You need to note that when the `Fill()` method is executed to fill a `DataSet`, the `Fill()` method retrieves rows from the data source using the `SELECT` statement specified by an associated `CommandText` property. The `Connection` object associated with the `SELECT` statement must be valid, but it does not need to be open. If the connection is closed before the `Fill()` is called, the `Fill()` method will open the connection and retrieve the data, then close the connection. If the connection is open before the `Fill()` is called, it still remains open.

The fill operation then adds rows to the destination, `DataTable` objects in the `DataSet`, and creates the `DataTable` objects if they do not already exist. When creating `DataTable` objects, the fill operation normally creates only the column named `metadata`. However, if the `MissingSchemaAction` property is set to `AddWithKey`, appropriate primary keys and constraints are also created.

If the `Fill()` returns the results of an `OUTER JOIN`, the `DataAdapter` does not set a `PrimaryKey` value for the resulting `DataTable`. You must explicitly define the primary key to ensure that duplicate rows are resolved correctly.

You can use the `Fill()` method multiple times on the same `DataTable`. If a primary key exists, incoming rows are merged with matching rows that already exist. If no primary key exists, incoming rows are appended to the `DataTable`.

3.4.7 DataTable Component

The `DataTable` class can be considered as a container that holds the `Rows` and `Columns` collections, and the `Rows` and `Columns` collections contain a set of rows (or `DataRow` objects) and a set of columns (or `DataColumn` objects) from a data table in a database. The `DataTable` is directly mapping to a real data table in a database or a data source, and it store its data in a mapping area, or a block of memory space that is associated to a data table in a database as the project runs. The `DataTable` object can be used in two ways as we mentioned in the previous sections. One way is that a group of `DataTable` objects, with each `DataTable` object mapped to a data table in the real database, can be integrated into a `DataSet` object. All of these `DataTable` objects can be populated by executing the `Fill()` method of the `DataAdapter` object (refer to the example in Section 3.4.6.4). The argument of the `Fill()` method is not a `DataTable` but a `DataSet` object since all `DataTable` objects are embedded into that `DataSet` object already. The second way to use the `DataTable` is that each `DataTable` can be considered as a single stand-alone data table object, and each table can be populated or manipulated by executing either the `ExecuteReader()` or the `ExecuteNonQuery()` method of the `Command` object.

The `DataTable` class is located in the `System.Data` namespace, and it is a `Data Provider`-independent component, which means that only one set of `DataTable` objects exist no matter what kind of `Data Provider` you are using in your applications.

The `DataTable` is a central object in the ADO.NET library. Other objects that use the `DataTable` include the `DataSet` and the `DataView`.

When accessing `DataTable` objects, note that they are conditionally case sensitive. For example, if one `DataTable` is named “faculty” and another is named “Faculty,” a string used to search for one of the tables is regarded as case sensitive. However, if “faculty” exists and “Faculty” does not, the search string is regarded as case insensitive. A `DataSet` can contain two `DataTable` objects that have the same `TableName` property value but different `Namespace` property values.

If you are creating a `DataTable` programmatically, you must first define its schema by adding `DataColumn` objects to the `DataColumnCollection` (accessed through the `Columns` property). To add rows to a `DataTable`, you must first use the `NewRow()` method to return a new `DataRow` object. The `NewRow()` method returns a row with the schema of the `DataTable`, as it is defined by the table’s `DataColumnCollection`. The maximum number of rows that a `DataTable` can store is 16,777,216.

The `DataTable` also contains a collection of `Constraint` objects that can be used to ensure the integrity of the data. The `DataTable` class is a member of the `System.Data` namespace within the .NET Framework class library. You can create and use a `DataTable` independently or as a member of a `DataSet`, and `DataTable` objects can also be used in conjunction with other .NET Framework objects, including the `DataView`. As we mentioned in the last section, you access the collection of tables in a `DataSet` through the `Tables` property of the `DataSet` object.

In addition to a schema, a `DataTable` must also have rows to contain and order data. The `DataRow` class represents the actual data contained in a table. You use the `DataRow` and its properties and methods to retrieve, evaluate, and manipulate the data in a table. As you access and change the data within a row, the `DataRow` object maintains both its current and original state.

3.4.7.1 *DataTable Constructor*

The `DataTable` has four overloaded constructors and Table 3.18 lists three of the most often used constructors. You can create a `DataTable` object by using the appropriate `DataTable` constructor. You can add it to the `DataSet` by using the `Add` method to add it to the `DataTable` object's `Tables` collection.

You can also create `DataTable` objects within a `DataSet` by using the `Fill()` or `FillSchema()` methods of the `DataAdapter` object, or from a predefined or inferred XML schema using the `ReadXml()`, `ReadXmlSchema()`, or `InferXmlSchema()` methods of the `DataSet`. Note that after you have added a `DataTable` as a member of the `Tables` collection of one `DataSet`, you cannot add it to the collection of tables of any other `DataSet`.

When you first create a `DataTable`, it does not have a schema (that is, a structure). To define the schema of the table, you must create and add `DataColumn` objects to the `Columns` collection of the table. You can also define a primary key column for the table and create and add `Constraint` objects to the `Constraints` collection of the table. After you have defined the schema for a `DataTable`, you can add rows of data to the table by adding `DataRow` objects to the `Rows` collection of the table.

You are not required to supply a value for the `TableName` property when you create a `DataTable`; you can specify the property at another time, or you can leave it empty. However, when you add a table without a `TableName` value to a `DataSet`, the table will be given an incremental default name of `TableN`, starting with "Table" for `Table0`.

Figure 3.17 shows an example of creating a new `DataTable` and a `DataSet` and then adding the `DataTable` into the `DataSet` object.

Table 3.18 Three Popular Constructors of the `DataTable` Class

Constructors	Descriptions
<code>DataTable()</code>	Initializes a new instance of the <code>DataTable</code> class with no arguments.
<code>DataTable(String)</code>	Initializes a new instance of the <code>DataTable</code> class with the specified table name.
<code>DataTable(String, String)</code>	Initializes a new instance of the <code>DataTable</code> class using the specified table name and namespace.

```
DataSet FacultyDataSet;
DataTable FacultyTable;

FacultyDataSet = new DataSet();
FacultyTable = new DataTable("Faculty");
FacultyDataSet.Tables.Add(FacultyTable);
```

Figure 3.17 Example of adding a `DataTable` into a `DataSet`.

Table 3.19 Popular Properties of the DataTable Class

Properties	Descriptions
Columns	The data type of the Columns property is DataColumnCollection, which means that it contains a collection of DataColumn objects. Each column in the DataTable can be considered as a DataColumn object. By calling this property, a collection of DataColumn objects that exists in the DataTable can be retrieved.
DataSet	Gets the DataSet to which this table belongs.
IsInitialized	Gets a value that indicates whether the DataTable is initialized.
Namespace	Gets or sets the namespace for the XML representation of the data stored in the DataTable.
PrimaryKey	Gets or sets an array of columns that function as primary keys for the data table.
Rows	The data type of the Rows property is DataRowCollection, which means that it contains a collection of DataRow objects. Each row in the DataTable can be considered as a DataRow object. By calling this property, a collection of DataRow objects that exists in the DataTable can be retrieved.
TableName	Gets or sets the name of the DataTable.

First, you need to create an instance for both the DataSet and the DataTable classes, respectively. Then you can add this new DataTable instance into the new DataSet object by using the Add() method.

3.4.7.2 DataTable Properties

The DataTable class has more than 20 properties. Table 3.19 lists some of the most often used properties. Among these properties, the Columns and Rows properties are very important to us, and both properties are collections of DataColumn and DataRow in the current DataTable object. The Columns property contains a collection of DataColumn objects in the current DataTable and each column in the table can be considered as a DataColumn object and can be added into this Columns collection. A similar situation happened to the Rows property. The Rows property contains a collection of DataRow objects that are composed of all rows in the current DataTable object. You can get the total number of columns and rows from the current DataTable by calling these two properties.

3.4.7.3 DataTable Methods

The DataTable class has about 50 different methods with 33 public methods, and Table 3.20 lists some of the most often used methods. Among these methods, three of them are important to us: NewRow(), ImportRow(), and LoadDataRow(). Calling the NewRow() adds a row to the data table using the existing table schema, but with default values for the row, and sets the DataRowState to Added. Calling the ImportRow() preserves the existing DataRowState along with other values in the row. Calling the LoadDataRow() is to find and update a data row from the current data table. This method

Table 3.20 Popular Methods of the DataTable Class

Methods	Descriptions
Clear	Clears the DataTable of all data.
Copy	Copies both the structure and data for this DataTable.
Dispose	Releases the resources used by the MarshalByValueComponent.
GetChanges	Gets a copy of the DataTable containing all changes made to it since it was last loaded or since AcceptChanges was called.
GetType	Gets the Type of the current instance.
ImportRow	Copies a DataRow into a DataTable, preserving any property settings, as well as original and current values.
Load	Fills a DataTable with values from a data source using the supplied IDataReader. If the DataTable already contains rows, the incoming data from the data source is merged with the existing rows.
LoadDataRow	Finds and updates a specific row. If no matching row is found, a new row is created using the given values.
Merge	Merge the specified DataTable with the current DataTable.
NewRow	Creates a new DataRow with the same schema as the table.
ReadXml	Reads XML schema and data into the DataTable.
RejectChanges	Rolls back all changes that have been made to the table since it was loaded or the last time AcceptChanges was called.
Reset	Resets the DataTable to its original state.
Select	Gets an array of DataRow objects.
ToString	Gets the TableName and DisplayExpression, if there is one as a concatenated string.
WriteXml	Writes the current contents of the DataTable as XML.

has two arguments, the Value (Object) and the Accept Condition (Boolean). The Value is used to update the data row if that row were found, and the Condition is used to indicate whether the table allows this update to be made or not. If no matching row is found, a new row is created with the given Value.

3.4.7.4 DataTable Events

The DataTable class contains 11 public events and Table 3.21 lists these events. The most often used events are ColumnChanged, Initialized, RowChanged, and RowDeleted. By using these events, one can track and monitor the real situations that occur in the DataTable.

Before we can finish this section, we need to show users how to create a data table and how to add data columns and rows into this new table. Figure 3.18 shows a complete example of creating a new data table object and adding columns and rows into this table. The data table is named FacultyTable.

Refer to Figure 3.18, starting from step **A**, a new instance of the data table FacultyTable is created and initialized to a blank table. In order to add data into this new table, you need to use the Columns and Rows collections, and these two collections

Table 3.21 Public Events of the DataTable Class

Events	Descriptions
ColumnChanged	Occurs after a value has been changed for the specified DataColumn in a DataRow.
ColumnChanging	Occurs when a value is being changed for the specified DataColumn in a DataRow.
Disposed	Adds an event handler to listen to the Disposed event on the component.
Initialized	Occurs after the DataTable is initialized.
RowChanged	Occurs after a DataRow has been changed successfully.
RowChanging	Occurs when a DataRow is changing.
RowDeleted	Occurs after a row in the table has been deleted.
RowDeleting	Occurs before a row in the table is about to be deleted.
TableCleared	Occurs after a DataTable is cleared.
TableClearing	Occurs when a DataTable is being cleared.
TableNewRow	Occurs when a new DataRow is inserted.

```

A //Create a new DataTable
  DataTable FacultyTable = new DataTable("FacultyTable");

B //Declare DataColumn and DataRow variables
  DataColumn column;
  DataRow row;

C //Create new DataColumn, set DataType, ColumnName and add to DataTable
  column = new DataColumn();
  column.DataType = System.Type.GetType("System.int32");
  column.ColumnName = "FacultyId";
  FacultyTable.Columns.Add(column);

D //Create another column.
  column = new DataColumn();
  column.DataType = Type.GetType("System.string");
  column.ColumnName = "FacultyOffice";
  FacultyTable.Columns.Add(column);

E //Create new DataRow objects and add to DataTable.
  int Index = 0;
  for (Index = 1; Index <= 10; Index++)
  {
F     row = FacultyTable.NewRow();
        row("FacultyId") = Index;
        row("FacultyOffice") = "TC- " + Index;
        FacultyTable.Rows.Add(row);
  }

```

Figure 3.18 Example of creating a new table and adding data into the table.

contain the DataColumn and DataRow objects. So next you need to create DataColumn and DataRow objects, respectively. Step **B** finished these object declarations.

In step **C**, a new instance of the DataColumn, column, is created by using the new keyword. Two DataColumn properties, DataType and ColumnName, are used to initialize the first DataColumn object with the data type as integer (System.int32) and with the column name as “FacultyId”, respectively. Finally the completed object of the DataColumn

Table 3.22 Completed FacultyTable

FacultyId	FacultyOffice
1	TC-1
2	TC-2
3	TC-3
4	TC-4
5	TC-5
6	TC-6
7	TC-7
8	TC-8
9	TC-9
10	TC-10

is added into the FacultyTable using the Add() method of the Columns collection class.

In step **D**, the second data column, with the column data type as string (System.string) and the column name as the “FacultyOffice”, is added into the FacultyTable in a similar way as we did for the first data column FacultyId.

In step **E**, a for loop is utilized to simplify the procedure of adding new data rows into this FacultyTable. First a loop counter Index is created, and a new instance of the DataRow is created with the method of the DataTable—NewRow(). In total we create and add 10 rows into this FacultyTable object. For the first column “FacultyId”, the loop counter Index is assigned to this column for each row. But for the second column “FacultyOffice”, the building name combined with the loop counter Index is assigned to this column for each row. Finally in step **F**, the DataRow object, row, is added into this FacultyTable using the Add() method that belongs to the Rows collection class.

When this piece of codes runs, a complete FacultyTable can be obtained and it should match the one shown in Table 3.22.

3.4.8 ADO.NET 3.5 Entity Framework

Most traditional databases use the relational model of data, such as Microsoft Access, SQL Server, and Oracle. But today almost all programming languages are object-oriented languages, and the object-oriented model of data structures are widely implemented in modern programs developed with those languages. Therefore, a potential contradiction exists between the relational model of data in databases and the object-oriented model of programming applied today. Although some new components were added into the ADO.NET 2.0 to try to solve this contradiction, still it does not give a full solution for this issue.

A revolutionary solution of this problem came with the release of ADO.NET 3.5 based on the .NET Framework 3.5 and the addition of Language Integrated Query (LINQ) to Visual Studio.NET 2008. The main contributions of the ADO.NET 3.5 include some new components: ADO.NET 3.5 Entity Framework (ADO.NET 3.5 EF) and ADO.NET 3.5 Entity Data Model Tools are added into ADO.NET 3.5. With these new components, the contradiction exists between the relational model of data used in databases and the object-oriented programming projects that can be fully resolved.

A primary goal of the ADO.NET 3.5 EF is to raise the level of abstraction available for data programming, thus simplifying the development of data-aware applications and enabling developers to write less code. The Entity Framework is the evolution of ADO.NET that allows developers to program in terms of the standard ADO.NET 3.5 abstraction or in terms of persistent objects (Object Relational Mapper ORM) and is built upon the standard ADO.NET 3.5 Provider model, which allows access to third-party databases. The Entity Framework introduces a new set of services around the Entity Data Model (EDM) (a medium for defining domain models for an application).

ADO.NET 3.5 provides an abstract database structure that converts the traditional logic database structure to an abstract or object structure with three layers:

- Conceptual layer
- Mapping layer
- Logical layer

ADO.NET 3.5 EF defines these three layers using a group of XML files, and these XML files provide a level of abstraction to enable users to program against the object-oriented Conceptual model instead of the traditional relational data model.

The Conceptual layer provides a way to allow developers to build object-oriented codes to access databases, and each component in databases can be considered as an object or entity in this layer. The Conceptual Schema Definition Language (CSDL) is used in those XML files to define entities and relationships that will be recognized and used by the Mapping layer to setup mapping between entities and relational data tables. The Mapping layer uses Mapping Schema Language (MSL) to establish mappings between entities in the Conceptual layer and the relational data structure in the Logical layer. The relational database schema is defined in an XML file using Store Schema Definition Language (SSDL) in the Logical layer. The Mapping layer works as a bridge or a converter to connect the Conceptual layer to the Logical layer and interpret between the object-oriented data model in the Conceptual layer and the relational data model in the Logical layer. This mapping, shown in Figure 3.19, allows users to code against the Conceptual layer and map those codes into the Logical layer.

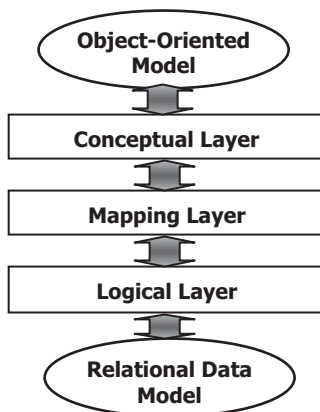


Figure 3.19 Mapping relationship between three layers.

A useful data component is provided by the Conceptual layer to enable users to develop object-oriented codes and it is called EntityClient. The EntityClient is a Data Provider with the associated components such as Connection (EntityConnection), Command (EntityCommand), and DataReader (EntityDataReader). The EntityClient is similar to other Data Providers we discussed in the previous sections in this chapter, but it includes new components and functionalities.

The core of ADO.NET 3.5 EF is its Entity Data Model (EDM), and the user can access and use this model using the ADO.NET 3.5 Entity Data Model Tools that includes the EDM item template, the EDM wizard, the EDM Designer, entity mapping details, and the entity model browser.

In the following sections, we will discuss the Entity Data Model and how to use these EDM Tools to create, build, and develop the Entity Data Model and implement it in actual data-driven applications.

First let's take a closer look at the ADO.NET 3.5 Entity Data Model.

3.4.8.1 ADO.NET 3.5 Entity Data Model

The ADO.NET 3.5 Entity Data Model (EDM) is a data model for defining application data as sets of entities and relationships to which common language runtime (CLR) types and storage structures can be mapped. This enables developers to create data access applications by programming against a conceptual application model instead of programming directly against a relational storage schema.

The following tools are designed to help you work with the EDM:

- The ADO.NET 3.5 Entity Data Model item template is available for Visual C# project type, and ASP.NET Web Site and Web Application projects, and launches the EDM Wizard.
- The EDM Wizard generates an EDM, which is encapsulated in an .edmx file. The wizard can generate the EDM from an existing database. The wizard also adds a connection string to the App.Config or Web.Config file and configures a single-file generator to run on the conceptual model contained in the .edmx file. This single-file generator will generate C# or VB code from the conceptual model defined in the .edmx file.
- The ADO.NET EDM Designer provides visual tools to view and edit the EDM graphically. You can open an .edmx file in the designer and create entities and map entities to database tables and columns.
- EdmGen.exe is a command-line tool that can be used to also generate models, validate existing models, and perform other functions on your EDM metadata files.

We will provide a detailed discussion for each of these tools in the following sections.

3.4.8.1.1 Entity Data Model Item Template The ADO.NET 3.5 EDM item template is the starting point to the EDM tools. The ADO.NET 3.5 EDM item template is available for Visual C# and Visual Basic project types. It can be added to Console Application, Windows Application, Class Library, ASP.NET Web Service Application, ASP.NET Web Application, or ASP.NET Web Site projects. You can add multiple ADO.NET 3.5 EDM items to the same project, with each item containing files that were generated from a different database and/or tables within the same database.

When you add the ADO.NET 3.5 EDM item template to your project, Visual Studio:

- Adds references to the System.Data, System.Data.Entity, System.Core, System.Security, and System.Runtime.Serialization assemblies if the project does not already have them.
- Starts the EDM Wizard. The wizard is used to generate an EDM from an existing database. The wizard creates an .edmx file, which contains the model information. You can use the .edmx file in the ADO.NET EDM Designer to view or modify the model.
- Creates a source code file that contains the classes generated from the conceptual model. The source code file is autogenerated and is updated when the .edmx file changes, and is compiled as part of the project.

Next let's have a look at the EDM Wizard.

3.4.8.1.2 Entity Data Model Wizard The EDM Wizard starts after you add an ADO.NET 3.5 Entity Data Model to your project. The wizard is used to generate an EDM. The wizard creates an .edmx file that contains the model information. The .edmx file is used by the ADO.NET 3.5 EDM Designer, which enables you to view and edit the mappings graphically.

You can select to create an empty model or to generate the model from an existing database. Generating the model from an existing database is the recommended practice for this release of the EDM tools.

The Wizard also creates a source code file that contains the classes generated from the CSDL information encapsulated in the .edmx file. The source code file is autogenerated and is updated when the .edmx file changes.

Depending on your selections, the Wizard will help you with the following steps:

- Choose the Model Contents: It is recommended that you select to generate the model from an existing database. The Wizard steps you through selecting the data source, database, and database objects to include in the EDM.
- Choose the Database Connection: You can choose an existing connection from the list of connections or click **New Database Connection** to open the **Connection Properties** dialog box and create a new connection to the database.
- Choose your Database Objects: You can select the tables, views, and stored procedures to include in the EDM.

Now let's have a look at the real part—ADO.NET 3.5 Entity Data Model Designer.

3.4.8.1.3 Entity Data Model Designer The ADO.NET 3.5 EDM Designer provides visual tools for creating and editing an EDM.

The ADO.NET EDM Designer includes the following components:

- A visual design surface for creating and editing the conceptual model. You can create, modify, or delete entities and associations.
- An Entity Mapping Details window to view and edit mappings. You can map entity types or associations to database tables and columns.
- An Entity Model Browser to give you a tree view of the EDM.
- Toolbox controls to create entities, associations, and inheritance relationships.

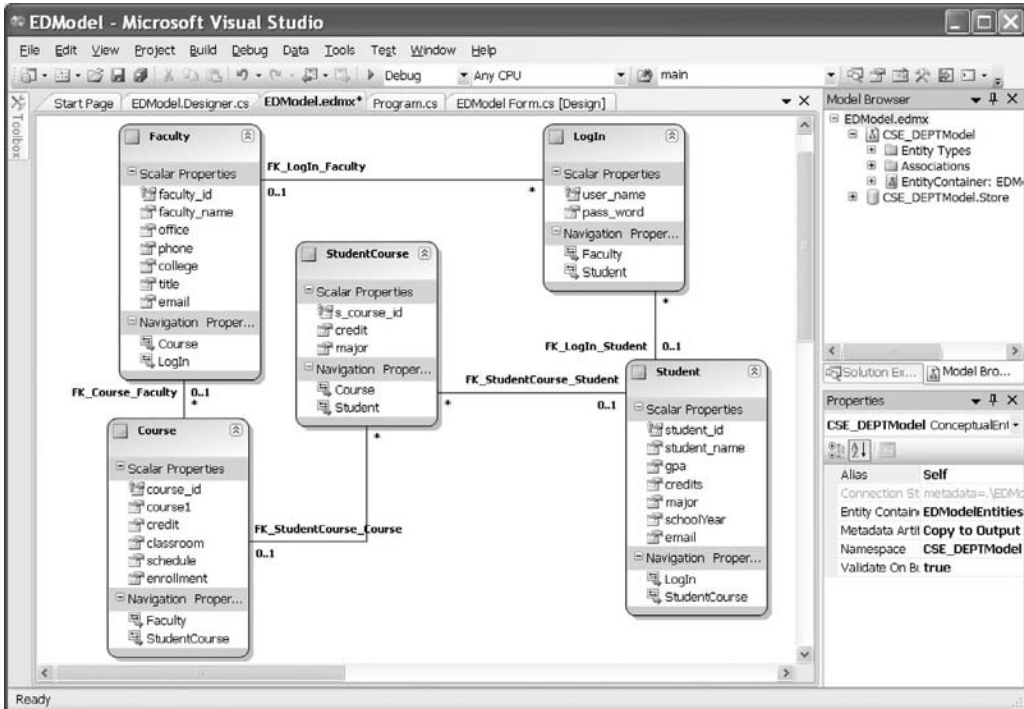


Figure 3.20 Example of the ADO.NET 3.5 Entity Data Model Designer.

The ADO.NET 3.5 EDM Designer is integrated with the Visual Studio.NET 2008 components. You can view and edit information using the Properties window and errors are reported in the Error List.

Figure 3.20 shows an example of the ADO.NET 3.5 EDM Designer. Two important functionalities of using the EDM Designer are:

Opening the ADO.NET Entity Data Model Designer The ADO.NET 3.5 EDM Designer is designed to work with an .edmx file. The .edmx file is an encapsulation of three EDM metadata artifact files, the CSDL, the SSDL, and the MSL files. When you run the EDM Wizard, an .edmx file is created and added to your solution. You open the ADO.NET EDM Designer by double-clicking on the .edmx file in the Solution Explorer.

Validating the EDM As you make changes to the EDM, the ADO.NET EDM Designer validates the modifications and reports errors in the Error List. You can also validate the EDM at any time by right-clicking on the design surface and selecting **Validate Model**.

3.4.8.1.4 Entity Model Browser The Entity Model Browser is a Visual Studio tool window that is integrated with the ADO.NET 3.5 EDM Designer. It provides a tree view of the EDM. The Entity Model Browser groups the information into two nodes.

The first node shows you the conceptual model. By expanding the underlying nodes, you can view all entity types and associations in the model.

The second node shows you the target database model. By expanding the underlying nodes you can see what parts of the database tables, views, and stored procedures have been imported into the model.

The EDM Browser enables you to do the following:

- Clicking on an item in the Entity Model Browser makes it active in the Properties window and the Entity Mapping Details View window. You can use these windows to modify the properties or entity mappings.
- Create a function to import a stored procedure.
- Update the SSDL information from the database.

The Entity Model Browser opens when the ADO.NET 3.5 EDM Designer is opened. If the Entity Model Browser is not visible, right-click on the main design surface and select **Show Entity Model Browser**.

3.4.8.2 Using ADO.NET 3.5 Entity Data Model Wizard

In this section, we will use a project example to illustrate how to use the EDM Wizard to develop a data-driven application to connect to our database, to create entity classes, to set up associations between entities, and to set up mapping relationships between entities and data tables in our database. Creating applications using the EDM can be significantly simplified by using the ADO.NET EDM item template and the EDM Wizard.

This section steps you through the following tasks:

- Create a new Visual C# Windows-based application.
- Use the EDM Wizard to select a data source and generate an EDM from our CSE_DEPT database.
- Use the entities in this application.

Let's begin with creating a new Visual C# Windows-based project named EDMModel.

3.4.8.2.1 Create New Visual C# Windows-Based Project Open Visual Studio. NET 2008 and select File|New|Project items to create a new project. Select the Visual C# as the project type and Windows Forms Applications as the Template for this new project. Enter EDMModel into the Name box and select any folder as the Location to save this project, then click the OK button to create this new project. Perform the following operations to change the properties of this new project:

1. Change the file object's name from Form1.cs to EDMModel Form.cs.
2. Change the Windows Form object's name from Form1 to EDMModelForm.
3. Change the content of the Text property of the Windows Form object from Form1 to Entity Data Model Form.
4. Change the StartPosition property of the form window to CenterScreen.
5. Add a Button control to the form window and name this button as cmdShow and set its Text property to 'Show Faculty'. Set its Font property to Bold—12.
6. Add a Listbox control to the form window and name it as FacultyList. Set its Font property to Bold—10.

Your finished EDMModelForm window should match the one shown in Figure 3.21.

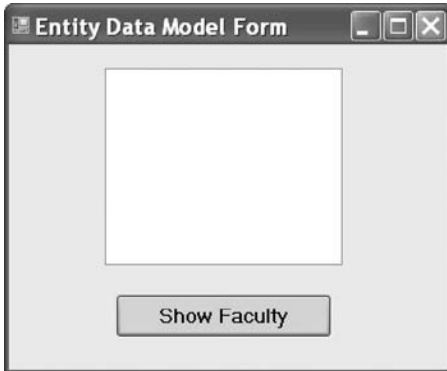


Figure 3.21 The EDMModelForm window.

Now let's generate our EDM Wizard using the EDM Tools. The ADO.NET EDM item template is the starting point for the EDM tools.

3.4.8.2.2 Generate Entity Data Model Files Before we can continue to generate the EDM files, we must first confirm whether we have installed ADO.NET 3.5 Entity Framework and ADO.NET 3.5 Entity Framework Tools in our computer. To do this confirmation, just right-click on the project EDMModel and select Add\New Item to open the Add New Item dialog box. If you cannot find the item ADO.NET EDM from the Templates box, this means that you have not installed ADO.NET 3.5 Entity Framework and its Tools. Therefore let's first download these components and install them on your computer.

Let's perform the following operations to complete the download and installation for these two components:

1. Open the Microsoft download home page: <http://www.microsoft.com/downloads>.
2. Select the item **Microsoft ADO.NET** from the **Recommended Downloads** box.
3. Two components need to be downloaded and installed from this page: **ADO.NET Entity Framework Beta 3** and **ADO.Net Entity Framework Tools Dec 07 CTP**.
4. Click the first component **ADO.NET Entity Framework Beta 3** and click the Download button associated with the file **EFB3SetupX86.exe**.
5. Click the Run button to complete this downloading and installation.
6. Click the Finish button to close this dialog box.
7. Return to the Microsoft ADO.NET page and click another component **ADO.Net Entity Framework Tools Dec 07 CTP**.
8. Click the Download and Run buttons to begin this downloading and installation processes.
9. You may encounter a potential bug displayed with an error message shown in Figure 3.22.
10. Click the OK button and open Internet Explorer and go to <http://go.microsoft.com/fwlink/?LinkID=104985>.
11. Click the Run button to complete this Visual Studio patch installation.

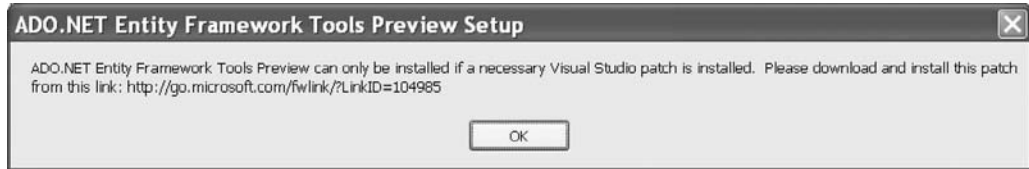


Figure 3.22 Error message related to install the Entity Framework Tools.

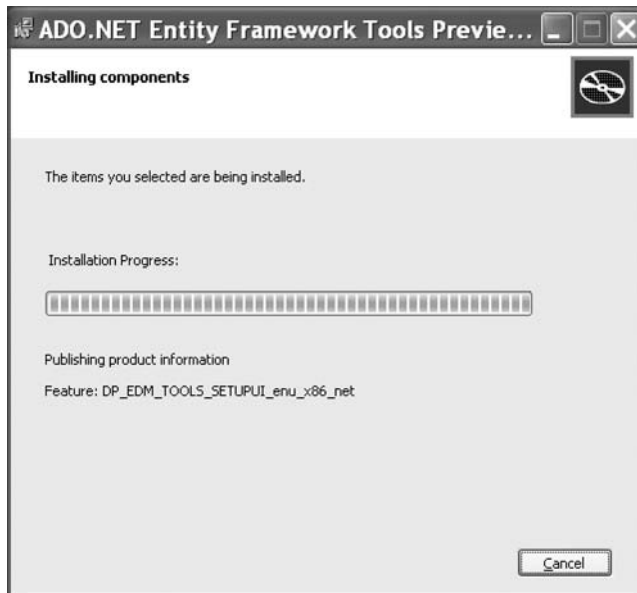


Figure 3.23 Installation process of ADO.NET EF Tools.

12. Then return to the Microsoft ADO.NET page and click the component **ADO.Net Entity Framework Tools Dec 07 CTP** to reinstall this EF Tools. The installation process is shown in Figure 3.23.

When the installation completes, click the Finish button as shown in Figure 3.24 to close this process. Now you have to reboot your computer to make these installations effective and available. Restart your computer to finish these installations. Now we can continue the process to generate the EDM Files and use the EDM Wizard.

Perform the following operations to generate our EDM Wizard:

1. Right-click on the project EDMModel from the Solution Explorer window and select the AddNew Item from the pop-up menu.
2. In the opened Add New Item dialog box, select the item ADOEntity Data Model from the Templates box and name it **EDMModel.edmx**. Your finished Add New Item dialog box should match the one shown in Figure 3.25. Click on the Add button to add this component to the new project.
3. The EDM Wizard is opened with two options: Generate from database and Empty model. Select the first item Generate from database since we want to create this EDM from our sample database CSE_DEPT. Click on the Next button to continue.



Figure 3.24 Last dialog box for installation of ADO.NET 3.5 EF Tools.

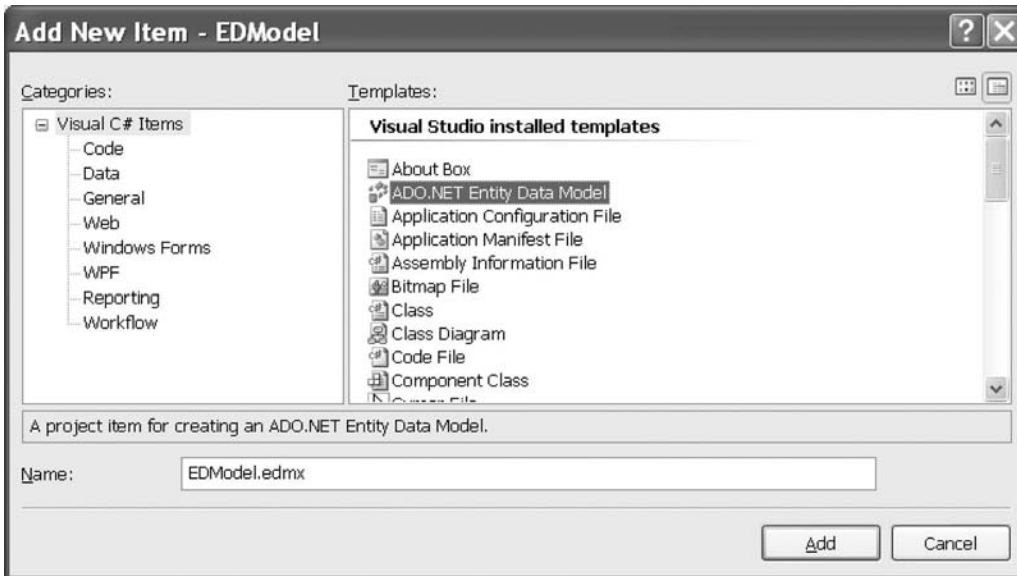


Figure 3.25 Add an ADO.NET Entity Data Model.

4. The next dialog box, Choose Your Data Connection, allows us to select our desired database to connect to. Click on the New Connection button to make a new connection. The Choose Data Source dialog box is displayed, which is shown in Figure 3.26.
5. In this application, we want to use a local SQL Server installed on our computer; therefore, select the second item: Microsoft SQL Server Database File and click on the Continue button to go to the next dialog box, Connection properties, which is shown in Figure 3.27a.

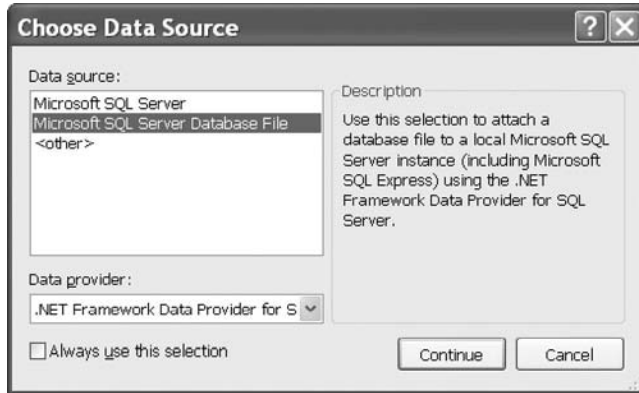


Figure 3.26 The Choose Data Source dialog box.

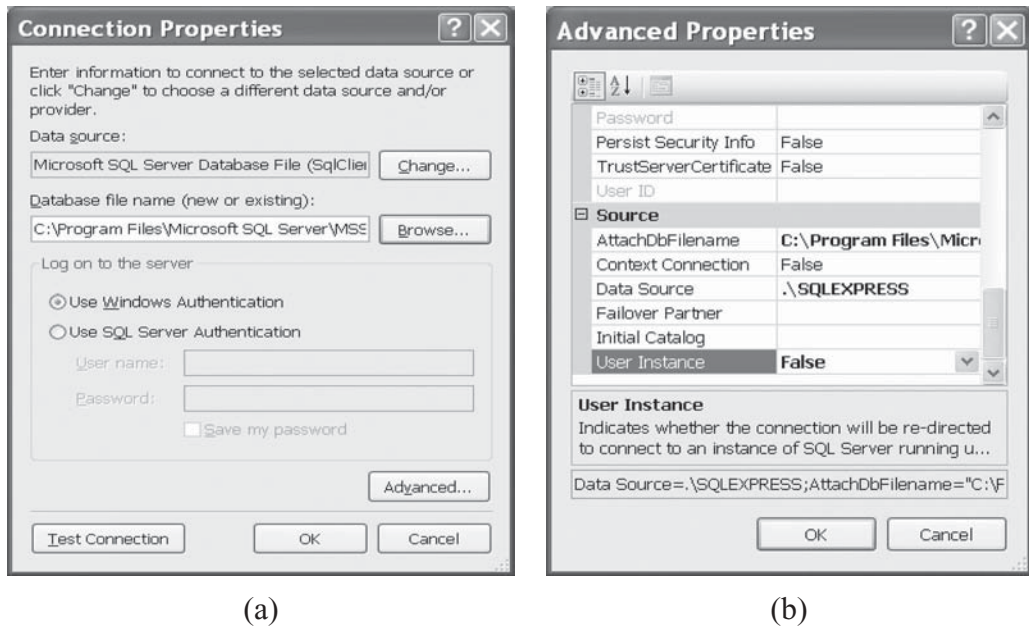


Figure 3.27 Connection properties and Advanced Properties dialog boxes.

6. Click the Browse button to find our desired database file CSE_DEPT.mdf that is located at the default folder of Microsoft SQL Server 2005 Express: C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data. Select this file and click on the Open button to add it into our project.
7. To avoid the possible conflict for duplicated installation of SQL Server Express with different versions, click on the Advanced button and change the property of User Instance from True to False, which is shown in Figure 3.27b. In this way, we can prevent the system from identifying this instance as a unique one. Click on the OK to return to Connection properties dialog box.

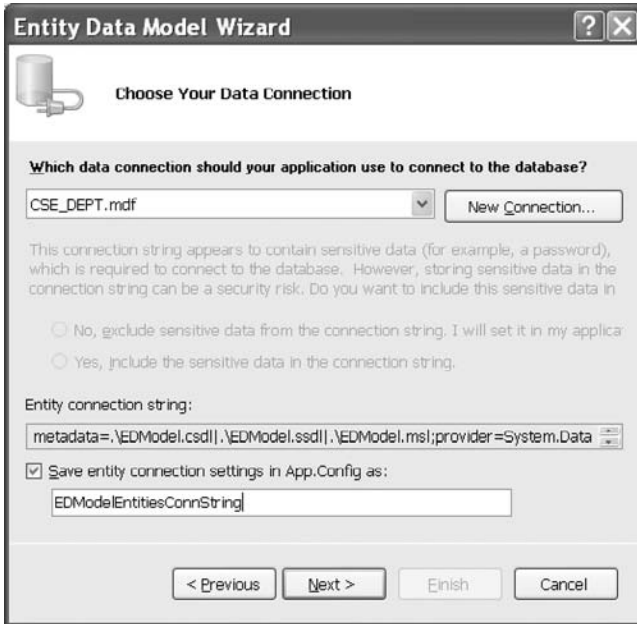


Figure 3.28 Choose Your Data Connection dialog box.

8. For the logon security, we prefer to use the default Windows Authentication mode. You can test this connection by clicking on the Test Connection button, and a successful connection message will be displayed if this connection is fine. Click on OK to go to the next step.
9. The Choose Your Data Connection dialog box appears again with all the settings we have created in the previous steps, which is shown in Figure 3.28.
10. Make sure that Save entity connection settings in App.Config is checked since we need this connection string when we access our database as the project runs. Also change this connection string to EDModelEntitiesConnString as shown in Figure 3.28. Click on the Next button to continue.
11. The Choose Your Database Objects dialog box appears, and this dialog box allows us to select our desired database objects such as tables, views, and stored procedures. Make sure to check both the Tables and the Stored Procedures checkboxes since we may need both of them. For the Views, which is optional and you can check it if you want to use this object to open and view the details for tables or stored procedures. But it will not hurt if you check it without using it later. So just check this object. Now expand the Tables and the Stored Procedures objects by clicking the small plus icon before each of them, and you can find all the data tables and stored procedures we developed for our sample database CSE_DEPT, which include five tables: LogIn, Faculty, Course, Student, and StudentCourse. Change the model name to CSE_DEPTModel. An example of this dialog box is shown in Figure 3.29.
12. Click on the Finish button to complete this process.

Now open the Solution Explorer window and you can find that an EDM named EDModel.edmx has been added into our project, which is shown in Figure 3.30.

To see this EDM in Designer view, double-click on the new added Entity Data Model EDModel.edmx. The Designer view is shown in Figure 3.31.



Figure 3.29 Example of Choose Your Database Objects dialog box.



Figure 3.30 Added EDMModel.

Five tables and connections between them are displayed in this view. On each table, two groups of entity properties are displayed, Scalar Properties and Navigation properties. The first category contains all entity properties (mapped to columns in our physical table), and the second category contains all related entities (mapped to related tables by using the primary and foreign keys) in this database. The connections between each entity (mapped to data table) are called associations.

As you double-click on this Entity Data Model EDMModel.edmx, another tool, Mapping Details, is also displayed under this Designer view, which is shown in Figure 3.32.

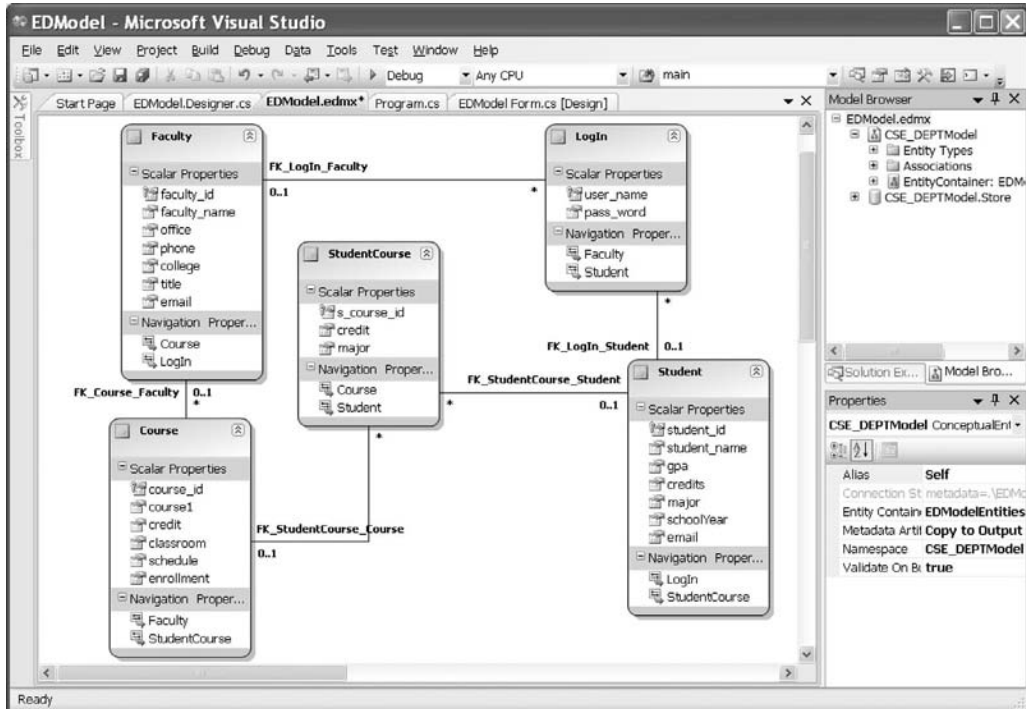


Figure 3.31 Designer view of the Entity Data Model EDMModel.

Column	Oper...	Value / Property
Tables		
Maps to Faculty		
<Add a Condition>		
Column Mappings		
faculty_id (nvarchar)	↔	faculty_id (String)
faculty_name (nvarchar)	↔	faculty_name (String)
office (text)	↔	office (String)
phone (text)	↔	phone (String)
college (text)	↔	college (String)
title (text)	↔	title (String)
email (text)	↔	email (String)
<Add a Table or View>		

Figure 3.32 Example of Mapping Details—Faculty entity.

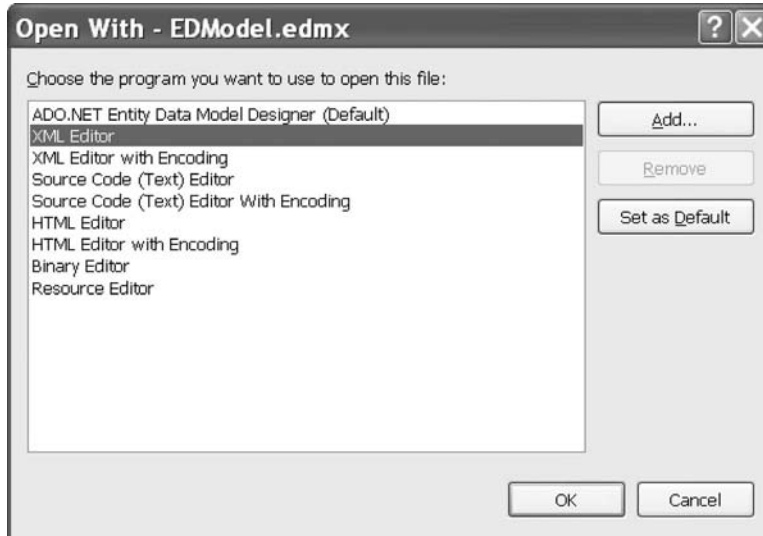


Figure 3.33 Open With dialog box.

If this Mapping Details did not open, you can open it by right-clicking on this Designer view and select the item Mapping Details from the pop-up menu. To see a Mapping Details, you also need to select an entity (table) to do it.

Besides these tools, an XML mapping file associated with our EDM EDModel is also created. To open this file, right-click on our new created EDM EDModel.edmx from the Solution Explorer window and select the item Open With to open the Open With dialog box, which is shown in Figure 3.33.

Select the item XML Editor and then click on the OK to open this XML mapping file. Now if you open the App.Config file, you can find that our connection string, EDModelEntitiesConnString created using the Entity Data Model Wizard, is under the <connectionStrings> tag in this file.

At this point, we have finished creating our EDM, and now we can use this model to build our Visual C# data-driven application to show readers how to make it work.

3.4.8.2.3 Use the ADO.NET 3.5 Entity Data Model Wizard The functionality of this project is that all faculty members in our Faculty table will be retrieved and displayed in the listbox control FacultyList as the user clicks the Show Faculty button as the project runs. Now let's use the EDM to perform the coding for the EDModelForm to realize this functionality.

The first coding is to add the namespace System.Data.EntityClient to the namespace declaration section of the code window of the EDModelForm object since we need to use this Data Provider that is defined in that namespace.

Then we need to do the coding for the Show Faculty button's Click method. Select the form object EDModel Form.cs from the Solution Explorer window and click the View Designer button to open its form window. Double-click on the Show Faculty button to open its Click method, and enter the codes are shown in Figure 3.34 into this method.

```

EDMModel.EDModelForm | cmdShow_Click()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
A using System.Data.EntityClient;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace EDMModel
{
    public partial class EDMModelForm : Form
    {
        public EDMModelForm()
        {
            InitializeComponent();
        }
        private void cmdShow_Click(object sender, EventArgs e)
        {
B         string cmdString = "SELECT fname.faculty_name FROM EDMModelEntitiesConnString.Faculty as fname";
C         EntityConnection Conn = new EntityConnection("name=EDMModelEntitiesConnString");
D         Conn.Open();
E         EntityCommand cmd = Conn.CreateCommand();
F         cmd.CommandText = cmdString;
G         EntityDataReader rd = cmd.ExecuteReader(CommandBehavior.SequentialAccess);
H         FacultyList.Items.Clear();
            while (rd.Read())
            {
                FacultyList.Items.Add(rd["faculty_name"]);
            }
I         Conn.Close();
        }
    }
}

```

Figure 3.34 Coding for the cmdShow_Click method.

Let's have a look at this piece of code to see how it works.

- A.** The namespace `System.Data.EntityClient` is added into the namespace declaration section of this code window to make sure that we can use this Data Provider.
- B.** The query string is defined first, and this string is different with those we used for SQL Server or Access databases. The `fname` is a nominal entity and the `Faculty` is the real entity that can be accessed via the connection string. The column we want to query is the `faculty_name` that is mapped to an entity property in this query string. The `FROM` clause is composed of `EntityContainer.EntitySet`, therefore the connection string that represents the `EntityContainer` is prefixed before the table `Faculty` that is exactly an `EntitySet`.
- C.** An `EntityConnection` object is created here to replace either a `SqlConnection` or `OleDbConnection` object with the connection string as the argument. You can copy this connection string from the `App.Config` file if you like.
- D.** The `Open()` method is executed to open this connection.
- E.** An `EntityCommand` instance `cmd` is created using the `CreateCommand()` method based on the `Connection` object. Then the `Command` object is initialized by assigning the query string `cmdString` to the `CommandText` property.

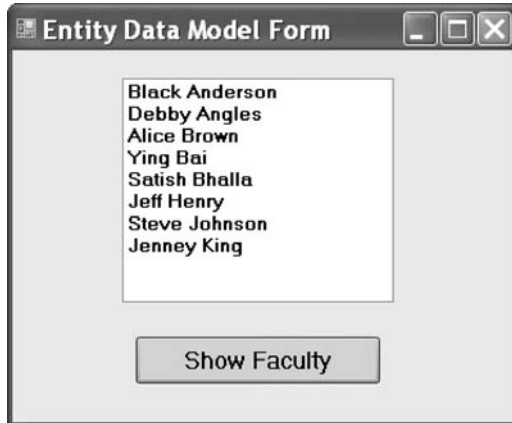


Figure 3.35 Running result of the project EDMModel.

- F.** The `ExecuteReader()` method is called to retrieve back all `faculty_name` and assign them to the `EntityDataReader` object.
- G.** The listbox control `FacultyList` is cleaned up before it can be filled.
- H.** A while loop is utilized to pick up all `faculty_name` from the `EntityDataReader` and add each of them into the `FacultyList` control by using the `Add()` method. The point is that all `faculty_name` is read out using the `SequentialAccess` mode; therefore, all data are read out and stored in a collection or an array in the `EntityDataReader`. In Visual C#, a square bracket is used to indicate each element in a collection or an array. Also since we created a nontyped `DataSet`, each column or entity property must be clearly indicated with the name of the column or the entity.
- I.** Finally the connection is closed to release the connection object.

Now let's run the project to test our codes. Click on the Start Debugging button to run the project. The `EDMModelForm` window is displayed as shown in Figure 3.35. Click on the Show Faculty button to connect to our sample database and retrieve back all faculty names. The running result is shown in Figure 3.35.

Click on the Close button located at the upper-right corner of this form to close our project. It can be found from this piece of code that it is relatively simple and easy to use the EDM to access and manipulate data against the database.

3.5 CHAPTER SUMMARY

The main topic of this chapter is an introduction to the ADO.NET, which includes the architectures, organizations, and components of the ADO.NET 2.0 and ADO.NET 3.5.

Detailed discussions and descriptions are provided in this chapter to give readers both fundamental and practical ideas and pictures of how to use components in ADO.NET 2.0 and ADO.NET 3.5 to develop professional data-driven applications. Two ADO.NET 2.0 architectures are discussed to enable users to follow the directions to design and build their preferred projects based on the different organizations of the ADO.NET 2.0.

A history of the development of ADO.NET 2.0 is first introduced in this chapter. Different data-related objects are discussed such as Data Access Object (DAO), Remote Data Object (RDO), Open Database Connectivity (ODBC), OLE DB, and the ADO. The difference between the ADO and the ADO.NET is provided in detail.

Fundamentally, the ADO.NET is a class container and it contains three basic components: Data Provider, DataSet, and DataTable. Furthermore, the Data Provider contains four subcomponents: Connection, Command, TableAdapter, and DataReader. You should keep in mind that the Data Provider comes in multiple versions based on the type of database you are using in your applications. So from this point of view, all four subcomponents of the Data Provider are called Data Provider–dependent components. The popular versions of the Data Provider are:

- OLE DB Data Provider
- ODBC Data Provider
- Microsoft SQL Server Data Provider
- Oracle Data Provider

Each version of Data Provider is used for one specific database. One exception is that both OLE DB and ODBC Data Providers can work for some other databases, such as Microsoft Access, Microsoft SQL Server, and Oracle databases. In most cases, you should use the matched version of the Data Provider for a specific database. Even the OLE DB and ODBC can work for that kind of database since the former can provide more efficient processing technique and faster accessing and manipulating speed compared with the latter.

To access and manipulate data in databases, you can use one of two ADO.NET 2.0 architectures: You can use the DataAdapter to access and manipulate data in the DataSet that is considered as a DataTables collector by executing some properties of the DataAdapter, such as SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand. Alternatively, you can treat each DataTable as a single table object and access and manipulate data in each table by executing the different methods of the Command object, such as ExecuteReader and ExecuteNonQuery.

A key point in using the Connection object of the Data Provider to set up connection between your applications and your data source is the connection string, which has a different format and style depending on the database you are using. The popular components of the connection string include Provider, Data Source, Database, User ID, and Password. But some connection strings only use a limited number of components, such as the Data Provider for the Oracle database.

An important point in using the Command object to access and manipulate data in your data source is the Parameter component. The Parameter class contains all properties and methods that can be used to set up specific parameters for the Command object. Each Parameter object contains a set of parameters, and each Parameter object can be assigned to the Parameters collection that is one property of the Command object.

The latest version of ADO.NET, ADO.NET 3.5 is discussed with some examples in the last section in this chapter. The properties and functionalities of the ADO.NET 3.5 Entity Framework (EF) and ADO.NET 3.5 Entity Framework Tools (EFT) are discussed in detail. The core of ADO.NET 3.5 EF, Entity Data Model, and associated Item

template, Wizard and Designer, is also discussed and analyzed with a real project example EDMModel.

By finishing this chapter, you should be able to:

- Understand the architecture and organization of the ADO.NET 2.0.
- Understand three components of the ADO.NET 2.0, such as the Data Provider, DataSet, and the DataTable.
- Use the Connection object to connect to a Microsoft Access, Microsoft SQL Server, and Oracle database.
- Use the Command and Parameter objects to select, insert, and delete data using a string variable containing an SQL statement.
- Use the DataAdapter object to fill a DataSet using the Fill method.
- Read data from the data source using the DataReader object.
- Read data from the DataTable using the SelectCommand property of the DataAdapter object.
- Create DataSet and DataTable objects and add data into the DataTable object.
- Understand the ADO.NET 3.5 Entity Framework (EF) and ADO.NET 3.5 Entity Framework Tools (EFT).
- Understand the ADO.NET 3.5 Entity Data Model (EDM) and associated Item template, Wizard, and Designer.
- Create and implement ADO.NET 3.5 Entity Data Model Tools to develop professional data-driven applications in Visual C# 2008 environment

In Chapter 4, we will discuss a new technique, Language Integrated Query (LINQ), that was released with ADO.NET 3.5 and .NET Framework 3.5 in Visual Studio.NET 2008. With the help of this new technique, the operational process of the data queries and manipulations with different data sources can be significantly simplified and the efficiency of the data actions against the data sources can be greatly improved.

HOMEWORK

I. True/False Selections

- ___ 1. ADO.NET 2.0 is composed of four major components: Data Provider, DataSet, DataReader, and DataTable.
- ___ 2. ADO is developed based on Object Linking and Embedding (OLE) and Component Object Model (COM) technologies.
- ___ 3. ADO.NET 3.5 is a new version of ADO.NET and it is based mainly on the Microsoft .NET Framework 2.0.
- ___ 4. The Connection object is used to set up a connection between your data-driven application and your data source.
- ___ 5. Both OLE DB and ODBC Data Providers can work for the SQL Server and Oracle databases.
- ___ 6. Different ADO.NET components are located at the different namespaces. The DataSet and DataTable are located at the System.Data namespace.

- ___7. The DataSet can be considered as a container that contains multiple data tables, but those tables are only a mapping of the real data tables in the database.
- ___8. The ExecuteReader() method is a data query method that can only be used to execute a read-out operation from a database.
- ___9. Both SQL Server and Oracle Data Providers used a so-called Named Parameter Mapping technique.
- ___10. The DataTable object is a Data Provider-independent object.

II. Multiple Choices

1. To populate data from a database to a DataSet object, one needs to use the _____.
 - a. Data Source
 - b. DataAdapter (TableAdapter)
 - c. Runtime object
 - d. Wizards
2. The Parameters property of the Command class _____.
 - a. Is a Parameter object
 - b. Contains a collection of Parameter objects
 - c. Contains a Parameter object
 - d. Contains the parameters of the Command object
3. To add a Parameter object to the Parameters property of the Command object, one needs to use the _____ method that belongs to the _____.
 - a. Insert, Command
 - b. Add, Command
 - c. Insert, Parameters collection
 - d. Add, Parameters collection
4. DataTable class is a container that holds the _____ and _____ objects.
 - a. DataTable, DataRelation
 - b. DataRow, DataColumn
 - c. DataRowCollection, DataColumnCollection
 - d. Row, Column
5. The _____ is a property of the DataTable class, and it is also a collection of DataRow objects. Each DataRow can be mapped to a _____ in the DataTable.
 - a. Rows, column
 - b. Columns, column
 - c. Row, row
 - d. Rows, row
6. The _____ data provider can be used to execute the data query for _____ data providers.
 - a. SQL Server, OleDb and Oracle
 - b. OleDb, SQL Server and Oracle
 - c. Oracle, SQL Server and OleDb
 - d. SQL Server, Odbc and Oracle

7. To perform a Fill() method to fill a data table, it executes _____ object with suitable parameters.
 - a. DataAdapter
 - b. Connection
 - c. DataReader
 - d. Command
8. The DataReader is a read-only class, and it can only be used to retrieve and hold the data rows returned from a database when executing a(n) _____ method.
 - a. Fill
 - b. ExecuteNonQuery
 - c. ExecuteReader
 - d. ExecuteQuery
9. One needs to use the _____ method to release all objects used for a data-driven application before one can exit the project.
 - a. Release
 - b. Nothing
 - c. Clear
 - d. Dispose
10. To _____ data between the DataSet and the database, the _____ object should be used.
 - a. Bind, BindingSource
 - b. Add, TableAdapter
 - c. Move, TableAdapter
 - d. Remove, DataReader

III. Exercises

1. Explain two architectures of the ADO.NET 2.0 and illustrate the functionality of these two architectures using block diagrams.
2. List three basic components of the ADO.NET 2.0 and the different versions of the Data Provider as well as their subcomponents.
3. Explain the relationship between the Command and Parameter objects. Illustrate how to add Parameter objects to the Parameters collection that is a property of the Command object using an example. Assuming that an SQL Server Data Provider is used with two parameters: parameter_name: username, password, parameter_value: “NoName”, “ComeBack”.
4. Explain the relationship between the DataSet and DataTable. Illustrate how to use the Fill method to populate a DataTable in the DataSet. Assume that the data query string is an SQL SELECT statement: “SELECT faculty_id, name FROM Faculty”, and an SQL Server Data Provider is utilized.
5. Explain the components and functionalities of ADO.NET 3.5 Entity Framework (EF) and ADO.NET 3.5 Entity Framework Tools (EFT).
6. Illustrate the relationship between the ADO.NET 3.5 Entity Data Model and its associated components, such as Item template, Wizard, and Designer.
7. Explain the relationship between three layers of the ADO.NET 3.5 Entity Data Model, Conceptual layer, Mapping layer, and Logical layer.
8. Explain the associations between entities listed in Figure 3.31.

Chapter 4

Introduction to Language-Integrated Query (LINQ)

Language-Integrated Query (LINQ) is a groundbreaking innovation in Visual Studio 2008 and the .NET Framework version 3.5 that bridges the gap between the world of objects and the world of data. Traditionally, queries about data are expressed as simple strings without type checking at compile time or IntelliSense support. Furthermore, you have to learn a different query language for each type of data source: Microsoft Access, SQL databases, XML documents, various Web services, and Oracle databases. LINQ makes a query as a first-class language construct in C# and Visual Basic. You write queries about strongly typed collections of objects by using language keywords and familiar operators.

In Visual Studio.NET you can write LINQ queries in C# with SQL Server databases, XML documents, ADO.NET DataSets, and any collection of objects that supports `IEnumerable` or the generic `IEnumerable<T>` interface. As we mentioned in Chapter 3, LINQ support for the ADO.NET 3.5 Entity Framework is also planned, and LINQ providers are being written by third parties for many Web services and other database implementations.

You can use LINQ queries in new projects or alongside non-LINQ queries in existing projects. The only requirement is that the project be developed under the .NET Framework 3.5 environment.

Before we can dig deeper into LINQ, we had better have a general and global picture about LINQ. Let's start from the basic introduction about LINQ.

4.1 OVERVIEW OF LANGUAGE-INTEGRATED QUERY

The LINQ pattern is established on the basis of a group of methods called Standard Query Operators (SQO). Most of these methods operate on sequences, where a sequence is an object whose type implements the `IEnumerable<T>` interface or the `IQueryable<T>` interface. The standard query operators provide query capabilities including filtering, projection, aggregation, sorting, and more.

All SQO methods are located at the namespace `System.Linq`. To use these methods, one must declare this namespace with a directive like: `using System.Linq` in the namespace declaration section of the code windows.

There are some confusing signs and terminologies, such as `IEnumerable`, `IEnumerable<T>`, `IQueryable`, and `IQueryable<T>` interfaces. Let's have a closer look at these terminologies first.

4.1.1 Some Special Interfaces Used in LINQ

Four interfaces, `IEnumerable`, `IEnumerable<T>`, `IQueryable`, and `IQueryable<T>`, are widely used in LINQ queries via SQO. In fact, two interfaces, `IEnumerable` and `IQueryable`, are mainly used for the nongeneric collections supported by the earlier versions of C#, such as C# 1.0 or earlier. The other two interfaces, `IEnumerable<T>` and `IQueryable<T>`, are used to convert the data type of collections compatible with those in the `System.Collection.Generic` in C# 2.0 to either `IEnumerable<T>` (LINQ to Objects) or `IQueryable<T>` (LINQ to SQL) since LINQ uses a stronger typed collection or sequence as the data sources, and any data in those data sources must be converted to this stronger typed collection before the LINQ can be implemented. Most LINQ queries are performed on arrays or collections that implement the `IEnumerable<T>` or `IEnumerable` interfaces. But a LINQ to SQL query is performed on classes that implement the `IQueryable<T>` interface. The relationship between the `IEnumerable<T>` and the `IQueryable<T>` interfaces is `IQueryable<T>` implements `IEnumerable<T>`. Therefore, besides the SQO, the LINQ to SQL queries have additional query operators since it uses the `IQueryable<T>` interface.

4.1.1.1 `IEnumerable` and `IEnumerable<T>` Interfaces

The `IEnumerable<T>` interface is a key part of LINQ to Objects, and it allows all of the C# 2.0 generic collection classes to implement it. This interface permits the enumeration of a collection's elements. All of the collections in the `System.Collections.Generic` namespace support the `IEnumerable<T>` interface. Here T means the converted data type of the sequence or collection. For example, if you have an `IEnumerable` of `int`, expressed by `IEnumerable<int>`, you have a sequence or a collection of `ints`.

For nongeneric collections existing in the old version of C#, such as C# 1.0 or older, they support the `IEnumerable` interface, but they do not support the `IEnumerable<T>` interface because of the stronger typed property of the latter. Therefore, you cannot directly call those SQO methods whose first argument is an `IEnumerable<T>` using nongeneric collections. However, you can still perform LINQ queries using those collections by calling the `Cast` or `OfType` SQO to generate a sequence that implements `IEnumerable<T>`.

A coding example of using LINQ to Object is shown in Figure 4.1.

The type `IEnumerable<int>` plays two important roles in this piece of code.

1. The query expression has a data source called `intArray`, which implements `IEnumerable<int>`.
2. The query expression returns an instance of `IEnumerable<int>`.

Every LINQ to Objects query expression, including the one shown in Figure 4.1, will begin with a line of this type:

```
// create an integer array
int[] myArray = new int[] {1, 2, 3, 4, 5 };
IEnumerable<int> intArray = myArray.Select(i => i);
//LINQ query expression
var query = from num in intArray
            where num >= 3
            select num;

foreach (var intResult in query)
{
    Console.WriteLine(intResult);
}
```

Figure 4.1 Coding example of using LINQ to Object query.

```
// create an integer array
int[] myArray = new int[] {1, 2, 3, 4, 5 };
IEnumerable<int> query = from num in myArray
                        where num >= 3
                        select num;

foreach (var intResult in query)
{
    Console.WriteLine(intResult);
}
```

Figure 4.2 Modification of the coding example of using LINQ to Objects query.

from x in y

In each case, the data source represented by the variable **y** must support the `IEnumerable<T>` interface. As you have already seen, the array of integers shown in this example supports that interface.

The query shown in Figure 4.1 can also be rewritten as shown in Figure 4.2.

This code makes explicit the type of variable returned by this query, `IEnumerable<int>`. In practice, you will find that most LINQ to Objects queries return `IEnumerable<T>` for different data type **T**.

By finishing these two examples, it should be clear to you that interfaces `IEnumerable` and `IEnumerable<T>` play a key role in LINQ to Objects queries. The former is used for the nongeneric collections and the latter is for the generic collections. The point is that a typical LINQ to Objects query expression not only takes a class that implements `IEnumerable<T>` as its data source, but it also returns an instance with the same type.

4.1.1.2 IQueryable and IQueryable<T> Interfaces

As we discussed in the previous section, `IQueryable` and `IQueryable<T>` are two interfaces used for LINQ to SQL queries. Similar to `IEnumerable` and `IEnumerable<T>` interfaces, in which the SQO methods are defined as the static members in the `Enumerable` class, the SQO methods applied for the `IQueryable<T>` interface are defined as static members of the `Queryable` class. The `IQueryable` interface is mainly used for the nongeneric collections and the `IQueryable<T>` is used for generic collections. Another point is

```

//create a database connection using the DataContext object
public CSE_DEPTDataContext cse_dept = new CSE_DEPTDataContext();

//create local string variables
string username = string.Empty;
string password = string.Empty;

//LINQ query expression
IQueryable<LogIn> loginfo = from lg in cse_dept.LogIns
                           where lg.user_name == txtUserName.Text &&
                                lg.pass_word == txtPassWord.Text
                           select lg;

foreach (LogIn log in loginfo)
{
    username = log.user_name;
    password = log.pass_word;
}

```

Figure 4.3 Coding example of using LINQ to SQL query.

that the `IQueryable<T>` interface is inherited from the `IEnumerable<T>` interface from the `Queryable` class and the definition of this interface is:

```
interface IQueryable<T> : IEnumerable<T>, Queryable
```

From this inheritance, one can treat an `IQueryable<T>` sequence as an `IEnumerable<T>` sequence.

Figure 4.3 shows an example of using the `IQueryable` interface to perform a query to a sample database `CSE_DEPT`. A database connection has been made using the `DataContext` object before this piece of codes can be executed. The `LogIn` is the name of a table in this sample database, and it has been converted to an entity before the LINQ query can be performed. An `IQueryable<T>` interface, specifically a Standard Query Operator, is utilized to perform this query. The `LogIn` works as a type in the `IQueryable<T>` interface to make sure that both input sequence and returned sequence are strongly typed sequences with the type of `LogIn`. The Standard Query Operator fetches and returns the matched sequence and assigns them to the associated string variables using the `foreach` loop.

Now let's have a closer look at the Standard Query Operator (SQO).

4.1.2 Standard Query Operators

There are two sets of LINQ Standard Query Operators, one that operates on objects of type `IEnumerable<T>` and the other that operates on objects of type `IQueryable<T>`. The methods that make up each set are static members of the `Enumerable` and `Queryable` classes, respectively. They are defined as extension methods of the type on which they operate. This means that they can be called by using either static method syntax or instance method syntax.

In addition, several SQO methods operate on types other than those based on `IEnumerable<T>` or `IQueryable<T>`. The `Enumerable` type defines two such methods that both operate on objects of type `IEnumerable`. These methods, `Cast<TResult>` (`IEnumerable`) and `OfType<TResult>` (`IEnumerable`), let you enable a nonparameter-

ized, or nongeneric, collection to be queried in the LINQ pattern. They do this by creating a strongly typed collection of objects. The Queryable class defines two similar methods, `Cast<TResult>(IQueryable)` and `OfType<TResult>(IQueryable)`, which operate on objects of type Queryable.

The standard query operators differ in the timing of their execution, depending on whether they return a singleton value or a sequence of values. Those methods that return a singleton value (e.g., `Average` and `Sum`) execute immediately. Methods that return a sequence defer the query execution and return an enumerable object.

In the case of the methods that operate on in-memory collections, that is, those methods that extend `IEnumerable<T>`, the returned enumerable object captures the arguments that were passed to the method. When that object is enumerated, the logic of the query operator is employed and the query results are returned.

In contrast, methods that extend `IQueryable<T>` do not implement any querying behavior but build an expression tree that represents the query to be performed. The query processing is handled by the source `IQueryable<T>` object.

Calls to query methods can be chained together in one query, which enables queries to become arbitrarily complex. According to its functionality, the Standard Query Operator can be divided into two categories: Deferred Standard Query Operators and Nondeferred Standard Query Operators. Table 4.1 lists some of the most often used Standard Query Operators.

Because of the limitation of the space, we will select some of the most often used SQO methods and give a detailed discussion of them one by one.

Table 4.1 Most Often Used Standard Query Operators

Standard Query Operator	Purpose	Deferred
All	Quantifiers	No
Any	Quantifiers	No
AsEnumerable	Conversion	Yes
Average	Aggregate	No
Cast	Conversion	Yes
Distinct	Set	Yes
ElementAt	Element	No
First	Element	No
Join	Join	Yes
Last	Element	No
OfType	Conversion	Yes
OrderBy	Ordering	Yes
Select	Projection	Yes
Single	Element	No
Sum	Aggregate	No
ToArray	Conversion	No
ToList	Conversion	No
Where	Restriction	Yes

4.1.3 Deferred Standard Query Operators

Both deferred Standard Query Operators and nondeferred operators are organized based on their purpose, and we start this discussion based on the alphabet order.

AsEnumerable (Conversion Purpose)

The `AsEnumerable` operator method has no effect other than to change the compile-time type of *source* from a type that implements `IEnumerable<T>` to `IEnumerable<T>` itself. This means that if an input sequence has a type of `IEnumerable<T>`, the output sequence will also be converted to one that has the same type, `IEnumerable`. An example coding of using this operator is shown in Figure 4.4.

The key point for this query structure is the operator `AsEnumerable()`. Since different database systems use different collections and query operators, therefore those collections must be converted to a type of `IEnumerable<T>` in order to use the LINQ technique because all data operations in LINQ use SQO methods that can perform complex data queries on an `IEnumerable<T>` sequence. A compiling error would be encountered without this operator.

Cast (Conversion Purpose)

A `Cast` operator provides a method for explicit conversion of the type of an object in an input sequence to an output sequence with a specific type. The compiler treats *cast-expression* as type *type-name* after a type cast has been made. A point to be noticed is that the `Cast` operator method works on the `IEnumerable` interface, not the `IEnumerable<T>` interface, and it can convert any object with an `IEnumerable` type to `IEnumerable<T>` type. An example coding of using this operator is shown in Figure 4.5.

```
FacultyDataAdapter.SelectCommand = accCommand;
FacultyDataAdapter.Fill(ds, "Faculty");
var facultyinfo = (from fi in ds.Tables["Faculty"].AsEnumerable()
                  where fi.Field<string>("faculty_name").Equals(ComboName.Text)
                  select fi);
foreach (var fRow in facultyinfo)
{
    //Display selected fRow elements...
}
```

Figure 4.4 Example coding for the operator `AsEnumerable`.

```
System.Collections.ArrayList fruits = new System.Collections.ArrayList();
fruits.Add("apple");
fruits.Add("mango");

IEnumerable<string> query = fruits.Cast<string>().Select(fruit => fruit);

foreach (string fruit in query)
    Console.WriteLine(fruit);

// the running result of this piece of codes is:

apple
mango
```

Figure 4.5 Example coding for the operator `Cast`.

Join (*Join Purpose*)

A join of two data sources is the association of objects in one data source with objects that share a common attribute in another data source. Joining is an important operation in queries that target data sources whose relationships to each other cannot be followed directly. In object-oriented programming, this could mean a correlation between objects that is not modeled, such as the backward direction of a one-way relationship. An example of a one-way relationship is a Customer class that has a property of type City, but the City class does not have a property that is a collection of Customer objects. If you have a list of City objects and you want to find all the customers in each city, you could use a join operation to find them.

The join methods provided in the LINQ framework are Join and GroupJoin. These methods perform equijoins, or joins that match two data sources based on the equality of their keys. In relational database terms, Join implements an inner join, a type of join in which only those objects that have a match in the other data set are returned. The GroupJoin method has no direct equivalent in relational database terms, but it implements a superset of inner joins and left outer joins. A left outer join is a join that returns each element of the first (left) data source, even if it has no correlated elements in the other data source. An example coding of using this operator is shown in Figure 4.6.

The issue is that we want to query all courses (course_id) taught by the selected faculty from the Course table based on the faculty_name. But the problem is that there is no faculty_name column in the Course table, and only the faculty_id is associated with related course_id. Therefore we have to get the faculty_id from the Faculty table first based on the faculty_name, and then query the course_id from the Course table based on the queried faculty_id. This problem can be effectively solved by using the join operator method shown in Figure 4.6.

OfType (*Conversion Purpose*)

This operator method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its GetEnumerator method directly or by using **foreach** in Visual C#. An example coding of using this operator is shown in Figure 4.7.

The OfType<T>(IEnumerable) method returns only those elements in *source* that can be cast to type *TResult*. To instead receive an exception if an element cannot be cast to type *TResult*, use Cast<T>(IEnumerable).

```
var courseinfo = Courses.
    Join (Faculty, ci =>ci.faculty_id, o => o.faculty_id).
    Where (fi.faculty_name == ComboName.Text).
    Select new
    {
        course_id = ci.course_id
    };
foreach (var cid in courseinfo)
{
    CourseList.Items.Add(cid.course_id);
}
```

Figure 4.6 Example coding for the operator Join.

```

System.Collections.ArrayList fruits = new System.Collections.ArrayList(2);
fruits.Add("Mango");
fruits.Add("Orange");

// Apply OfType() to the ArrayList.
IEnumerable<string> query = fruits.OfType<string>();

Console.WriteLine("Elements of type 'string' are:");
foreach (string fruit in query)
    Console.WriteLine(fruit);

// the running result of this piece of codes is:

Elements of type 'string' are:
Mango
Orange

```

Figure 4.7 Example coding for the operator `OfType`.

```

public static void OrderByEx()
{
    Pet[] pets = { new Pet { Name="Barley", Age=8 },
                  new Pet { Name="Boots", Age=4 },
                  new Pet { Name="Whiskers", Age=1 } };

    IEnumerable<Pet> query = pets.OrderBy(pet => pet.Age);

    foreach (Pet pet in query)
        Console.WriteLine("{0} - {1}", pet.Name, pet.Age);
}

// the running result of this piece of codes is:

Whiskers - 1
Boots - 4
Barley - 8

```

Figure 4.8 Example coding for the operator `OrderBy`.

This method is one of the few standard query operator methods that can be applied to a collection that has a nonparameterized type, such as an `ArrayList`. This is because `OfType<TResult>` extends the type `IEnumerable`. `OfType<TResult>` cannot only be applied to collections that are based on the parameterized `IEnumerable<T>` type, but collections that are based on the nonparameterized `IEnumerable` type also.

By applying `OfType<TResult>` to a collection that implements `IEnumerable`, you gain the ability to query the collection by using the Standard Query Operators. For example, specifying a type argument of `Object` to `OfType<TResult>` would return an object of type `IEnumerable<Object>` in C#, to which the standard query operators can be applied.

OrderBy (Ordering Purpose)

This operator method is used to sort the elements of an input sequence in ascending order based on the `keySelector` method. The output sequence will be an ordered one in a type of `IOrderedEnumerable<T>`. Both `IEnumerable` and `IQueryable` classes contain this operator method. An example coding of using this operator is shown in Figure 4.8.

```

IEnumerable<int> squares = Enumerable.Range(1, 5).Select(x => x * x);
foreach (int num in squares)
    Console.WriteLine(num);

// the running result of this piece of codes is:
1
4
9
16
25

```

Figure 4.9 Example coding for the operator **Select**.

```

List<string> fruits = new List<string> { "apple", "banana", "mango", "orange",
                                       "blueberry", "grape", "strawberry" };

IEnumerable<string> query = fruits.Where(fruit => fruit.Length < 6);
foreach (string fruit in query)
    Console.WriteLine(fruit);

// the running result of this piece of codes is:
apple
mango
grape

```

Figure 4.10 Example coding for the operator **Where**.

Select (Projection Purpose)

Both `IEnumerable` and `IQueryable` classes contain this operator method. This operator method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using **foreach** in Visual C#.

This projection method requires the transform function, *selector*, to produce one value for each value in the source sequence, *source*. If *selector* returns a value that is itself a collection, it is up to the consumer to traverse the subsequences manually. In such a situation, it might be better for your query to return a single coalesced sequence of values. To achieve this, use the `SelectMany` method instead of `Select`. Although `SelectMany` works similarly to `Select`, it differs in that the transform function returns a collection that is then expanded by `SelectMany` before it is returned.

In query expression syntax, a **select** in Visual C# clause translates to an invocation of `Select`. An example coding of using this operator is shown in Figure 4.9.

Where (Restriction Purpose)

Both `IEnumerable` and `IQueryable` classes contain this operator method. This method is implemented by using deferred execution. The immediate return value is an object that stores all the information that is required to perform the action. The query represented by this method is not executed until the object is enumerated either by calling its `GetEnumerator` method directly or by using **foreach** in Visual C#. An example coding of using this operator is shown in Figure 4.10.

```
// Create a string array
string[] names = { "Hartono, Tommy", "Adams, Terry", "Andersen, Henriette", "Hedlund, Magnus", "Ito, Shu" };
string name = names.ElementAt(2);

Console.WriteLine("The name chosen at random is '{0}'.", name);

// the running result of this piece of codes is:
Andersen, Henriette
```

Figure 4.11 Example coding for the operator `ElementAt`.

```
// Create a string array
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54, 83, 23, 87, 435, 67, 12, 19 };
int firstNum = numbers.First();

Console.WriteLine(firstNum);

// the running result of this piece of codes is:
9
```

Figure 4.12 Example coding for the operator `First`.

In query expression syntax, a **where** in Visual C# clause translates to an invocation of `Where<TSource>IEnumerable<TSource>, Func<TSource, Boolean>`.

4.1.4 Nondeferred Standard Query Operators

Some of the most often used nondeferred SQO methods are discussed in this section.

ElementAt (*Element Purpose*)

This operator method returns the element at a specified index in a sequence. If the type of *source* implements `ICollection<T>`, that implementation is used to obtain the element at the specified index. Otherwise, this method obtains the specified element.

This method throws an exception if *index* is out of range. To instead return a default value when the specified index is out of range, use the `ElementAtOrDefault<TSource>` method. An example coding of using this operator is shown in Figure 4.11.

First (*Element Purpose*)

This operator method returns the first element of an input sequence. The method `First<TSource>(IEnumerable<TSource>)` throws an exception if the source contains no elements. To instead return a default value when the source sequence is empty, use the `FirstOrDefault` method. An example coding of using this operator is shown in Figure 4.12.

Last (*Element Purpose*)

This operator method returns the last element of a sequence. The method `Last<TSource>(IEnumerable<TSource>)` throws an exception if *source* contains no

```
int[] numbers = { 9, 34, 65, 92, 87, 435, 3, 54, 83, 23, 87, 67, 12, 19 };
int last = numbers.Last();
Console.WriteLine(last);
// the running result of this piece of codes is 19
```

Figure 4.13 Example coding for the operator Last.

```
string[] fruits = { "orange" };
string fruit1 = fruits.Single();
Console.WriteLine(fruit1);
// the running result of this piece of codes is orange
```

Figure 4.14 Example coding for the operator Single.

elements. To instead return a default value when the source sequence is empty, use the LastOrDefault method. An example coding of using this operator is shown in Figure 4.13.

Single (Element Purpose)

This operator method returns a single, specific element of an input sequence of values. The Single<TSource>(IEnumerable<TSource>) method throws an exception if the input sequence is empty. To instead return **nullNothingnullptr** null reference when the input sequence is empty, use SingleOrDefault. An example coding of using this operator is shown in Figure 4.14.

ToArray (Conversion Purpose)

This operator method converts a collection to an array. This method forces query execution. The ToArray<TSource>(IEnumerable<TSource>) method forces immediate query evaluation and returns an array that contains the query results. You can append this method to your query in order to obtain a cached copy of the query results. An example coding of using this operator is shown in Figure 4.15.

ToList (Conversion Purpose)

This operator method converts a collection to a List<T>. This method forces query execution. The ToList<TSource>(IEnumerable<TSource>) method forces immediate query evaluation and returns a List<T> that contains the query results. You can append this method to your query in order to obtain a cached copy of the query results. An example coding of using this operator is shown in Figure 4.16.

Now we have finished a detailed discussion about the SQO methods, and they are actual methods to be executed to perform a LINQ query. Next we will discuss the LINQ query. We organize this part in the following sequence. First we will provide an introduction about the LINQ query. Then we divide this discussion into seven sections:

```

public class ToArrayClass
{
    public static void Main()
    {
        string[] sArray = {"G", "H", "a", "H", "over", "Jack"};

        string[] names = sArray.OfType<string>().ToArray();
        foreach (string name in names)
            Console.WriteLine(name);
    }
}
// the running result of this piece of codes is:
GHaHoverJack

```

Figure 4.15 Example coding for the operator ToArray.

```

string[] fruits = { "apple", "banana", "mango", "orange", "blueberry", "grape", "strawberry" };
List<int> lengths = fruits.Select(fruit => fruit.Length).ToList();
foreach (int length in lengths)
    Console.WriteLine(length);
// the running result of this piece of codes is:
5
6
5
6
9
5
10

```

Figure 4.16 Example coding for the operator ToList.

1. Architecture and components of LINQ
2. LINQ to Objects
3. LINQ to DataSet
4. LINQ to SQL
5. LINQ to Entities
6. LINQ to XML
7. C# 3.0 Language Enhancement for LINQ

Three components—LINQ to DataSet, LINQ to SQL, and LINQ to Entities—belong to LINQ to ADO.NET. Now let's start with the first part, introduction to LINQ query.

4.2 INTRODUCTION TO LINQ QUERY

A query is basically an expression that retrieves data from a data source. Queries are usually expressed in a specialized query language such as Microsoft Access, SQL Server, Oracle, or XML document. Different languages have been developed over time for the

various types of data sources, for example, SQL for relational databases and XQuery for XML. Therefore, developers have had to learn a new query language for each type of data source or data format that they must support. LINQ simplifies this situation by offering a consistent model for working with data across various kinds of data sources and formats. In a LINQ query, you are always working with objects. You use the same basic coding patterns to query and transform data in XML documents, SQL databases, ADO.NET DataSets, .NET collections, and any other format for which a LINQ provider is available.

LINQ can be considered as a pattern or model that is supported by a collection of so-called SQO methods we discussed in the last section, and all those SQO methods are static methods defined in either `IEnumerable` or `IQueryable` classes in the namespace `System.Linq`. The data operated in LINQ query are object sequences with the data type of either `IEnumerable<T>` or `IQueryable<T>`, where T is the actual data type of the objects stored in the sequence.

From another point of view, LINQ can also be considered as a converter or bridge that sets up a mapping relationship between the abstract objects implemented in Standard Query Operators and the physical relational databases implemented in the real world. It is the LINQ that allows developers direct access and the ability to manipulate data in different databases using objects with the same basic coding patterns. With the help of LINQ, the headache caused by learning and using different syntaxes, formats, and query structures for different data sources in order to access and query them can be removed. The efficiency of database queries can be significantly improved and the query process can also be greatly simplified.

Structurally, all LINQ query operations consist of three distinct actions:

1. Obtain the data source.
2. Create the query.
3. Execute the query.

In order to help you to have a better understanding about LINQ and its running process, let's have an example to illustrate how the three parts of a query operation are expressed in source code. The example uses an integer array as a data source for convenience; however, the same concepts apply to other data sources, too. The example code is shown in Figure 4.17.

The exact running process of this piece of code is shown in Figure 4.18.

The key point is: In LINQ, the execution of the query is distinct from the query itself; in other words when you create a query in step 2, you have not retrieved any data and the real data query occurs in step 3, Query Execution using the `foreach` loop.

Let's have a closer look at this piece of code and the mapped process to have a clear picture about the LINQ query and its process.

The Data Source used in this example is an integer array *numbers*, and it implicitly supports the generic `IEnumerable<T>` interface. This fact means it can be queried with LINQ. A query is executed in a `foreach` statement, and `foreach` requires `IEnumerable` or `IEnumerable<T>`. Types that support `IEnumerable<T>` or a derived interface such as the generic `IQueryable<T>` are called queryable types.

The Query specifies what information to retrieve from the data source or sources. Optionally, a query also specifies how that information should be sorted, grouped, and

```

static void IntroLINQ()
{
    // The Three Parts of a LINQ Query:
    // 1. Data source.
    int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

    // 2. Query creation. The numQuery is an IEnumerable<int>
    var numQuery = from num in numbers
                  where (num % 2) == 0
                  select num;

    // 3. Query execution.
    foreach (int num in numQuery)
    {
        Console.WriteLine("{0,1} ", num);
    }
}
// the running result of this piece of codes is:
0, 2, 4, 6

```

Figure 4.17 Example coding for the LINQ query.

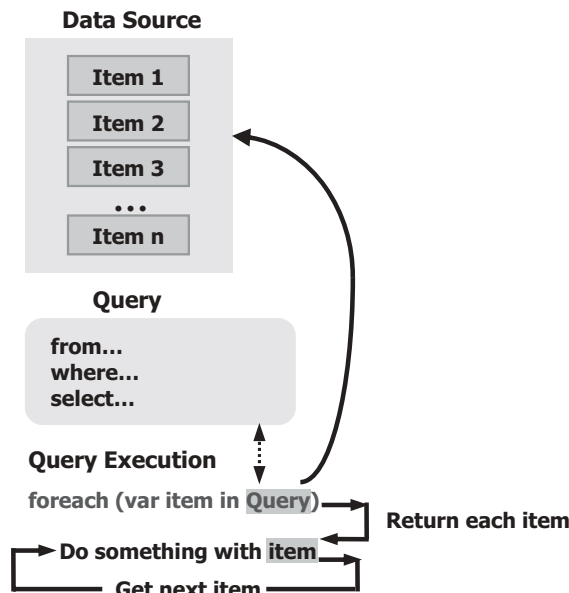


Figure 4.18 Running process of a LINQ query.

shaped before it is returned. A query is stored in a query variable and initialized with a query expression. To make it easier to write queries, C# has introduced new query syntax. A typical basic form of the query expression is shown in Figure 4.19.

Three clauses, **from**, **where**, and **select**, are mostly used for most LINQ queries. The query used in this example returns all the even numbers from the integer array. The query expression contains three clauses: **from**, **where**, and **select**. If you are familiar with SQL, you will have noticed that the ordering of the clauses is reversed from the order in SQL.


```

from [identifier] in [data source]
let [expression]
where [boolean expression]
order by [[expression](ascending/descending)], [optionally repeat]
select [expression]
group [expression] by [expression] into [expression]

```

Figure 4.19 Typical query expression of LINQ query.

The **from** clause specifies the data source, the **where** clause applies the filter, and the **select** clause specifies the type of the returned elements. For now, the important point is that in LINQ, the query variable itself takes no action and returns no data. It just stores the information that is required to produce the results when the query is executed at some later point.

The **Query Execution** in this example is a deferred execution since all operator methods used in this query are deferred operators (refer to Table 4.1).

The **foreach** statement with an iteration variable *num* is used for this query execution to pick up each item from the data source and assign it to the variable *num*. A Console.WriteLine() method is executed to display each received data item, and this query process will continue until all data items have been retrieved from the data source.

Because the query variable itself never holds the query results, you can execute it as often as you like. For example, you may have a database that is being updated continually by a separate application. In your application, you could create one query that retrieves the latest data, and you could execute it repeatedly at some interval to retrieve different results every time.

Queries that perform aggregation functions over a range of source elements must first iterate over those elements. Examples of such queries are **Count**, **Max**, **Average**, and **First**. These execute without an explicit **foreach** statement because the query itself must use **foreach** in order to return a result. Note also that these types of queries return a single value, not an IEnumerable collection. To force immediate execution of any query and cache its results, you can call the ToList<TSource> or ToArray<TSource> methods. You can also force execution by putting the **foreach** loop immediately after the query expression. However, by calling ToList or ToArray you also cache all the data in a single collection object.

4.3 ARCHITECTURE AND COMPONENTS OF LINQ

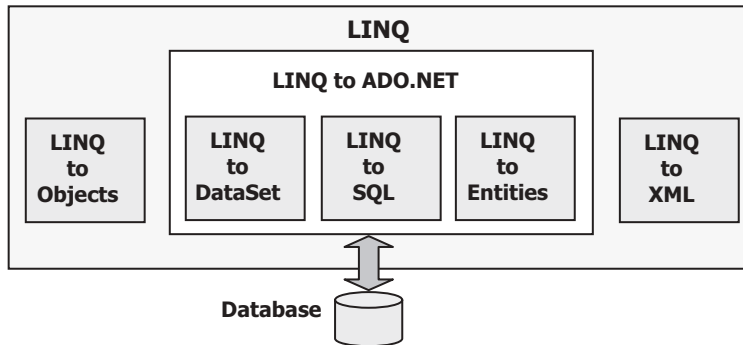
LINQ is composed of three major components: LINQ to Objects, LINQ to ADO.NET, and LINQ to XML. A detailed organization of the LINQ can be written as:

1. LINQ to Objects
2. LINQ to ADO.NET (LINQ to DataSet, LINQ to SQL, and LINQ to Entities)
3. LINQ to XML

All of three components are located at the different namespaces provided by .NET Framework 3.5, which is shown in Table 4.2.

Table 4.2 LINQ-Related Namespaces

Namespace	Purpose
System.Linq	Classes and interfaces that support LINQ queries are located at this namespace.
System.Collections.Generic	All components related to IEnumerable and IEnumerable<T> are located at this namespace (LINQ to Objects).
System.Data.Linq	All classes and interfaces related to LINQ to SQL are defined in this namespace.
System.XML.Linq	All classes and interfaces related to LINQ to XML are defined in this namespace.
System.Data.Linq.Mapping	Map a class as an entity class associated with a physical database.

**Figure 4.20** Typical LINQ architecture.

A typical LINQ architecture is shown in Figure 4.20. Now let's give a brief introduction for each component in LINQ.

4.3.1 Overview of LINQ to Objects

The LINQ to Objects component refers to the use of LINQ queries with any IEnumerable or IEnumerable<T> collection directly, without the use of an intermediate LINQ provider or API such as LINQ to SQL or LINQ to XML. The actual LINQ queries are performed by using the SQO methods that are static methods of the static System.Linq.Enumerable class that you used to create LINQ to Objects queries. You can use LINQ to query any enumerable collections such as List<T>, Array, or Dictionary<TKey, TValue>. The collection may be user defined or may be returned by a .NET Framework API.

In a basic sense, LINQ to Objects represents a new approach to collections, which includes arrays and in-memory data collections. The old way, you had to write complex foreach loops that specified how to retrieve data from a collection. In the LINQ approach, you write declarative code that describes what you want to retrieve.

In addition, LINQ queries offer three main advantages over traditional foreach loops:

1. They are more concise and readable, especially when filtering multiple conditions.
2. They provide powerful filtering, ordering, and grouping capabilities with a minimum of application code.
3. They can be ported to other data sources with little or no modification.

In general, the more complex the operation you want to perform on the data, the more benefit you will realize by using LINQ instead of traditional iteration techniques.

4.3.2 Overview of LINQ to DataSet

LINQ to DataSet belongs to LINQ to ADO.NET, and it is a subcomponent of LINQ to ADO.NET. LINQ to DataSet makes it easier and faster to query data cached in a DataSet object. Specifically, LINQ to DataSet simplifies querying by enabling developers to write queries from the programming language itself, instead of by using a separate query language. This is especially useful for Visual Studio developers, who can now take advantage of the compile-time syntax checking, static typing, and IntelliSense support provided by the Visual Studio in their queries.

LINQ to DataSet can also be used to query data that has been consolidated from one or more data sources. This enables many scenarios that require flexibility in how data is represented and handled, such as querying locally aggregated data and middle-tier caching in Web applications. In particular, generic reporting, analysis, and business intelligence applications require this method of manipulation.

The LINQ to DataSet functionality is exposed primarily through the extension methods in the DataRowExtensions and DataTableExtensions classes. LINQ to DataSet builds on and uses the existing ADO.NET 2.0 architecture and is not meant to replace ADO.NET 2.0 in application code. Existing ADO.NET 2.0 code will continue to function in a LINQ to DataSet application. The relationship of LINQ to DataSet to ADO.NET 2.0 and the data store is illustrated in Figure 4.21.

It can be found from Figure 4.21 that LINQ to DataSet is built based on ADO.NET 2.0 and uses all its components, which include Connection, Command, DataAdapter, and DataReader. The advantage of this structure is that all developers using ADO.NET 2.0 can continue their database implementations and developments without problems.

4.3.3 Overview of LINQ to SQL

LINQ to SQL belongs to LINQ to ADO.NET, and it is a subcomponent of LINQ to ADO.NET. LINQ to SQL is a component of .NET Framework version 3.5, which provides a runtime infrastructure for managing relational data as objects. As we discussed in Chapter 3, in LINQ to SQL, the data model of a relational database is mapped to an object model expressed in the programming language of the developer with three layers. When the application runs, LINQ to SQL translates into SQL the language-integrated queries in the object model and sends them to the database for execution. When the database returns the results, LINQ to SQL translates them back to objects that you can work with in your own programming language.

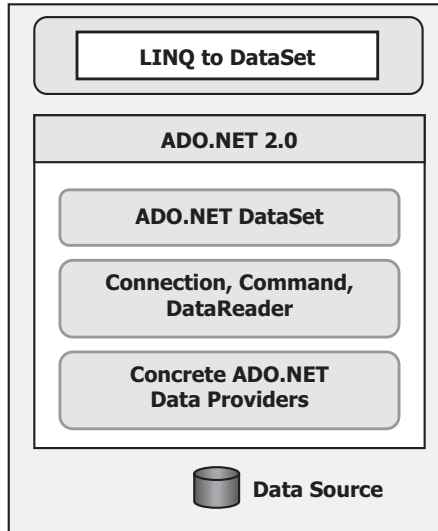


Figure 4.21 Relationship between LINQ to DataSet and ADO.NET 2.0.

Two popular LINQ to SQL Tools, SQLMetal and Object Relational Designer, are widely used in developing applications using LINQ to SQL. The SQLMetal provides a DOS-like template with a black-white window. Developers using Visual Studio typically use the Object Relational Designer, which provides a graphic user interface for implementing many of the features of LINQ to SQL.

4.3.4 Overview of LINQ to Entities

LINQ to Entities belongs to LINQ to ADO.NET, and it is a subcomponent of LINQ to ADO.NET. Through the Entity Data Model we discussed in Section 3.4.8.1, ADO.NET 3.5 exposes entities as objects in the .NET environment. This makes the object layer an ideal target for LINQ support. Therefore, LINQ to ADO.NET includes LINQ to Entities. LINQ to Entities enables developers to write queries against the database from the same language used to build the business logic. Figure 4.22 shows the relationship between LINQ to Entities and the Entity Framework, ADO.NET 2.0, and the data store.

It can be found that the Entities and Entity Data Model (EDM) released by ADO.NET 3.5 locates at the top of this LINQ to Entities, and they are converted to the logical model by the Mapping Provider and interfaced to the data components such as Data Providers defined in ADO.NET 2.0. The bottom components used for this model are still “old” components that work for the ADO.NET 2.0.

Most applications are currently written on the relational databases and they are compatible with ADO.NET 2.0. At some point, these applications will have to interact with the data represented in a relational form. Database schemas are not always ideal for building applications, and the conceptual models of applications differ from the logical models of databases. The EDM released with ADO.NET 3.5 is a conceptual data model

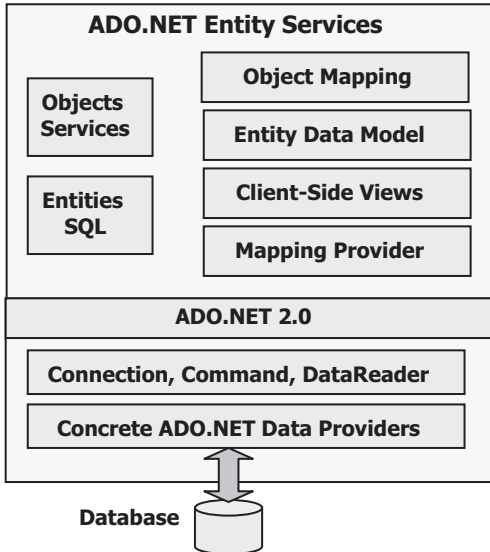


Figure 4.22 Relationship between LINQ to Entities, the Entity Framework, and ADO.NET 2.0.

that can be used to model the data of a particular domain so that applications can interact with data as entities or objects.

4.3.5 Overview of LINQ to XML

LINQ to XML is a LINQ-enabled, in-memory XML programming interface that enables you to work with XML from within the .NET Framework programming languages. LINQ to XML provides an in-memory XML programming interface that leverages the .NET LINQ Framework. LINQ to XML uses the latest .NET Framework language capabilities and is comparable to an updated, redesigned Document Object Model (DOM) XML programming interface. This interface was previously known as Xling in older prereleases of LINQ.

The LINQ family of technologies provides a consistent query experience for objects (LINQ), relational databases (LINQ to SQL), and XML (LINQ to XML). At this point, we have finished an overview of the LINQ family. Now let's go a little deeper for those topics to get a more detailed discussion of each of them.

4.4 LINQ TO OBJECTS

As we mentioned in the previous section, LINQ to Objects is used to query any sequences or collections that are either explicitly or implicitly compatible with `IEnumerable` sequences or `IEnumerable<T>` collections. Since any `IEnumerable` collection contains a sequence of objects with a data type that is compatible with `IEnumerable<T>`, therefore there is no need to use any LINQ API such as LINQ to SQL to convert or map this collection from an object model to a relational model, and the LINQ to Objects can be directly implemented to those collections or sequences to perform the queries.

Regularly LINQ to Objects is mainly used to query arrays and in-memory data collections. In fact, it can be used to query for any enumerable collections such as `List<T>`, `Array`, or `Dictionary<TKey, TValue>`. All of these queries are performed by executing SQO methods defined in the `IEnumerable` class. The difference between the `IEnumerable` and `IEnumerable<T>` interfaces is that the former is used for nongeneric collections and the latter is used for generic collections. In Sections 4.1.3 and 4.1.4, we provided a very detailed discussion about the Standard Query Operators. Now let's give a little more detailed discussion about the LINQ to Objects using those Standard Query Operators. We divide this discussion into the following four parts:

1. LINQ and `ArrayList`
2. LINQ and Strings
3. LINQ and File Directories
4. LINQ and Reflection

Let's start with the first part, LINQ and `ArrayList`.

4.4.1 LINQ and `ArrayList`

When using LINQ to query nongeneric `IEnumerable` collections such as `ArrayList`, you must explicitly declare the type of range variable to reflect the specific type of the objects in the collection. For example, if you have an `ArrayList` of `Student` objects, your `from` clause in a query should look like this:

```
var query = from Student s in arrList
```

By specifying the type of the range variable `s` with `Student`, you are casting each item in the `ArrayList arrList` to a `Student`.

The use of an explicitly typed range variable in a query expression is equivalent to calling the `Cast<TResult>` method. `Cast<TResult>` throws an exception if the specified cast cannot be performed. `Cast<TResult>` and `OfType<TResult>` are the two SQO methods we discussed in Section 4.1.3 and these two methods operate on nongeneric `IEnumerable` types.

Let's illustrate this query with a C# example project named `NonGenericLINQ.cs`. Create a new C# Console project and name it `NonGenericLINQ`, and enter the codes shown in Figure 4.23 into the code window of this new project.

Let's take a closer look at this piece of code to see how it works.

- A. The `System.Collections` namespace is first added into this project since all nongeneric collections are defined in this namespace. In order to use any nongeneric collection such as `ArrayList`, you must declare this namespace in this project before it can be used.
- B. A new `Student` class with two members is created, and this class is used as a protocol for those objects to be created and added into the `ArrayList` nongeneric collection later.
- C. A new instance of the `ArrayList` class `arrList` is created and initialized by adding four new `Student` objects.
- D. A LINQ query is created with the `student` as the range variable whose type is defined as `Student` by a `Cast` operator method. The filtering condition is that all student objects should be selected as long as their first `Scores`'s value is greater than 95.

```

using System;
A using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NonGenericLINQ
{
B     public class Student
    {
        public string StudentName { get; set; }
        public int[] Scores { get; set; }
    }
    class Program
    {
C         static void Main(string[] args)
        {
            ArrayList arrList = new ArrayList();
            arrList.Add(new Student { StudentName = "Svetlana Omelchenko", Scores = new int[] { 98, 92, 81, 60 } });
            arrList.Add(new Student { StudentName = "Claire O'Donnell", Scores = new int[] { 75, 84, 91, 39 } });
            arrList.Add(new Student { StudentName = "Sven Mortensen", Scores = new int[] { 88, 94, 65, 91 } });
            arrList.Add(new Student { StudentName = "Cesar Garcia", Scores = new int[] { 97, 89, 85, 82 } });

D             var query = from Student student in arrList
                            where student.Scores[0] > 95
                            select student;

E             foreach (Student s in query)
                Console.WriteLine(s.StudentName + ": " + s.Scores[0]);

F             // Keep the console window open in debug mode.
            Console.WriteLine("Press any key to exit... ");
            Console.ReadKey();
        }
    }
}

```

Figure 4.23 Coding for the example project NonGenericLINQ.

- E.** The *foreach* loop is used to pick up all query results one by one and assign it to the iteration variable *s*. A `Console.WriteLine()` method is executed to display each received data item, including the student's name and scores. This query process will continue until all data items have been retrieved from the `ArrayList`.
- F.** Two code lines here allow users to run this project in the Debugging mode. As you know, the Console window cannot be kept in the opening status if you run the project in the Debugging mode without these two lines of codes.

Since now you can Build and Run the project, the running result should be:

```

Svetlana Omelchenko: 98
Cesar Garcia: 97

```

A complete C# Console project named NonGenericLINQ can be found from the folder DBProjects\Chapter 4, which is located at the accompanying ftp site (see Chapter 1).

4.4.2 LINQ and Strings

LINQ can be used to query and transform strings and collections of strings. It can be especially useful with semistructured data in text files. LINQ queries can be combined

with traditional string functions and regular expressions. For example, you can use the `Split` or `Split()` method to create an array of strings that you can then query or modify by using LINQ. You can use the `IsMatch()` method in the **where** clause of a LINQ query. And you can use LINQ to query or modify the `MatchCollection` results returned by a regular expression.

You can query, analyze, and modify text blocks by splitting them into a queryable array of smaller strings by using the `Split()` method. You can split the source text into words, sentences, paragraphs, pages, or any other criteria and then perform additional splits if they are required in your query. Many different types of text files consist of a series of lines, often with similar formatting, such as tab- or comma-delimited files or fixed-length lines. After you read such a text file into memory, you can use LINQ to query and/or modify the lines. LINQ queries also simplify the task of combining data from multiple sources.

Two example projects are provided in this part to illustrate (1) how to query a string to determine the number of numeric digits it contains, and (2) how to sort lines of structured text, such as comma-separated values, by any field in the line.

4.4.2.1 Query a String to Determine Number of Numeric Digits

Because the `String` class implements the generic `IEnumerable<T>` interface, any string can be queried as a sequence of characters. However, this is not a common use of LINQ. For complex pattern-matching operations, use the `Regex` class.

The following example queries a string to determine the number of numeric digits it contains. Note that the query is “reused” after it is executed the first time. This is possible because the query itself does not store any actual results.

Create a new C# Console project and name it as `QueryStringLINQ`, and enter the codes shown in Figure 4.24 into the code window of this new project.

Let’s take a closer look at this piece of code to see how it works.

- A. The namespace `System.IO` is added into the namespace declaration section of this project since we need to use some components defined in that namespace.
- B. A string object or a generic collection `aString` is created, and this will work as a data source to be queried by LINQ to Objects.
- C. The LINQ to Objects query is created and initialized with three clauses. The method `IsDigit()` is used as the filtering condition for the `where` clause and `ch` is the range variable. All digital elements in this string collection will be filtered, selected, and returned. A `Cast()` operator is used for the returned query collection with an `IEnumerable<T>` interface, and `T` is replaced by the real data type `char` here.
- D. The query is executed by using a `foreach` loop and `c` is an iteration variable. The queried digits are displayed by using the `Console.WriteLine()` method.
- E. The `Count()` method is executed to query the number of digits existing in the queried string. This query is “reused” because the query itself does not store any actual results.
- F. Another query, or the second query, is created and initialized. The purpose of this query is to retrieve all letters before the first dash line in the string collection.
- G. The second query is executed and the result is displayed using the `Console.WriteLine()` method.


```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
A using System.IO;
namespace QueryStringLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
B         string aString = "ABCDE99F-J74-12-89A";
C         IEnumerable<char> stringQuery = from ch in aString
            where Char.IsDigit(ch)
            select ch;

D         // Execute the query
            foreach (char c in stringQuery)
                Console.Write(c + " ");

E         // Call the Count method on the existing query.
            int count = stringQuery.Count();
            Console.WriteLine("Count = {0}", count);

F         // Select all characters before the first '-'
            IEnumerable<char> stringQuery2 = aString.TakeWhile(c => c != '-');

G         // Execute the second query
            foreach (char c in stringQuery2)
                Console.Write(c);

H         Console.WriteLine(System.Environment.NewLine + "Press any key to exit");
            Console.ReadKey();
        }
    }
}

```

Figure 4.24 Coding for the example project QueryStringLINQ.

H. The purpose of these two coding lines is to allow users to run this project in a Debugging mode.

Now you can run the project in Debugging mode by clicking Debug|Start Debugging menu item, and the running result of this project is:

```

9 9 7 4 1 2 8 9 Count = 8
ABCDE99F

```

A complete C# Console project named QueryStringLINQ can be found from the folder DBProjects\Chapter 4 that is located at the accompanying ftp site (see Chapter 1).

Our first example is successful, and let's have a look at the second example.

4.4.2.2 Sort Lines of Structured Text by Any Field in Line

This example shows readers how to sort lines of structured text, such as comma-separated values, by any field in the line. The field may be dynamically specified at runtime. Assume that the fields in a sample text file scores.csv represent a student's ID number, followed by a series of four test scores.

111,	97,	92,	81,	60
112,	75,	84,	91,	39
113,	88,	94,	65,	91
114,	97,	89,	85,	82
115,	35,	72,	91,	70
116,	99,	86,	90,	94
117,	93,	92,	80,	87
118,	92,	90,	83,	78
119,	68,	79,	88,	92
120,	99,	82,	81,	79
121,	96,	85,	91,	60
122,	94,	92,	91,	91

Figure 4.25 Content of sample text file scores.csv.

First let's create a new C# Console project named SortLinesLINQ. Then we need to create a sample text file scores.csv. Open the NotePad editor and enter the codes shown in Figure 4.25 into this opened text editor.

This file represents spreadsheet data. Column 1 is the student's ID, and columns 2 through 5 are test scores.

Click the File/Save As menu item from the NotePad editor to open the Save As dialog box. Browse the folder in which our new C# project SortLinesLINQ is located. Enter "scores.csv" into the File name box and click the Save button to save this sample file. The point to be noticed is that the file name scores.csv must be enclosed by a pair of double quotation marks when you save this file in the extension .csv. Otherwise the file will be saved with a text extension.

Close the NotePad editor and now let's develop the codes for our new project. Open the code window of our new C# project SortLinesLINQ and enter the codes shown in Figure 4.26 into this code window.

Let's take a closer look at this piece of code to see how it works.

- A.** The namespace System.IO is added into the namespace declaration section of this project since we need to use some components defined in that namespace.
- B.** A string collection *scores* is created as the data source for this project, and this collection is a generic collection that is compatible with the IEnumerable<T> data type. The method ReadAllLines() is executed to open and read the sample file scores.csv we created at the beginning of this section, and assign this file to the *scores* string collection.
- C.** A local integer variable *sortField* is initialized to 1, which means that we want to use the first column in this string collection, student ID, as the filtering criteria. You can change this criteria by selecting any other column if you like.
- D.** The query is built and executed by calling a method RunQuery() with two arguments: the data source *scores* and the filtering criteria *sortField*. The queried results are displayed by executing the method Console.WriteLine().
- E.** The purpose of these two coding lines is to allow users to run this project in a Debugging mode.
- F.** The body of the method RunQuery() starts from here. One point to be noticed is that the keyword *static* must be prefixed in front of this method to indicate that this method is a static method. This is very important because this method will be called from inside the main() method, which is a static method. Otherwise a compiling error would be encountered if you missed this keyword.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
A using System.IO;
namespace SortLinesLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
            B // Create an IEnumerable data source
            string[] scores = System.IO.File.ReadAllLines(@"..\..\scores.csv");
            C // Change this to any value from 0 to 4.
            int sortField = 1;
            Console.WriteLine("Sorted highest to lowest by field [{0}]:", sortField);
            // Demonstrates how to return query from a method.
            // The query is executed here.
            D foreach (string str in RunQuery(scores, sortField))
            {
                Console.WriteLine(str);
            }
            // Keep the console window open in debug mode.
            E Console.WriteLine("Press any key to exit");
            Console.ReadKey();
        }
        // Returns the query variable, not query results!
        F static IEnumerable<string> RunQuery(IEnumerable<string> source, int num)
        {
            // Split the string and sort on field[num]
            G var scoreQuery = from line in source
                let fields = line.Split(',')
                orderby fields[num] descending
                select line;
            H return scoreQuery;
        }
    }
}

```

Figure 4.26 Coding for the example project SortLinesLINQLINQ.

G. The query is built with four clauses. The `Split()` method is used in the *let* clause to allow the string to be split into different pieces at each comma. The queried result is distributed in a descending order by using the `orderby` operator.

H. The queried result is returned to the calling method.

Now run the project by clicking the `Debug|Start Debugging` menu item, and the running result of this project is shown in Figure 4.27.

A complete C# Console project named `SortLinesLINQ` can be found from the folder `DBProjects\Chapter 4` that is located at the accompanying ftp site (see Chapter 1). Next let's take care of another LINQ to Objects query, LINQ and File Directories.

4.4.3 LINQ and File Directories

Many file system operations are essentially queries and are therefore well-suited to the LINQ approach. Note that the queries for those file systems are read-only. They are not

```
Sorted highest to lowest by field [1]:
116, 99, 86, 90, 94
120, 99, 82, 81, 79
111, 97, 92, 81, 60
114, 97, 89, 85, 82
121, 96, 85, 91, 60
122, 94, 92, 91, 91
117, 93, 92, 80, 87
118, 92, 90, 83, 78
113, 88, 94, 65, 91
112, 75, 84, 91, 39
119, 68, 79, 88, 92
115, 35, 72, 91, 70
```

Figure 4.27 Running result of the project SortLinesLINQ.

used to change the contents of the original files or folders. This follows the rule that queries should not cause any side effects. In general, any code (including queries that perform create/update/delete operators) that modifies source data should be kept separate from the code that just queries the data.

Different file operations or queries exist for the file systems. The most typical operations include

1. Query for Files with a Specified Attribute or Name
2. Group Files by Extension (LINQ)
3. Query for the Total Number of Bytes in a Set of Folders (LINQ)
4. Query for the Largest File or Files in a Directory Tree (LINQ)
5. Query for Duplicate Files in a Directory Tree (LINQ)
6. Query the Contents of Files in a Folder (LINQ)

There is some complexity involved in creating a data source that accurately represents the contents of the file system and handles exceptions gracefully. The examples in this section create a snapshot collection of `FileInfo` objects that represents all the files under a specified root folder and all its subfolders. The actual state of each `FileInfo` may change in the time between when you begin and end executing a query. For example, you can create a list of `FileInfo` objects to use as a data source. If you try to access the **Length** property in a query, the `FileInfo` object will try to access the file system to update the value of **Length**. If the file no longer exists, you will get a `FileNotFoundException` in your query, even though you are not querying the file system directly. Some queries in this section use a separate method that consumes these particular exceptions in certain cases. Another option is to keep your data source updated dynamically by using the `FileSystemWatcher`.

Because of limitations of space, here we only discuss one file operation or query, which is to open, inspect, and query the contents of files in a selected folder (the sixth operation).

4.4.3.1 Query Contents of Files in a Folder

This example shows how to query all the files in a specified directory tree, open each file, and inspect its contents. This type of technique could be used to create indexes or reverse

indexes of the contents of a directory tree. A simple string search is performed in this example. However, more complex types of pattern matching can be performed with a regular expression.

Create a new C# Console project named QueryContentsLINQ, and then open the code window of this new project and enter the codes shown in Figure 4.28 into the code window of this project.

Let's take a closer look at this piece of code to see how it works.

- A.** A string object `startFolder` is created and the value of this object is the default path of the Visual Studio.NET 2008, in which all files of the Visual Studio.NET 2008 are installed. You can modify this path if you installed your Visual Studio.NET 2008 at a different folder in your computer.
- B.** An `IEnumerable<T>` interface is used to define the data type of the queried files `fileList`. The real data type applied here is `System.IO.FileInfo`, which is used to replace the nominal type `T`. The method `GetFiles()` is executed to open and access the queried files with the file path as the argument of this method.
- C.** The query criteria “Visual Studio”, which is a keyword to be searched by this query, is assigned to a string object `searchTerm` that will be used in the following query process.
- D.** The LINQ query is created and initialized with four clauses, *from*, *let*, *where*, and *select*. The range variable *file* is selected from the opened files `fileList`. The method `GetFileText()` will be executed to read back the contents of the matched files using the *let* clause. Two *where* clauses are used here to filter the matched files with both an extension `.htm` and a keyword “Visual Studio” in the file name.
- E.** The `Console.WriteLine()` method is executed to indicate that the following matched files contain the searched keyword “Visual Studio” in their file names.
- F.** The LINQ query is executed to pick up all files that have a file name that contains the keyword “Visual Studio”, and all searched files are displayed by using the method `Console.WriteLine()`.
- G.** The purpose of these two coding lines is to allow users to run this project in a Debugging mode.
- H.** The body of the method `GetFileText()` starts from here. The point is that this method must be defined as a static method prefixed with the keyword *static* in front of this method since it will be called from the `main()` method, which is a static method, too.
- I.** The string object `fileContents` is initialized with an empty string object.
- J.** The system method `Exists()` is executed to find all files whose names contain the keyword “Visual Studio”. All of matched files will be opened and the contents will be read back by the method `ReadAllText()` and assigned to the string object `fileContents`.
- K.** The read-out `fileContents` object is returned to the calling method.
- L.** The body of the method `GetFiles()` starts from here with the path as the argument of this method. The point is that this method must be defined as a static method and the returned data type is an `IEnumerable<T>` type.
- M.** An exception will be thrown out if the desired path did not exist in the current computer.
- N.** A new nongeneric collection `List<T>` is created with a `Cast` to convert it to the `IEnumerable<T>` type.
- O.** The system method `GetFiles()` is executed to find the names of all files that are under the current path and assign them to the string object array `fileNames`.

```

QueryContentsLINQ.Program ▼ Main() ▼
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace QueryContentsLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
            // Modify this path as necessary.
A string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\";
            // Take a snapshot of the file system.
B IEnumerable<System.IO.FileInfo> fileList = GetFiles(startFolder);
            // Search the contents of each file. The queryMatchingFiles is an IEnumerable<string>.
C string searchTerm = @"Visual Studio";
            // Search the contents of each file. The queryMatchingFiles is an IEnumerable<string>.
D var queryMatchingFiles = from file in fileList
                            where file.Extension == ".htm"
                            let fileText = GetFileText(file.FullName)
                            where fileText.Contains(searchTerm)
                            select file.FullName;

            // Execute the query.
E Console.WriteLine("The term \"{0}\" was found in:", searchTerm);
F foreach (string filename in queryMatchingFiles)
            {
                Console.WriteLine(filename);
            }
            // Keep the console window open in debug mode.
G Console.WriteLine("Press any key to exit ...");
            Console.ReadKey();
        }
        // Read the contents of the file.
H static string GetFileText(string name)
        {
            string fileContents = String.Empty;
            // If the file has been deleted since we took the snapshot, ignore it and return the empty string.
J if (System.IO.File.Exists(name))
                fileContents = System.IO.File.ReadAllText(name);
K return fileContents;
        }
        // This method assumes that the application has discovery permissions for all folders under the specified path.
L static IEnumerable<System.IO.FileInfo> GetFiles(string path)
        {
            if (!System.IO.Directory.Exists(path))
                throw new System.IO.DirectoryNotFoundException();
            string[] fileNames = null;
N List<System.IO.FileInfo> files = new List<System.IO.FileInfo>();
O fileNames = System.IO.Directory.GetFiles(path, "*", System.IO.SearchOption.AllDirectories);
P foreach (string name in fileNames)
            {
                files.Add(new System.IO.FileInfo(name));
            }
Q return files;
        }
    }
}

```

Figure 4.28 Coding for the example project QueryContentsLINQ.

- P.** The foreach loop is executed to add all searched file names into the nongeneric collection `List<T>` object files.
- Q.** All of those files are returned to the calling method.

Now you can build and run the project by clicking the Debug|Start Debugging menu item. All files that have the extension `.htm`, and under the path `C:\program files\Microsoft Visual Studio 9.0\` and whose name contains the keyword “Visual Studio” are found and displayed as this project runs.

Press any key on the keyboard to exit this project.

A complete C# Console project named `QueryContentsLINQ` can be found in the folder `DBProjects\Chapter 4` located at the accompanying ftp site. Next let’s have a discussion about another query related to LINQ to Objects, the LINQ and Reflection.

4.4.4 LINQ and Reflection

The .NET Framework 3.5 class library reflection APIs can be used to examine the metadata in a .NET assembly and create collections of types, type members, parameters, and so on that are in that assembly. Because these collections support the generic `IEnumerable` interface, they can be queried by using LINQ to Objects query.

To make it simple and easy, in this section we use one example project to illustrate how LINQ can be used with reflection to retrieve specific metadata about methods that match a specified search criterion. In this case, the query will find the names of all the methods in the assembly that return enumerable types such as arrays.

Create a new C# console project and name it `QueryReflectionLINQ`. Open the code window of this new project and enter the codes shown in Figure 4.29 into this window.

Let’s take a closer look at this piece of code to see how it works.

- A.** The namespace `System.Reflection` is added into the namespace declaration part of this project since we need to use some components defined in this namespace in this coding.
- B.** An `Assembly` object is created with the `Load()` method and is executed to load and assign this new `Assembly` to the instance assembly.
- C.** The LINQ query is created and initialized with three clauses. The `GetTypes()` method is used to obtain the data type of all queried methods. The first where clause is used to filter methods in the `Public` type. The second from clause is used to get the desired methods based on the data type `Public`. The second where clause is used to filter all methods with three criteria: (1) the returning type of the method is array, (2) those methods should have a valid interface, and (3) the returning type of those methods should not be `System..string`. Also the queried methods’ names are converted to string.
- D.** Two foreach loops are utilized here. The first one is used to retrieve and display the data type of the queried methods, and the second one is used to retrieve and display the names of the queried methods.
- E.** The purpose of these two coding lines is to allow users to run this project in a Debugging mode.

Now you can build and run the project by clicking the Debug|Start Debugging menu item. The running results are displayed in the console window. A complete C# Console project named `QueryReflectionLINQ` can be found in the folder `DBProjects\Chapter 4` located at the accompanying ftp site (see Chapter 1).

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
A using System.Reflection;
namespace QueryReflectionLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
B         Assembly assembly = Assembly.Load("System.Core, Version=3.5.0.0, Culture=neutral, " +
C             "PublicKeyToken= b77a5c561934e089");
            var pubTypesQuery = from type in assembly.GetTypes()
                                where type.IsPublic
                                from method in type.GetMethods()
                                where method.ReturnType.IsArray == true ||
                                     (method.ReturnType.GetInterface(typeof
                                     (System.Collections.Generic.IEnumerable<>).FullName) != null
                                     && method.ReturnType.FullName != "System.String")
                                group method.ToString() by type.ToString();
D
            foreach (var groupOfMethods in pubTypesQuery)
            {
                Console.WriteLine("Type: {0}", groupOfMethods.Key);
                foreach (var method in groupOfMethods)
                {
E                     Console.WriteLine(" {0}", method);
                }
            }
            Console.WriteLine("Press any key to exit ... ");
            Console.ReadKey();
        }
    }
}

```

Figure 4.29 Coding for the example project QueryReflectionLINQ.

4.5 LINQ TO DATASET

As we discussed in the previous section, LINQ to DataSet is a subcomponent of LINQ to ADO.NET. The DataSet, of which we provided a very detailed discussion in Chapter 3, is one of the most widely used components in ADO.NET, and it is a key element of the disconnected programming model upon which ADO.NET is built. Despite this prominence, however, the DataSet has limited query capabilities.

LINQ to DataSet enables you to build richer query capabilities into DataSet by using the same query functionality that is available for many other data sources. Because the LINQ to DataSet is built on the existing ADO.NET 2.0 architecture, the codes developed by using ADO.NET 2.0 will continue to function in a LINQ to DataSet application without modifications. This is a very valuable advantage since any new component has its own architecture and tools with a definite learning curve needed in order to understand it.

Among all LINQ to DataSet query operations, the following three are most often implemented in most popular applications:

1. Perform operations to DataSet objects.
2. Perform operations to DataRow objects using the extension methods.
3. Perform operations to DataTable objects.

First let's get a deeper understanding of the LINQ to DataSet or the operations to the DataSet objects.

4.5.1 Operations to DataSet Objects

Data sources that implement the `IEnumerable<T>` generic interface can be queried through LINQ using the SQO methods. Using `AsEnumerable` SQO to query a `DataTable` returns an object that implements the generic `IEnumerable<T>` interface, which serves as the data source for LINQ to DataSet queries.

In the query, you specify exactly the information that you want to retrieve from the data source. A query can also specify how that information should be sorted, grouped, and shaped before it is returned. In LINQ, a query is stored in a variable. If the query is designed to return a sequence of values, the query variable itself must be an enumerable type. This query variable takes no action and returns no data; it only stores the query information. After you create a query you must execute that query to retrieve any data.

In a query that returns a sequence of values, the query variable itself never holds the query results and only stores the query commands. Execution of the query is deferred until the query variable is iterated in a **foreach** loop. This is called deferred execution; that is, query execution occurs some time after the query is constructed. This means that you can execute a query as often as you want. This is useful when, for example, you have a database that is being updated by other applications. In your application, you can create a query to retrieve the latest information and repeatedly execute the query, returning the updated information every time.

In contrast to deferred queries, which return a sequence of values, queries that return a singleton value are executed immediately. Some examples of singleton queries are `Count`, `Max`, `Average`, and `First`. These execute immediately because the query results are required to calculate the singleton result. For example, in order to find the average of the query results the query must be executed so that the averaging function has input data with which work. You can also use the `ToList<TSource>` or `ToArray<TSource>` methods on a query to force immediate execution of a query that does not produce a singleton value. These techniques to force immediate execution can be useful when you want to cache the results of a query.

Basically, to perform a LINQ to DataSet query, three steps are needed:

1. Create a new `DataSet` instance.
2. Populate the `DataSet` instance using the `Fill()` method.
3. Query the `DataSet` instance using LINQ to DataSet.

After a `DataSet` object has been populated with data, you can begin querying it. Formulating queries with LINQ to DataSet is similar to using LINQ against other LINQ-enabled data sources. Remember, however, that when you use LINQ queries over a `DataSet` object you are querying an enumeration of `DataRow` objects, instead of an enumeration of a custom type. This means that you can use any of the members of the `DataRow` class in your LINQ queries. This lets you create rich and complex queries.

As with other implementations of LINQ, you can create LINQ to DataSet queries in two different forms: query expression syntax and method-based query syntax. Basically, the query expression syntax will be finally converted to the method-based query syntax

as the compiling time if the query is written as the query expression, and the query will be executed by calling the SQO methods as the project runs.

4.5.1.1 Query Expression Syntax

A query expression is a query expressed in query syntax. A query expression is a first-class language construct. It is just like any other expression and can be used in any context in which a C# expression is valid. A query expression consists of a set of clauses written in a declarative syntax similar to SQL or XQuery. Each clause in turn contains one or more C# expressions, and these expressions may themselves be either a query expression or contain a query expression.

A query expression must begin with a **from** clause and must end with a **select** or **group** clause. Between the first **from** clause and the last **select** or **group** clause, it can contain one or more of these optional clauses: **where**, **orderby**, **join**, **let**, and even additional **from** clauses. You can also use the **into** keyword to enable the result of a **join** or **group** clause to serve as the source for additional query clauses in the same query expression.

In all LINQ queries (including LINQ to DataSet), all of clauses will be converted to the associated SQO methods, such as From, Where, OrderBy, Join, Let, and Select, as the queries are compiled. Refer to Table 4.1 to get the most often used Standard Query Operators and their definitions.

In LINQ, a query variable is always strongly typed, and it can be any variable that stores a query instead of the results of a query. More specifically, a query variable is always an enumerable type that will produce a sequence of elements when it is iterated over in a **foreach** loop or a direct call to its method `IEnumerator.MoveNext`.

The code example in Figure 4.30 shows a simple query expression with one data source, one filtering clause, one ordering clause, and no transformation of the source elements. The **select** clause ends the query.

An integer array is created here and this array works as a data source. The variable `scoreQuery` is a query variable, and it contains only the query command and does not

```

static void Main()
{
    // Data source.
    int[] scores = { 90, 71, 82, 93, 75, 82 };
    // Query Expression.
    IEnumerable<int> scoreQuery = from score in scores           //required
                                where score > 80              //optional
                                orderby score descending        //optional
                                select score;                  //must end with select or group

    // Execute the query to produce the results
    foreach (int testScore in scoreQuery)
    {
        Console.WriteLine(testScore);
    }
}

// Outputs: 90 82 93 82

```

Figure 4.30 Example codes for the query expression syntax.

```

static void Main()
{
    // Data source.
    int[] scores = { 90, 71, 82, 93, 75, 82 };
    // Query Expression.
    var scoreQuery = from score in scores           //required
                    where score > 80              //optional
                    orderby score descending      //optional
                    select score;                 //must end with select or group

    // Execute the query to produce the results
    foreach (var testScore in scoreQuery)
    {
        Console.WriteLine(testScore);
    }
}

// Outputs: 90 82 93 82

```

Figure 4.31 Example codes for the query expression in implicit typing of query variable.

contain any query result. This query is composed of four clauses: from, where, orderby, and select. Both the first and the last clause are required and the others are optional. The query is cast to a type of `IEnumerable<int>` by using an `IEnumerable<T>` interface. The `testScore` is an iteration variable that is scanned through the `foreach` loop to get and display each queried data when this query is executed. Basically, when the `foreach` statement executes, the query results are not returned through the query variable `scoreQuery`. Rather, they are returned through the iteration variable `testScore`.

An alternative way to write this query expression is to use the so-called implicit typing of query variables. The difference between the explicit and implicit typing of query variables is that in the former situation, the relationship between the query variable `scoreQuery` and the `select` clause is clearly indicated by the `IEnumerable<T>` interface, and this makes sure that the type of returned collection is `IEnumerable<T>`, which can be queried by LINQ. In the latter situation, we do not exactly know the data type of the query variable, and therefore an implicit type `var` is used to instruct the compiler to infer the type of a query variable (or any other local variable) at the compiling time. The example codes written in Figure 4.30 can be expressed in another format as shown in Figure 4.31 by using the implicit typing of query variable.

Here the implicit type `var` is used to replace the explicit type `IEnumerable<T>` for the query variable, and it can be converted to the `IEnumerable<int>` automatically as this piece of code is compiled.

4.5.1.2 Method-Based Query Syntax

Most queries used in the general LINQ queries are written as query expressions by using the declarative query syntax introduced in C# 3.0. However, the .NET Common Language Runtime (CLR) has no notion of query syntax in itself. Therefore, at compile time, query expressions are converted to something that the CLR can understand—method calls. These methods are SQO methods, and they have names equivalent to query clauses such as **Where**, **Select**, **GroupBy**, **Join**, **Max**, **Average**, and so on. You can call them directly by using method syntax instead of query syntax. In Sections 4.1.3 and 4.1.4, we provided

a very detailed discussion about the Standard Query Operator methods. Refer to that section to get more details for those methods and their implementations.

In general, we recommend query syntax because it is usually simpler and more readable; however, there is no semantic difference between method syntax and query syntax. In addition, some queries, such as those that retrieve the number of elements that match a specified condition, or that retrieve the element that has the maximum value in a source sequence, can only be expressed as method calls. The reference documentation for the Standard Query Operators in the `System.Linq` namespace generally uses method syntax. Therefore, even when getting started writing LINQ queries, it is useful to be familiar with how to use method syntax in queries and in query expressions themselves.

We have discussed the Standard Query Operator with quite few examples using the method syntax in Sections 4.1.3 and 4.1.4. Refer to those sections to get a clear picture of how to create and use method syntax to directly call SQO methods to perform LINQ queries. In this section, we just give an example to illustrate the different format using the query syntax and the method syntax for a given data source.

Create a new C# console project named `QueryMethodSyntax`. Open the code window of this new project and enter the codes shown in Figure 4.32 into this code window.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace QueryMethodSyntax
{
    class Program
    {
        static void Main(string[] args)
        {
            A int[] numbers = {5, 10, 8, 3, 6, 12};
            //Query syntax:
            B IEnumerable<int> querySyntax = from num in numbers
                                           where num % 2 == 0
                                           orderby num
                                           select num;

            //Method syntax:
            C IEnumerable<int> methodSyntax = numbers.Where(num => num % 2 == 0).OrderBy(n => n);
            //Execute the query in query syntax
            D foreach (int i in querySyntax)
              {
                Console.Write(i + " ");
              }
            Console.WriteLine(System.Environment.NewLine);
            //Execute the query in method syntax
            E foreach (int i in methodSyntax)
              {
                Console.Write(i + " ");
              }
            F // Keep the console open in debug mode.
              Console.WriteLine(System.Environment.NewLine);
              Console.WriteLine("Press any key to exit ... ");
              Console.ReadKey();
        }
    }
}

```

Figure 4.32 Coding for the example project `QueryMethodSyntax`.

Let's take a close look at this piece of code to see how it works.

- A. An integer array is created and it works as a data source for this project.
- B. The first query that uses a query syntax is created and initialized with four clauses. The query variable is named `querySyntax` with a type of `IEnumerable<int>`.
- C. The second query that uses a method syntax is created and initialized with the SGO methods `Where()` and `OrderBy()`.
- D. The first query is executed using a `foreach` loop, and the query result is displayed by using the `Console.WriteLine()` method.
- E. The second query is executed and the result is displayed, too.
- F. The purpose of these two coding lines is to allow users to run this project in a Debugging mode.

It can be found that the method syntax looks simpler in structure and easy to code compared with the query syntax from this piece of code. In fact, the first query with the query syntax will be converted to the second query with the method syntax as the project is compiled.

Now you can build and run the project. You can find that the running result is identical for both syntaxes. A complete C# Console project named `QueryMethodSyntax` can be found in the folder `DBProjects\Chapter 4` located at the accompanying ftp site (see Chapter 1).

Besides the general and special properties of query expressions discussed above, the following points are also important in understanding query expressions:

1. Query expressions can be used to query and to transform data from any LINQ-enabled data source. For example, a single query can retrieve data from a `DataSet` and produce an XML stream as output.
2. Query expressions are easy to master because they use many familiar C# language constructs.
3. The variables in a query expression are all strongly typed, although in many cases you do not have to provide the type explicitly because the compiler can infer it if an implicit type `var` is used.
4. A query is not executed until you iterate over the query variable in a **foreach** loop.
5. At compile time, query expressions are converted to SGO method calls according to the rules set forth in the C# specification. Any query that can be expressed by using query syntax can also be expressed by using method syntax. However, in most cases query syntax is more readable and concise.
6. As a rule when you write LINQ queries, we recommend that you use query syntax whenever possible and method syntax whenever necessary. There is no semantic or performance difference between the two different forms. Query expressions are often more readable than equivalent expressions written in method syntax.
7. Some query operations, such as `Count` or `Max`, have no equivalent query expression clause and must therefore be expressed as a method call. Method syntax can be combined with query syntax in various ways.
8. Query expressions can be compiled to expression trees or to delegates, depending on the type to which the query is applied. `IEnumerable<T>` queries are compiled to delegates. `IQueryable` and `IQueryable<T>` queries are compiled to expression trees.

Now let's start the LINQ to DataSet with the single table query.

4.5.1.3 Query the Single Table

Language-Integrated Query queries work on data sources that implement the `IEnumerable<T>` interface or the `IQueryable` interface. The `DataTable` class does not implement either interface, so you must call the `AsEnumerable` method if you want to use the `DataTable` as a source in the **From** clause of a LINQ query.

As we discussed in Section 4.5.1, to perform LINQ to DataSet query, the first step is to create an instance of the `DataSet` and fill it with the data from the database. To fill a `DataSet`, a `DataAdapter` can be used with the `Fill()` method attached to that `DataAdapter`. Each `DataAdapter` can only be used to fill a single `DataTable` in a `DataSet`.

In this section, we show readers an example to query a single `DataTable` using the LINQ to DataSet. Create a new C# console project and name it `DataSetSingleTableLINQ`. Open the code window of this new project and enter the codes shown in Figure 4.33.

Let's take a closer look at this piece of code to see how it works.

```

using System;
A using System.Data;
using System.Data.OleDb;
using System.Linq;

namespace DataSetSingleTableLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
B         string cmdString = "SELECT * FROM Faculty";
            OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
            OleDbConnection accConnection = new OleDbConnection();
            OleDbCommand accCommand = new OleDbCommand();
            DataSet ds = new DataSet();
C         string connString = "Provider=Microsoft.ACE.OLEDB.12.0;" + //modify this based on your appl.
                "Data Source=C:\\database\\Access\\CSE_DEPT.accdb;";
D         accConnection = new OleDbConnection(connString);
            accConnection.Open();
E         accCommand.Connection = accConnection;
            accCommand.CommandType = CommandType.Text;
            accCommand.CommandText = cmdString;
F         dataAdapter.SelectCommand = accCommand;
            dataAdapter.Fill(ds, "Faculty");
G         var facultyinfo = (from fi in ds.Tables["Faculty"].AsEnumerable()
                where fi.Field<string>("faculty_name").Equals("Ying Bai")
                select fi);
H         foreach (var fRow in facultyinfo)
            {
                Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}", fRow.Field<string>("title"), fRow.Field<string>("office"),
                    fRow.Field<string>("phone"), fRow.Field<string>("college"), fRow.Field<string>("email"));
            }
I         accConnection.Close();
        }
    }
}

```

Figure 4.33 Coding for the example project `DataSetSingleTableLINQ`.

- A. Two namespaces, `System.Data` and `System.Data.OleDb`, must be added into the namespace declaration section of this project since we need to use some OleDb data components such as `DataAdapter`, `Command`, and `Connection`.
- B. An SQL query string is created to query all columns from the Faculty data table in the `DataSet`. Also all OleDb data components are created in this part including a non-OleDb data component, `DataSet`.
- C. The connection string is declared since we need to use it to connect to our sample database `CSE_DEPT.accdb`, which is developed in Microsoft Access 2007. You need to modify this string based on the real location in which you save your database.
- D. The `Connection` object `accConnection` is initialized with the connection string and a connection is executed by calling the `Open()` method. Regularly a `try ... catch` block should be used for this connection operation to catch up any possible exception. Here we skip it since we try to make this connection coding simple.
- E. The `Command` object is initialized with `Connection`, `CommandType`, and `CommandText` properties.
- F. The initialized `Command` object is assigned to the `SelectCommand` property of the `DataAdapter` and the `DataSet` is filled with the `Fill()` method. The point is that only a single table, `Faculty`, is filled in this operation.
- G. A LINQ to DataSet query is created with three clauses, `from`, `where`, and `select`. The data type of the query variable `facultyinfo` is an implicit, and it can be inferred by the compiler as the project is compiled. The `Faculty` data table works as a data source for this LINQ to DataSet query, therefore the `AsEnumerable()` method must be used to convert it to an `IEnumerable<T>` type. The `where` clause is used to filter the desired information for the selected faculty member (`faculty_name`). All of these clauses will be converted to the associated SQO methods that will be executed to perform and complete this query.
- H. The `foreach` loop then enumerates the enumerable object returned by `select` and yields the query results. Because query is an `Enumerable` type, which implements `IEnumerable<T>`, the evaluation of the query is deferred until the query variable is iterated using the `foreach` loop. Deferred query evaluation allows queries to be kept as values that can be evaluated multiple times, each time yielding potentially different results.
- I. Finally the connection to our sample database is closed by calling the `Close()` method.

Now you can build and run this project by clicking `Debug|Start Without Debugging`. Related information for the selected faculty will be retrieved and displayed in the console window.

A complete C# Console project named `DataSetSingleTableLINQ` can be found in the folder `DBProjects\Chapter 4` located at the accompanying ftp site (see Chapter 1).

Next let's take a look at querying the cross tables using LINQ to DataSet.

4.5.1.4 Query the Cross Tables

A `DataSet` object must first be populated before you can query it with LINQ to DataSet. There are several different ways to populate the `DataSet`. From the example we discussed in the last section, we used the `DataAdapter` class with the `Fill()` method to do this population operation.

In addition to querying a single table, you can also perform cross-table queries in LINQ to DataSet. This is done by using a join clause. A join is the association of objects

in one data source with objects that share a common attribute in another data source, such as a `faculty_id` in the `LogIn` table and in the `Faculty` table. In object-oriented programming, relationships between objects are relatively easy to navigate because each object has a member that references another object. In external database tables, however, navigating relationships is not as straightforward. Database tables do not contain built-in relationships. In these cases, the `Join` operation can be used to match elements from each source. For example, given two tables that contain faculty information and course information, you could use a `join` operation to match course information and faculty for the same `faculty_id`.

The LINQ framework provides two `join` operators, `Join` and `GroupJoin`. These operators perform equi-joins: that is, joins that match two data sources only when their keys are equal. (By contrast, Transact-SQL supports `join` operators other than **equals**, such as the **less than** operator.)

In relational database terms, `Join` implements an inner join. An inner join is a type of `join` in which only those objects that have a match in the opposite data set are returned.

In this section, we use an example project to illustrate how to use `Join` operator to perform a multitable query using LINQ to `DataSet`. The functionality of this project is:

1. Populate a `DataSet` instance, and exactly populate two data tables, `Faculty` and `Course`, with two `DataAdapters`.
2. Using LINQ to `DataSet` `join` query to perform the cross-table query.

Now create a new C# console project and name it `DataSetCrossTableLINQ`. Open the code window of this new project and enter the code shown in Figure 4.34 into this window.

Let's have a closer look at this piece of code to see how it works.

- A. Two namespaces, `System.Data` and `System.Data.OleDb`, must be added into the namespace declaration section of this project since we need to use some `OleDb` data components such as `DataAdapter`, `Command`, and `Connection`.
- B. Two SQL query strings are created to query some columns from the `Faculty` and the `Course` data tables in the `DataSet`. Also all `OleDb` data components, including two sets of `Command` and `DataAdapter` objects, are created in this part including a non-`OleDb` data component, `DataSet`. Each set of components is used to fill an associated data table in the `DataSet`.
- C. The connection string is declared since we need to use it to connect to our sample database `CSE_DEPT.accdb`, which is developed in Microsoft Access 2007. You need to modify this string based on the real location in which you save your database.
- D. The `Connection` object `accConnection` is initialized with the connection string and a connection is executed by calling the `Open()` method. Regularly a `try ... catch` block should be used for this connection operation to catch up any possible exception. Here we skip it since we try to make this connection coding simple.
- E. The `facultyCommand` object is initialized with `Connection`, `CommandType`, and `CommandText` properties.
- F. The initialized `facultyCommand` object is assigned to the `SelectCommand` property of the `facultyAdapter`, and the `DataSet` is filled with the `Fill()` method. The point is that only a single table, `Faculty`, is filled in this operation.


```

DataSetCrossTableLINQ.Program ▼ Main() ▼
using System;
A using System.Data;
using System.Data.OleDb;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace DataSetCrossTableLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
B         string strFaculty = "SELECT faculty_id, faculty_name FROM Faculty";
            string strCourse = "SELECT course_id, faculty_id FROM Course";
            OleDbDataAdapter facultyAdapter = new OleDbDataAdapter();
            OleDbDataAdapter courseAdapter = new OleDbDataAdapter();
            OleDbConnection accConnection = new OleDbConnection();
            OleDbCommand facultyCommand = new OleDbCommand();
            OleDbCommand courseCommand = new OleDbCommand();
            DataSet ds = new DataSet();

C         string connString = "Provider=Microsoft.ACE.OLEDB.12.0;" +
            "Data Source=C:\\database\\Access\\CSE_DEPT.accdb;";
D         accConnection = new OleDbConnection(connString);
            accConnection.Open();

E         facultyCommand.Connection = accConnection;
            facultyCommand.CommandType = CommandType.Text;
            facultyCommand.CommandText = strFaculty;
F         facultyAdapter.SelectCommand = facultyCommand;
            facultyAdapter.Fill(ds, "Faculty");

G         courseCommand.Connection = accConnection;
            courseCommand.CommandType = CommandType.Text;
            courseCommand.CommandText = strCourse;
            courseAdapter.SelectCommand = courseCommand;
            courseAdapter.Fill(ds, "Course");

H         DataTable faculty = ds.Tables["Faculty"];
            DataTable course = ds.Tables["Course"];

I         var courseinfo = from ci in course.AsEnumerable()
            join fi in faculty.AsEnumerable()
            on ci.Field<string>("faculty_id") equals fi.Field<string>("faculty_id")
            where fi.Field<string>("faculty_name") == "Ying Bai"
            select new
            {
                course_id = ci.Field<string>("course_id")
            };
J         foreach (var cid in courseinfo)
            {
                Console.WriteLine(cid.course_id);
            }
K         accConnection.Close();
            facultyCommand.Dispose();
            courseCommand.Dispose();
            facultyAdapter.Dispose();
            courseAdapter.Dispose();
        }
    }
}

```

Figure 4.34 Coding for the example project DataSetCrossTableLINQ.

- G. The `courseCommand` object is initialized with `Connection`, `CommandType`, and `CommandText` properties. The initialized `courseCommand` object is assigned to the `SelectCommand` property of the `courseAdapter`, and the `DataSet` is filled with the `Fill()` method. The point is that only a single table, `Course`, is filled in this operation.
- H. Two `DataTable` objects, `faculty` and `course`, are created and mapped to the `DataSet`.
- I. A LINQ to `DataSet` query is created with a join clause. The data type of the query variable `courseinfo` is implicit, and it can be inferred by the compiler as the project is compiled. Two data tables, `Faculty` and `Course`, work as a joined data source for this LINQ to `DataSet` query, therefore the `AsEnumerable()` method must be used to convert them to an `IEnumerable<T>` type. Two identical fields, `faculty_id`, which is a primary key in the `Faculty` table and a foreign key in the `Course` tables, works as a joined criterion to link two tables together. The `where` clause is used to filter the desired course information for the selected faculty member (`faculty_name`). All of these clauses will be converted to the associated SQO methods that will be executed to perform and complete this query.
- J. The `foreach` loop then enumerates the enumerable object returned by `select` and yields the query results. Because query is an `Enumerable` type, which implements `IEnumerable<T>`, the evaluation of the query is deferred until the query variable is iterated using the `foreach` loop. Deferred query evaluation allows queries to be kept as values that can be evaluated multiple times, each time yielding potentially different results. All courses taught by the selected faculty are retrieved and displayed when this `foreach` loop is done.
- K. Finally the connection to our sample database is closed by calling the `Close()` method, and all data components used in this project are released.

Now you can build and run this project. One point to note is the connection string implemented in this project. You need to modify this string in step C if you installed your database file `CSE_DEPT.acddb` in a different folder.

Click the `Debug|Start Without Debugging` menu item to run the project, and you can find that all courses (`course_id`) taught by the selected faculty are retrieved and displayed in this console window.

A complete C# Console project named `DataSetCrossTableLINQ` can be found from the folder `DBProjects\Chapter 4` located at the accompanying ftp site (see Chapter 1).

Next let's take a look at querying typed `DataSet` with LINQ to `DataSet`.

4.5.1.5 Query Typed DataSet

If the schema of the `DataSet` is known at application design time, it is highly recommended that you use a typed `DataSet` when using LINQ to `DataSet`. A typed `DataSet` is a class that is derived from a `DataSet`. As such, it inherits all the methods, events, and properties of a `DataSet`. Additionally, a typed `DataSet` provides strongly typed methods, events, and properties. This means that you can access tables and columns by name, instead of using collection-based methods. This makes queries simpler and more readable.

LINQ to `DataSet` also supports querying over a typed `DataSet`. With a typed `DataSet`, you do not have to use the generic `Field` method or `SetField` method to access column data. Property names are available at compile time because the type information is included in the `DataSet`. LINQ to `DataSet` provides access to column values as the correct type, so that the type mismatch errors are caught when the code is compiled instead of at runtime.

Before you can begin querying a typed DataSet, you must generate the class by using the DataSet Designer in Visual Studio 2008.

In this section, we show readers how to use LINQ to DataSet to query a typed DataSet. In fact, it is very easy to perform this kind of query as long as a typed DataSet has been created. There are two ways to create a typed DataSet: using the Data Source Configuration Wizard or using the DataSet Designer. Both belong to the Design Tools and Wizards provided by Visual Studio.NET 2008.

We will use the second method, DataSet Designer, to create a typed DataSet. The database we will use is our sample database CSE_DEPT.accdb developed in Microsoft Access 2007.

Create a new C# console project and name it TypedDataSetLINQ. Now let's first create our typed DataSet. On the opened new project, right-click on our new project from the Solution Explorer window. Select the Add\New Item from the pop-up menu to open the Add New item dialog box, which is shown in Figure 4.35.

Click on the DataSet from the Template list and enter CSE_DEPTDataSet.xsd into the Name box as the name for this DataSet. Click on the Add button to add this DataSet into our project. Your finished Add New Item dialog box should match the one shown in Figure 4.35.

Next we need to select our data source for our new DataSet. Open the Server Explorer window and right-click on the first folder Data Connections if you have not connected any data source. Then click on the Add Connection item from the pop-up menu, and the Add Connection dialog box appears, which is shown in Figure 4.36a.

Make sure that the Data source box contains Microsoft Access Database File and click on the Browse button to locate the folder in which our sample database file CSE_DEPT.accdb is located. In this application, it is C:\database\Access. Browse to this folder and select our sample database file CSE_DEPT.accdb and click the Open button. Your finished Add Connection dialog box should match the one that is shown in Figure 4.36a.

You can click on the Test Connection button to test this connection. Click on the OK button to finish this process if the connection test is successful.

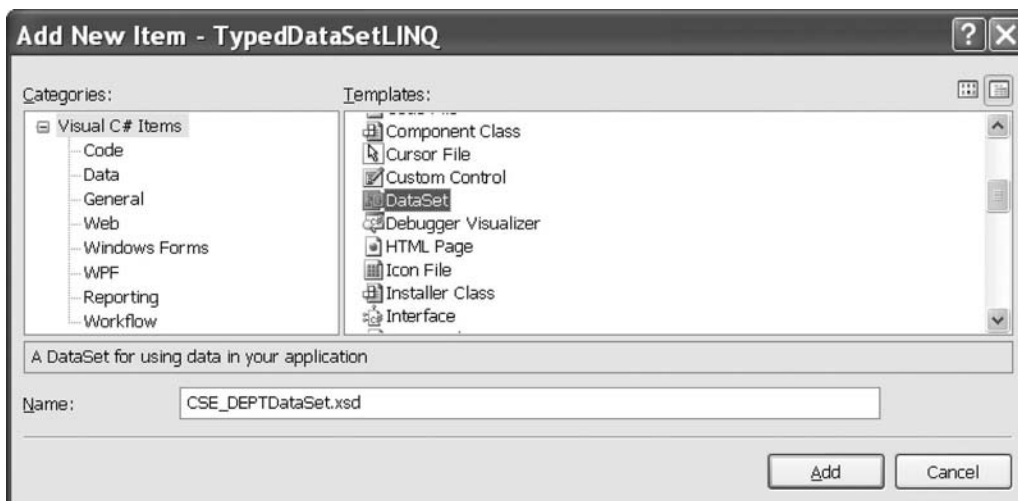


Figure 4.35 Opened Add New Item dialog box.

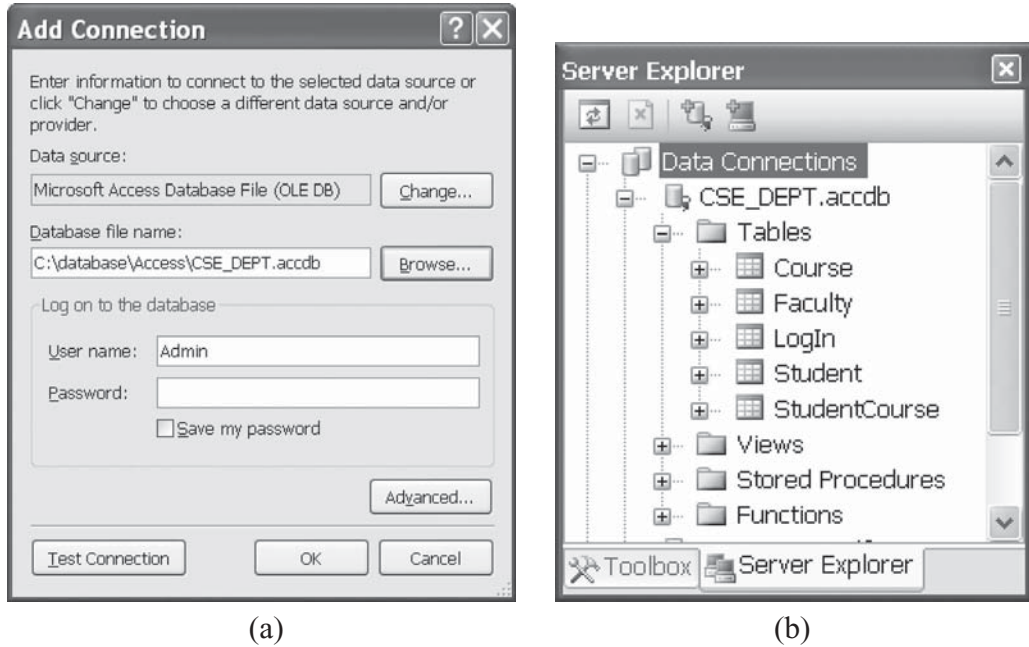


Figure 4.36 Add Connection dialog and the Server Explorer window.

Now you can find that a new data connection folder has been added to the Server Explorer window with our sample database CSE_DEPT.accdb. Expand the Tables folder under this data source, you can find all five tables, which is shown in Figure 4.36b.

Open the DataSet Designer by double-clicking on the item CSE_DEPTDataSet.xsd from the Solution Explorer window if it is not opened, drag the Faculty and Course tables from the Server Explorer window and place them to the DataSet Designer. You can drag/place all five tables if you like, but here we only need to drag two of them. Basically we only need to use the Faculty table in this project, and it does not matter if you drag more tables without using them.

Now we have finished creating our typed DataSet and the connection to our data source. Next we need to perform the coding to use LINQ to DataSet to perform the query to this typed DataSet.

Double-click on the item Program.cs from the Solution Explorer window to open the code window of this project. Enter the codes shown in Figure 4.37 into this window.

Let's take a closer look at this piece of code to see how it works.

- A.** Two namespaces, System.Data and System.Data.OleDb, must be added into the namespace declaration section of this project since we need to use some OleDb data components such as DataAdapter, Command, and Connection.
- B.** A new instance of the FacultyTableAdapter da is created since we need it to fill the DataSet later. All TableAdapters are defined in the CSE_DEPTDataSetTableAdapters namespace; therefore we must prefix it in front of the FacultyTableAdapter class.
- C.** A new DataSet instance ds is also created.

```

using System;
A using System.Data;
using System.Data.OleDb;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TypedDataSetLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
B         CSE_DEPTDataSetTableAdapters.FacultyTableAdapter da = new
C             CSE_DEPTDataSetTableAdapters.FacultyTableAdapter();
D         CSE_DEPTDataSet ds = new CSE_DEPTDataSet();
E         da.Fill(ds.Faculty);
F         var faculty = from fi in ds.Faculty
                        where fi.faculty_name == "Ying Bai"
                        select fi;
        foreach (var f in faculty)
        {
            Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}", f.title, f.office, f.phone, f.college, f.email);
        }
    }
}

```

Figure 4.37 Coding for the example project TypedDataSetLINQ.

- D.** The new instance of DataSet is populated with data using the Fill() method. Basically only the Faculty table is filled with data obtained from the Faculty table in our sample database CSE_DEPT.
- E.** The LINQ to DataSet query is created with three clauses. The data type of the query variable is an implicit data type **var**, and it can be inferred to the suitable type as the compiling time. Since we are using a typed DataSet, we can directly use the table name, Faculty, after the DataSet without worry about the Field<> setup with the real table name.
- F.** The foreach loop is executed to perform this query, and each queried column from the Faculty table is displayed using the Console.WriteLine() method. Compared with the same displaying operation in Figure 4.33, you can find that each column in the queried result can be accessed by using its name in this operation since a typed DataSet is used in this project.

Now you can build and run the project. Click on the Debug!Start Without Debugging item to run the project, and you can find that all information related to the selected faculty is retrieved and displayed in this console window. Our project is successful!

A complete C# Console project named TypedDataSetLINQ can be found in the folder DBProjects\Chapter 4 located at the accompanying ftp site (see Chapter 1).

4.5.2 Operations to DataRow Objects Using Extension Methods

The LINQ to DataSet functionality is exposed primarily through the extension methods in the DataRowExtensions and DataTableExtensions classes. In C#, you can call either

of these methods as an instance method on any object of type. When you use instance method syntax to call this method, omit the first parameter. The DataSet API has been extended with two new methods of the DataRow class, Field and SetField. You can use these to form LINQ expressions and method queries against DataTable objects. They are the recommended methods to use for accessing column values within LINQ expressions and method queries.

In this section, we show readers how to access and manipulate column values using the extension methods provided by the DataRow class, the Field(), and SetField() methods. These methods provide easier access to column values for developers, especially regarding null values. The DataSet uses Value to represent null values, whereas LINQ uses the nullable-type support introduced in the .NET Framework 2.0. Using the preexisting column accessor in DataRow requires you to cast the return object to the appropriate type. If a particular field in a DataRow can be null, you must explicitly check for a null value because returning Value and implicitly casting it to another type throws an InvalidCastException.

The Field() method allows users to obtain the value of a column from the DataRow object and handles the casting of DBNull.Value. Basically the Field() method has six different prototypes. The SetField() method, which has three prototypes, allows users to set a new value for a column from the DataRow object including handling a nullable data type whose value is null.

Now let's create a new C# console project to illustrate how to use the Field() method to retrieve some of columns' values from the DataRow object. The database we will use is still our sample Access 2007 database CSE_DEPT.accdb. Open Visual Studio.NET 2008 and create a new C# project and name it DataRowFieldLINQ. Open the code window of this new project and enter the code shown in Figure 4.38 into this window.

Let's take a closer look at this piece of code to see how it works.

- A.** Two namespaces, System.Data and System.Data.OleDb, must be added into the namespace declaration section of this project since we need to use some OleDb data components such as DataAdapter, Command, and Connection.
- B.** An SQL query string is created to query all columns from the Faculty data table in the DataSet. Also all OleDb data components are created in this part including a non-OleDb data component, DataSet.
- C.** The connection string is declared since we need to use it to connect to our sample database CSE_DEPT.accdb, which is developed in Microsoft Access 2007. You need to modify this string based on the real location in which you save your database.
- D.** The Connection object accConnection is initialized with the connection string and a connection is executed by calling the Open() method. Regularly a try ... catch block should be used for this connection operation to catch up any possible exception. Here we skip it since we try to make this connection coding simple.
- E.** The Command object is initialized with Connection, CommandType, and CommandText properties.
- F.** The initialized Command object is assigned to the SelectCommand property of the DataAdapter and the DataSet is filled with the Fill() method. The point is that only a single table, Faculty, is filled in this operation.
- G.** A single DataTable object, Faculty, is created and a DataRow object fRow is built based on the Faculty table with a casting <DataRow>.

```

DataRowFieldLINQ.Program
Main()
using System;
A using System.Data;
using System.Data.OleDb;
using System.Linq;
namespace DataRowFieldLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
B         string cmdString = "SELECT * FROM Faculty";
            OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
            OleDbConnection accConnection = new OleDbConnection();
            OleDbCommand accCommand = new OleDbCommand();
            DataSet ds = new DataSet();
C         string connString = "Provider=Microsoft.ACE.OLEDB.12.0;" +
                                "Data Source=C:\\database\\Access\\CSE_DEPT.accdb;";
D         accConnection = new OleDbConnection(connString);
            accConnection.Open();
E         accCommand.Connection = accConnection;
            accCommand.CommandType = CommandType.Text;
            accCommand.CommandText = cmdString;
F         dataAdapter.SelectCommand = accCommand;
            dataAdapter.Fill(ds, "Faculty");
G         DataTable dt = ds.Tables["Faculty"];
            IEnumerable<DataRow> fRow = dt.AsEnumerable();
H         string FacultyID = (from fi in fRow
                                where fi.Field<string>("faculty_name").Equals("Ying Bai")
                                select fi.Field<string>(dt.Columns[0], DataRowVersion.Current)).
                                Single<string>();
I         Console.WriteLine("\nThe Selected FacultyID is: {0}", FacultyID);
J         accConnection.Close();
        }
    }
}

```

Figure 4.38 Coding for the example project DataRowFieldLINQ.

- H.** The query is created and executed with the Field() method to pick up a single column, faculty_id, which is the first column in the Faculty table. The first prototype of the Field() method is used for this query. You can use any one of six prototypes if you like to replace this one. The SQO method Single() is also used in this query to indicate that we only need to retrieve a single column's value from this row.
- I.** The obtained faculty_id is displayed by using the Console.WriteLine() method.
- J.** The database connection is closed after this query is done.

Now you can build and run this project to test the functionality of querying a single column from a DataRow object. Click on the Debug!Start Without Debugging menu item to run the project. The desired faculty_id will be obtained and displayed in this console window.

A complete C# Console project named DataRowFieldLINQ can be found in the folder DBProjects\Chapter 4 located at the accompanying ftp site (see Chapter 1).

Before we can finish this section, we want to show users another example to illustrate how to modify a column's value by using the SetField() method via the DataRow object. Open Visual Studio.NET 2008 and create a new C# Console project and name it


```

using System;
A using System.Data;
using System.Data.OleDb;
using System.Linq;
namespace DataRowSetFieldLINQ
{
    class Program
    {
        static void Main(string[] args)
        {
            string cmdString = "SELECT * FROM Faculty";
            OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
            OleDbConnection accConnection = new OleDbConnection();
            OleDbCommand accCommand = new OleDbCommand();
            DataSet ds = new DataSet();

            string connString = "Provider=Microsoft.ACE.OLEDB.12.0;" +
                "Data Source=C:\\database\\Access\\CSE_DEPT.accdb;";
            accConnection = new OleDbConnection(connString);
            accConnection.Open();
            accCommand.Connection = accConnection;
            accCommand.CommandType = CommandType.Text;
            accCommand.CommandText = cmdString;

            dataAdapter.SelectCommand = accCommand;
            dataAdapter.Fill(ds, "Faculty");
            DataTable dt = ds.Tables["Faculty"];
            IEnumerable<DataRow> facultyRow = dt.AsEnumerable();
B
            DataRow frow = (from fi in facultyRow
                where fi.Field<string>("faculty_name").Equals("Ying Bai")
                select fi).Single<DataRow>();
C
            frow.AcceptChanges();
D
            frow.SetField("faculty_name", "Susan Bai");
E
            Console.WriteLine(" Original Faculty Name = {0};\n Current Faculty Name = {1}",
                frow.Field<string>("faculty_name", DataRowVersion.Original),
                frow.Field<string>("faculty_name", DataRowVersion.Current));
F
            accConnection.Close();
        }
    }
}

```

Figure 4.39 Coding for the example project DataRowSetFieldLINQ.

DataRowSetFieldLINQ. Open the code window of this new project and enter the codes shown in Figure 4.39 into this window.

The codes between steps **A** and **B** are identical with those we developed for our last project DataRwoFieldLINQ. Refer to that project to get more details for these codes and their functionalities.

Let's take a closer look at this piece of code to see how it works.

- A.** Two namespaces, System.Data and System.Data.OleDb, must be added into the namespace declaration section of this project since we need to use some OleDb data components such as DataAdapter, Command, and Connection.
- B.** A LINQ to DataSet query is created with the Field() method via DataRow object. This query should return a complete data row from the Faculty table.
- C.** The AcceptChanges() method is executed to allow the DataRow object to accept the current value of each DataColumn object in the Faculty table as the original version of

the value for that column. This method is very important and there would be no original version of the DataColumn object's values without this method.

- D. Now we call SetField() method to set up a new value to the column faculty_name in the Faculty table. This new name will work as the current version of this DataColumn object's value. The second prototype of this method is used here, and you can try to use any one of the other two prototypes if you like.
- E. The Console.WriteLine() method is executed to display both the original and the current values of the DataColumn object faculty_name in the Faculty table.
- F. The database connection is closed after this query is done.

Now you can build and run the project to test the functionality of the method SetField(). Click on the Debug|Start Without Debugging menu item to run the project. You can find that both the original and the current version of the DataColumn object faculty_name is retrieved and displayed in the console window.

A complete C# Console project named DataRowSetFieldLINQ can be found in the folder DBProjects\Chapter 4 located at the accompanying ftp site (see Chapter 1).

4.5.3 Operations to DataTable Objects

Besides the DataRow operators defined in the DataRowExtensions class, there are some other extension methods that can be used to work for the DataTable class defined in the DataTableExtensions class. Extension methods enable you to “add” methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are a special kind of static method, but they are called as if they were instance methods on the extended type. For client code written in C#, there is no apparent difference between calling an extension method and the methods that are actually defined in a type.

The most common extension methods are the LINQ standard query operators that add query functionality to the existing IEnumerable and IEnumerable<T> types. To use the standard query operators, first bring them into scope with a System.Linq directive. Then any type that implements IEnumerable<T> appears to have instance methods. You can see these additional methods in IntelliSense statement completion when you type a dot operator after an instance of an IEnumerable<T> type such as List<T> or Array.

Two extension methods defined in the DataTableExtensions class, AsEnumerable() and CopyToDataTable(), are widely implemented in most data-driven applications. Because of the space limitation, we only discuss the first method in this section.

The functionality of the extension method AsEnumerable() is to convert and return a sequence of type IEnumerable<DataRow> from a DataTable object. Some readers may have already noticed that we have used this method in quite a few example projects in the previous sections. For example, in the example projects DataRowFieldLINQ and DataRowSetFieldLINQ we discussed in the last section, you can find this method and its functionality. Refer to Figures 4.38 and 4.39 to get a clear picture of how to use this method to return a DataRow object.

Next let's have our discussion about the LINQ to SQL query.

4.6 LINQ TO SQL

As we mentioned in the previous section, LINQ to SQL belongs to LINQ to ADO.NET, and it is a subcomponent of LINQ to ADO.NET. LINQ to SQL is absolutely implemented to the SQL Server database. Different databases need to use different LINQ models to perform the associated queries, such as LINQ to MySQL, LINQ to DB2, or LINQ to Oracle.

LINQ to SQL query is performed on classes that implement the `IQueryable<T>` interface. Since the `IQueryable<T>` interface is inherited from the `IEnumerable<T>` with additional components, therefore besides the SGO, the LINQ to SQL queries have additional query operators since it uses the `IQueryable<T>` interface.

LINQ to SQL is an application programming interface (API) that allows users to easily and conveniently access the SQL Server database from the SGO related to the LINQ. To use this API, you must first convert your data tables in the relational database that is built based on a relational logic model to the related entity classes that are built based on the objects model, and then set up a mapping relationship between your relational database and a group of objects that are instantiated from entity classes. The LINQ to SQL or the SGO will interface to these entity classes to perform the real database operations. In other words, each entity class can be mapped or is equivalent to a physical data table in the database, and each entity class's property can be mapped or is equivalent to a data column in that table. Only after this mapping relationship has been setup can one use the LINQ to SQL to access and manipulate data against the databases.

After entity classes are created and the mapping relationships between each physical table and each entity class has been built, the conversion for data operations between the entity class and the real data table is needed. The class `DataContext` will work in this role. Basically, the `DataContext` is a connection class used to establish a connection between your project and your database. In addition to this connection role, the `DataContext` also provides the conversion function to convert or interpret operations of the SGO for the entity classes to the SQL statements that can be run in real databases.

Two tools provided by LINQ to SQL are `SQLMetal` and the `Object Relational Designer`. With the help of these tools, users can easily build all required entity classes, set the mapping relationships between the relational database and the objects model used in LINQ to SQL, and create our `DataContext` object.

The difference between the `SQLMetal` and the `Object Relational Designer` is that the former is a console-based application but the latter is a window-based application. This means that the `SQLMetal` provides a DOS-like template, and the operations are performed by entering single commands into a black-white window. The `Object Relational Designer` provides a graphic user interface (GUI) and allows the user to drag-place tables represented by graphic icons into the GUI. Obviously, the second method or tool is more convenient and easier compared with the first one.

We will process this section in the following three parts:

1. LINQ to SQL entity classes and `DataContext` class
2. LINQ to SQL database operations
3. LINQ to SQL implementations

Let's start from the first part and provide an introduction to entity classes and `DataContext` object.

4.6.1 LINQ to SQL Entity Classes and DataContext Class

As we discussed in the last section, to use LINQ to SQL to perform data queries we must convert our relational database to the associated entity classes using either SQLMetal or Object Relational Designer tools. Also we need to set up a connection between our project and the database using the DataContext object. In this section, we show users how to create entity classes and add the DataContext object to connect to our sample database CSE_DEPT.mdf using a real project SQLSelectRTOBJECTLINQ, which is a blank project with five form windows and is located in the folder DBProjects\Chapter 5 at the accompanying ftp site (see Chapter 1), and you can copy and paste it into your folder to use it.

The procedure to use LINQ to SQL to perform data actions against the SQL Server database can be described as a sequence:

1. Add the System.Data.Linq.dll assembly into the project that will use LINQ to SQL by adding the reference System.Data.Linq.
2. Create an entity class for each data table by using one of two popular tools: SQLMetal or Object Relational Designer.
3. Add a connection to the selected database using the DataContext class or the derived class from the DataContext class.
4. Use LINQ to SQL to access the database to perform the desired data actions.

Open the blank project SQLSelectRTOBJECTLINQ and open the LogIn form window by clicking on it from the Solution Explorer window. We need to develop this form based on the steps listed above.

First, we need to add the System.Data.Linq.dll assembly into the project by adding the reference System.Data.Linq. We need to do this in two steps:

1. First, we need to add a reference to our project.
2. Second, we need to add a namespace to the code window of the related form.

Let's start from the first step. Right-click on our project SQLSelectRTOBJECTLINQ from the Solution Explorer window, and select the Add Reference item from the pop-up menu to open the Add Reference dialog. Keep the default tab, NET selected and scroll down the list until you find the item System.Data.Linq. Select it by clicking on it and then click on the OK button to add this reference to our project.

Now open the code window of the LogIn form and add the namespace System.Data.Linq to the namespace declaration section in this code window.

Next we need to create entity classes to set up mapping relationships between each physical data table and the related entity class. We prefer to use the Object Relational Designer in this project since it provides a graphic user interface and is easy to use.

To open the Object Relational Designer, right-click on our project SQLSelectRTOBJECTLINQ from the Solution Explorer window, and then select the item Add\New Item from the pop-up menu to open the Add New Item dialog. On the opened dialog, select the item LINQ to SQL Classes by clicking on it, and enter CSE_DEPT.dbml into the Name box as the name for this intermediate DBML file, which is shown in Figure 4.40. Then click on the Add button to open this Object Relational Designer.

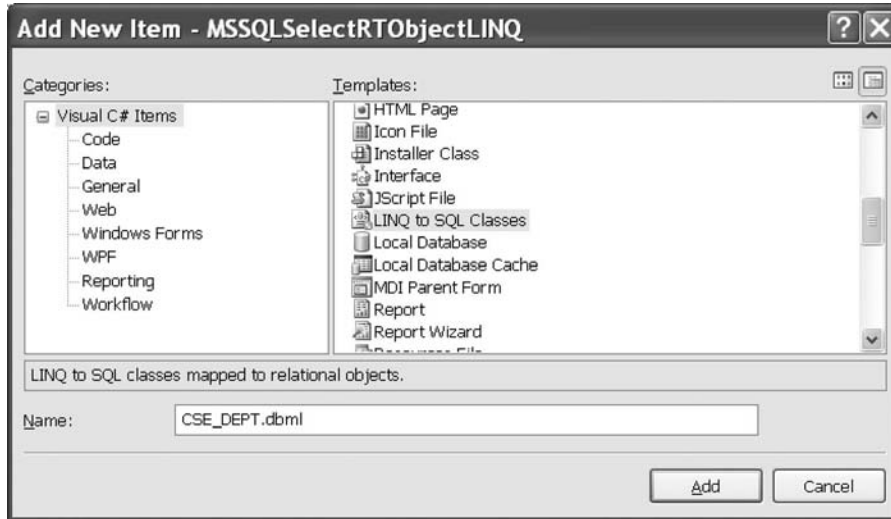


Figure 4.40 Add New item dialog.

The intermediate DBML file is an optional file when you create the entity classes, and this file allows you to control and modify the names of those created entity classes and properties; it gives you flexibility or controllability over the entity classes. You can use any meaningful name for this DBML file, but normally the name should be identical with the database's name. Therefore we used CSE_DEPT, which is our database's name as the name for this file. The opened Object Relational Designer is shown in Figure 4.41.

You can find that a CSE_DEPT.dbml folder has been added into our project in the Solution Explorer window, which is shown in Figure 4.41. Two related files, CSE_DEPT.dbml.layout and CSE_DEPT.designer.cs, are attached under that folder. The first file is basically the designer shown as a blank window in Figure 4.41, and the second file is autogenerated by the Object Relational Designer and it contains the codes to create a child class CSE_DEPTDataContext that is derived from the DataContext class. Four overloaded constructors of the child class CSE_DEPTDataContext are declared in this file.

Now we need to connect our sample SQL Server database CSE_DEPT to this project using the DataContext object. You can open our sample database file CSE_DEPT.mdf from the Server Explorer window if you connected this database to our project before. Otherwise you must add a new connection to our sample database. To do that, open the Server Explorer if it is not opened by going to View|Server Explorer menu item. Right-click on the Data Connections node and choose the Add Connection menu item to open the Add Connection dialog, which is shown in Figure 4.42.

Click on the Change button and select the Microsoft SQL Server Database File item from the Data source box, and click on the OK button to select this data source. Click on the Browse button to scan and find our sample database file CSE_DEPT.mdf on your computer, select this file, and click the Open button to add this database file into our connection object. You can test this connection by clicking on the Test Connection button and a successful connection message will be displayed if this connection is fine. One point to be noted is that you need to change the User Instance property to False if you reinstall

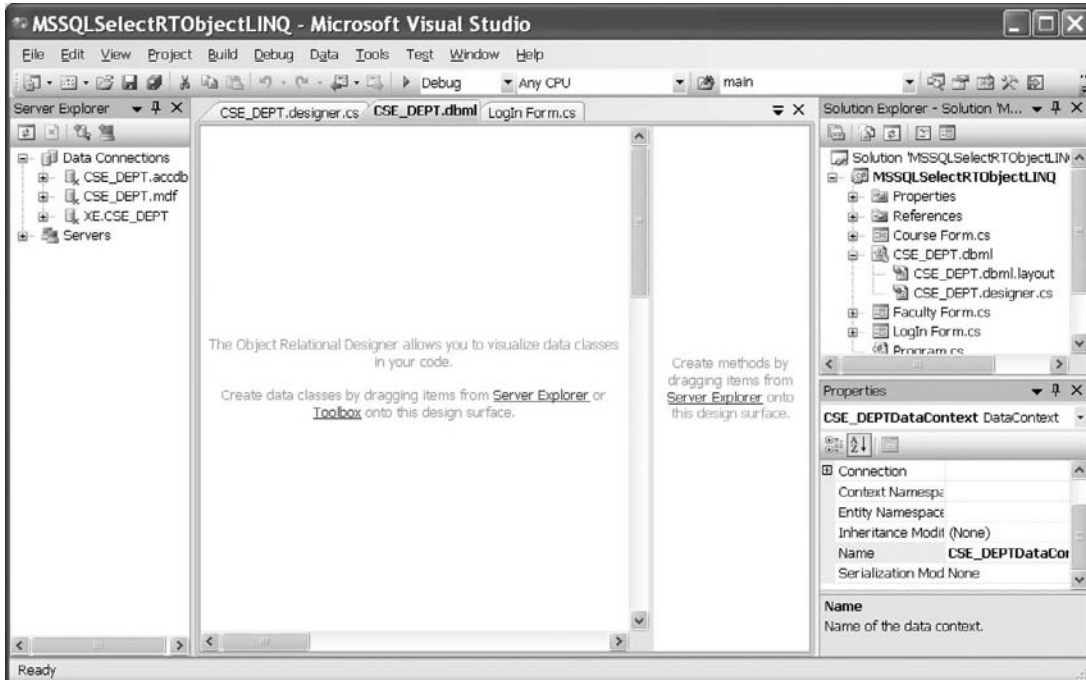


Figure 4.41 Opened Object Relational Designer.



Figure 4.42 Add Connection dialog box.

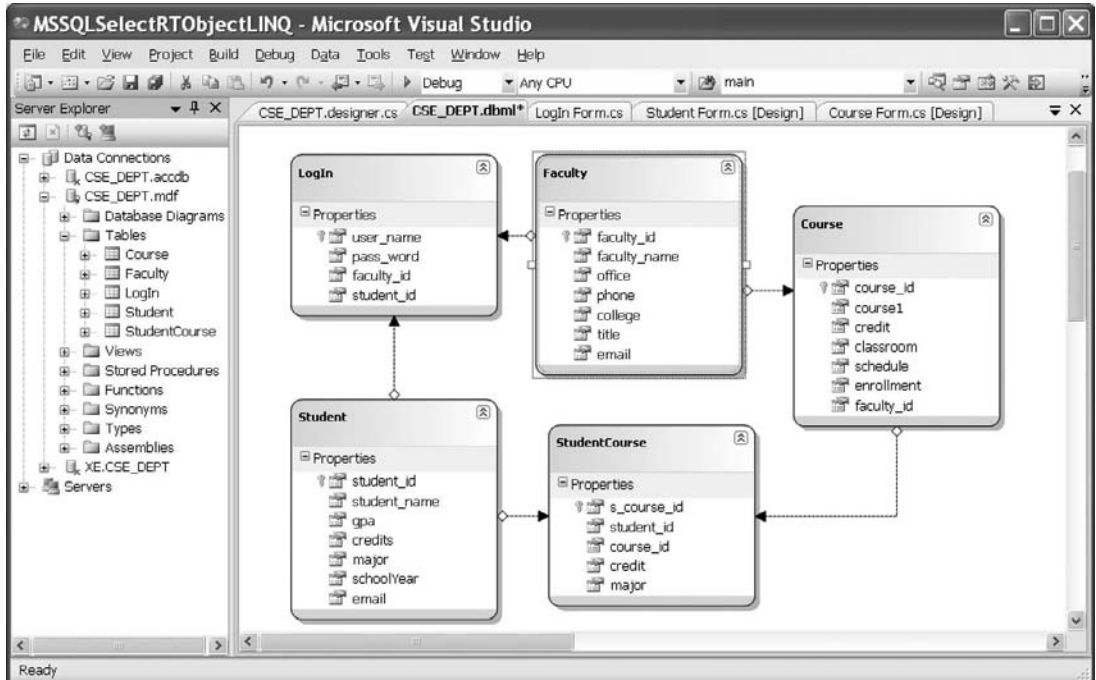


Figure 4.43 Finished Designer.

the Microsoft SQL Server 2005 after removing an old version of Microsoft SQL Server database by using the Advanced setup function (click on the Advanced button). Otherwise you do not need to do this setup action. Another point is that you can find our sample database CSE_DEPT.mdf in the folder Database\SQLServer located at the accompanying ftp site (see Chapter 1). You can copy this database file and save it to any folder in your computer.

Click on the OK button to close this dialog. Your finished Add Connection dialog box should match one that is shown in Figure 4.42.

Now you can find that a node named CSE_DEPT.mdf under the Data Connections node has been added into the Server Explorer window. Expand this database file to the Tables node and you can find all of the five data tables.

To create an entity class for each table, just perform a drag-place operation for each table between the Server Explorer window and the blank Design window. Starting from the LogIn table, drag it from the Server Explorer window and place it in the Design window. By dragging the LogIn table to the designer canvas, the source code for the LogIn entity class is created and added into the CSE_DEPT.designer.cs file. Then you can use this entity class to access and manipulate data from this project to the LogIn table in our sample database CSE_DEPT.

Perform the similar drag-place operations to all other tables, and your finished design should match the one shown in Figure 4.43.

The arrow between tables is called an association, which is a new terminology used in the LINQ to SQL, and it represents the relationship between tables.

Now we can start to use these entity classes and the DataContext object to perform the data actions against our sample database using the LINQ to SQL technique.

4.6.2 LINQ to SQL Database Operations

In this section, we provide a fundamental end-to-end LINQ to SQL scenario for adding, modifying, and deleting data in a database. As you know, LINQ to SQL queries can perform not only the data selections but also data insertion, updating, and deletion. The standard LINQ to SQL queries include:

- Select
- Insert
- Update
- Delete

To perform any of these operations or queries, we need to use entity classes and DataContext, which we discussed in the last section, to do LINQ to SQL actions against our sample database. Because the blank project SQLSelectRTOBJECTLINQ will be used in Chapter 5, we had better create another C# console sample project to illustrate how to use LINQ to SQL to perform data queries against our sample database CSE_DEPT.mdf.

Create a new C# console project and name it QueryLINQSQL. Perform the following operations to prepare our LINQ to SQL queries:

1. Add System.Data.Linq reference to this new project by right-clicking on our new project from the Solution Explorer window, and scroll down and select the item System.Data.Linq from the list and click on the OK button.
2. Add the following directives at the top of Program.cs file:
 - Using System.Data.Linq
 - Using System.Data.Linq.Mapping
3. Follow the steps listed in Section 4.6.1 to create entity classes using the Object Relational Designer. The database used in this project is **CSE_DEPT.mdf**, and it is located in the folder C:\database\SQLServer. Open the Server Explorer window and add this database by right clicking on the Data Connections item and select Add Connection.
4. We need to create five entity classes and each of them is associated with a data table in our sample database. Drag each table from the Server Explorer window and place it on the Object Relational Designer canvas. The mapping file's name is CSE_DEPT.dbml. Make sure that you enter this name into the Name box in the Object Relational Designer.
5. Right-click the mapping file CSE_DEPT.dbml from the Solution Explorer window and select View Code item to create a C# code file for our database, CSE_DEPT.cs.

Now open the code window of this new project and enter the code shown in Figure 4.44 into this window.

Let's take a closer look at this piece of code to see how it works.

- A. Two namespaces, System.Data.Linq and System.Data.Linq.Mapping, are added into the namespace declaration part of this project since we need to use some data components defined in LINQ to SQL namespaces.


```

QueryLINQSQL.Program Main()
using System;
using System.Collections.Generic;
using System.Linq;
A using System.Data.Linq;
using System.Data.Linq.Mapping;

namespace QueryLINQSQL
{
    class Program
    {
        static void Main(string[] args)
        {
B         CSE_DEPTDataContext cse_dept = new CSE_DEPTDataContext();
C         Console.WriteLine("Make your selection:\n");
            Console.WriteLine("1: LINQ to SQL Select query");
            Console.WriteLine("2: LINQ to SQL Insert query");
            Console.WriteLine("3: LINQ to SQL Update query");
            Console.WriteLine("4: LINQ to SQL Delete query");
            Console.WriteLine("5: Exit the project\n");
D         string input = Console.ReadLine();
E         switch (input.ToString())
            {
                case "1":
                    LINQSelect(cse_dept);
                    break;
                case "2":
                    LINQInsert(cse_dept);
                    break;
                case "3":
                    LINQUpdate(cse_dept);
                    break;
                case "4":
                    LINQDelete(cse_dept);
                    break;
                default:
                    break;
            }
F         // Keep the console window open after activity stops.
            Console.WriteLine("Press Enter key to continue ....");
            Console.ReadLine();
        }
        .....
    }
}

```

Figure 4.44 Coding for the main method.

- B.** A new object of the DataContext class is created since we need to use this object to connect to our sample database to perform data queries. Because we have connected this DataContext class to our sample database CSE_DEPT.mdf in step 3 in this section, and the connection string has been added into our app.config file when step 3 is done. Therefore we do not need to indicate the special connection string when we create this object.
- C.** A customer running menu is displayed to allow users to select an item to perform the desired query.
- D.** The users' answer is received by running a system method Console.ReadLine().
- E.** A switch block is used to identify the user's selection and direct the program to the associated method to perform the desired query.
- F.** The purpose of these two lines is to allow users to run this project in the Debugging mode and keep the console window open as the project runs.


```

QueryLINQSQL.Program LINQSelect()
public static void LINQSelect(CSE_DEPTDataContext db)
{
    var faculty = (from fi in db.Faculties
                   where fi.faculty_id == "B78880"
                   select fi);
    foreach (var f in faculty)
    {
        Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}\n{5}", f.faculty_name, f.title,
                          f.office, f.phone, f.college, f.email);
    }
}

```

Figure 4.45 Coding for the LINQSelect method.

Now that we have finished all prerequisite jobs to make a desired LINQ to SQL query, let's start from the first query, data selection.

4.6.2.1 Data Selection Query

Create a new static method LINQSelect() in this project and enter the code shown in Figure 4.45 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** The new created DataContext object is passed into this method since we need to use this object to access our sample database to perform the selection query. The method must be a static type since it will be called from another static method main().
- B.** The query is created and initialized with three clauses. The Faculties is an instance of our entity class and the faculty_id works as the query criterion for this query.
- C.** The query is executed by running a foreach loop and the queried result is displayed by calling the Console.WriteLine() method.

It can be found that the coding is very simple after the prerequisite codes have been done. Next we need to take care of the data insertion query.

4.6.2.2 Data Insertion Query

Two options can be used to insert a record into the database: (1) Insert a new record into the database, and (2) insert some new columns to an existing record in the database. Option 2 is similar to the data updating query; therefore, we concentrate on the first option in this section only.

Create a new method LINQInsert() in this project and enter the code shown in Figure 4.46 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** The new created DataContext object is passed into this method since we need to use this object to access our sample database to perform the selection query. The method must be a static type since it will be called from another static method main().
- B.** A new Faculty entity object is created and it is equivalent to a DataRow object.
- C.** The new created Faculty entity object is initialized with all desired columns' values.

QueryLINQSQL.Program	LINQInsert()
A	<code>public static void LINQInsert(CSE_DEPTDataContext db)</code>
	<code>{</code>
B	<code> // Create the new Faculty object.</code>
C	<code> Faculty newFaculty = new Faculty();</code>
	<code> newFaculty.faculty_id = "D19886";</code>
	<code> newFaculty.faculty_name = "David Winner";</code>
	<code> newFaculty.title = "Department Chair";</code>
	<code> newFaculty.office = "MTC-333";</code>
	<code> newFaculty.phone = "750-330-1255";</code>
	<code> newFaculty.college = "University of Hawaii";</code>
	<code> newFaculty.email = "dwinner@college.edu";</code>
	<code> // Add the faculty to the Faculty table.</code>
D	<code> db.Faculties.InsertOnSubmit(newFaculty);</code>
E	<code> db.SubmitChanges();</code>
	<code> // Query back the new inserted faculty member</code>
F	<code> Faculty fi = db.Faculties.Where(f => f.faculty_id == "D19886").First();</code>
	<code> Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}\n{5}\n",</code>
	<code> fi.faculty_name, fi.title,</code>
	<code> fi.office, fi.phone, fi.college, fi.email);</code>
	<code> // Reset the database by deleting the new inserted faculty</code>
G	<code> db.Faculties.DeleteOnSubmit(newFaculty);</code>
H	<code> db.SubmitChanges();</code>
	<code>}</code>

Figure 4.46 Coding for the LINQInsert method.

- D.** The `InsertOnSubmit()` method is executed to add this new object into the Faculty entity object (Faculty data table). One point to be noted is that this new record can be exactly added into the Faculty table only after the next method `SubmitChanges()` is executed.
- E.** The method `SubmitChanges()` is executed to insert this new record into the database.
- F.** These two lines of code are used to retrieve back and display the new inserted record to confirm our data insertion operation. Two SQO methods, `Where()` and `First()`, and a Lambda operator `=>` are used in this query. Do not worry about the Lambda operator at this moment, and we will provide a very detailed discussion for this topic in Section 4.9.
- G.** To make our database clean, we need to delete this new inserted record from our database after this insertion is successful.
- H.** The new inserted record is deleted after the method `SubmitChanges()` is executed.

Next let's take care of the data updating query.

4.6.2.3 Data Updating Query

Basically, to perform a data updating operation, the following operation sequence should be followed:

1. Child table: delete records.
2. Parent table: insert, update, and delete records.
3. Child table: insert and update records.

For our sample database CSE_DEPT, all five tables are related with different primary keys and foreign keys. For example, among the LogIn, Faculty, and Course tables, the `faculty_id` is a primary key in the Faculty table, but a foreign key in both LogIn and the Course tables. In order to update or delete data from any of those tables, one needs to

follow the sequence above. As a case of updating or deleting a record against the database, the following data operations need to be performed:

1. Remove or delete those records that are foreign key but related to the primary key in the parent table from the child tables, such as LogIn and Course tables, respectively
2. Update or delete those records that are primary keys from the parent table, such as Faculty table.
3. Finally the updated record can be inserted into the child tables such as Login and Course tables for the data updating operation. There are no data actions for the data deleting operations for the child tables.

It would be terribly complicated if we try to update or delete a completed record (including update or delete the primary key) for an existing data in our sample database because of the relationships between the parent and child tables. We will provide a detailed discussion about the data updating and deleting queries against a relational database in Chapter 7. In this section, to make it simple, we only show users how to update a single entity class's property (a single column in a data table).

The following example code shows how to update the `faculty_name` column from the Faculty table. Create a new method `LINQUpdate()` in this project and enter the code shown in Figure 4.47 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. A selection query is executed using the SGO methods with the `faculty_id` as the query criterion. The `First()` method is used to return only the first matched record. It does not matter for our application since we have only one record that is associated with this specified `faculty_id`.
- B. Before the data updating query can be performed, the original record with the `faculty_name` is displayed.
- C. Update the `faculty_name` column to "New Faculty".
- D. This updating is officially effective after this `SubmitChanges()` method is executed.

```

QueryLINQSQL.Program LINQUpdate()
public static void LINQUpdate(CSE_DEPTDataContext db)
{
    Faculty fi = db.Faculties.Where(f => f.faculty_id == "B78880").First();
    // Display the existing faculty information
    Console.WriteLine("Before the Faculty table is updated...\n");
    Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}\n{5}\n", fi.faculty_name, fi.title,
        fi.office, fi.phone, fi.college, fi.email);

    // Change the faculty name.
    fi.faculty_name = "New Faculty";
    db.SubmitChanges();
    Console.WriteLine("After the Faculty table is updated...\n");
    Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}\n{5}\n", fi.faculty_name, fi.title,
        fi.office, fi.phone, fi.college, fi.email);

    // Recover the original column for the Faculty table
    fi.faculty_name = "Ying Bai";
    db.SubmitChanges();
}

```

Figure 4.47 Coding for the LINQUpdate method.

- E. The updated record is displayed again to compare with the original record.
- F. This coding line is used to recover the faculty_name to its original value.
- G. This recovery will be officially effective after the SubmitChanges() method is executed.

Finally let's take care of the data deleting query.

4.6.2.4 Data Deletion Query

Because of the data integrity in a relational database, deleting a record from a parent table is very complicated since all related records in the child tables must be deleted first, and then the record in the parent table can be deleted. Fortunately, we can make this deleting quite simple by using the Cascaded Deleting property we set up in our sample database CSE_DEPT. Recall that in Sections 2.10.4.1 and 2.10.4.3, we set the Delete Rule as Cascade when we built the relationships between the LogIn and the Faculty tables and between the Faculty and the Course tables in our sample database. The purpose of this setup allows all related records in child tables to be cascaded and removed if an associated record in the parent table is deleted.

To delete a record from a database using LINQ to SQL, one must delete the entity object (equivalent to DataRow object) from the Table<T> of which it is a member with the Table<T> object's DeleteOnSubmit() method.

The following example code shows how to delete a record from the Faculty table that is a parent table and delete all related records from the child tables (LogIn and Course) using the cascade property.

Create a new method LINQDelete() in this project and enter the code shown in Figure 4.48 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. First a Select query is created and performed to retrieve a record from the Faculty table. The query criterion is a specific faculty_id, B78880. The SQO method Single() is used to retrieve back only a single row.
- B. The queried row faculty is placed into the deleting pool with the DeleteOnSubmit() method.

```

QueryLINQSQL.Program LINQDelete()
public static void LINQDelete(CSE_DEPTDataContext db)
{
    A      var faculty = (from fi in db.Faculties
                        where fi.faculty_id == "B78880"
                        select fi).Single<Faculty>());
    B      db.Faculties.DeleteOnSubmit(faculty);
    C      db.SubmitChanges();
    // Try to retrieve back and display the deleted faculty information
    D      var delfaculty = (from fi in db.Faculties
                           where fi.faculty_id == "B78880"
                           select fi).SingleOrDefault<Faculty>();
    E      Console.WriteLine("Faculty {0} found.\n", delfaculty != null? "is":"is not");
}

```

Figure 4.48 Coding for the LINQDelete method.

- C. The selected record from the Faculty table as well as the related records from the child tables, LogIn and Course, are deleted by executing the SubmitChanges() method.
- D. To verify this deleting query, a similar Select query is performed again to try to pick up the deleted record from the Faculty table. The SQO method SingleOrDefault() is used to make sure that either a matched single row or a default row can be retrieved and returned from this query.
- E. The returned result is displayed by executing the method Console.WriteLine(). The conditional operator delfaculty != null works as a Boolean operator. The “is” will be filled into the {0} if the returned result is a valid record (delfaculty != null is true). Otherwise the “is not” will be filled if no matched result can be found and returned (delfaculty != null is false).

Note that this deletion is a permanent deletion, which means that if this query is performed, both the matched record in the parent (Faculty) table and all related records in the child (LogIn and Course) tables will also be deleted permanently. It is highly recommended that you need to recover these three tables by adding those deleted records back to those tables as soon as you have finished this deletion operation in this project.

Now you can build and run this project to test all LINQ to SQL queries developed in this project. A complete C# Console project named QueryLINQSQL can be found in the folder DBProjects\Chapter 4 located at the accompanying ftp site (see Chapter 1).

As we mentioned, if you performed the deletion query from this project, you need to recover those deleted records for three tables. Now let's first open those three tables to check what and how many records have been deleted by running this deletion query.

Depending on how you use this sample database in this project, different databases should be opened. If you integrate this database with your project, the database file CSE_DEPT.mdf should be located at the folder C:\Book 6\Chapter 4\QueryLINQSQL\QueryLINQSQL\bin\Debug. If you did not integrate this database with your project, the database file should be located at the folder C:\database\SQLServer. You need to go to the folder in which you stored this sample database file if you save it in any other location.

Open the Microsoft SQL Server Management Studio Express and expand the Databases folder to find your database. Then expand your database to locate all data tables by expanding the Tables folder. Right-click on a data table and select the Open Table item to open that table. We need to open the following three tables: Faculty, LogIn, and Course.

On the opened Faculty table, you can find that one record whose faculty_id is B78880 has been deleted from this table. The deleted record is shown in Table 4.3. On the opened LogIn table, you can find that one record whose user_name is ybai has also been removed from this table and the removed record is shown in Table 4.4. On the opened Course

Table 4.3 Deleted Record from the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu

Table 4.4 Deleted Record from the LogIn Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	

Table 4.5 Deleted Records from the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880

table, you can find that the following courses whose `faculty_id` is `B78880` have been deleted from this table. Those deleted course records are shown in Table 4.5.

Recover those deleted records for these three tables by adding each record shown in Tables 4.3, 4.4, and 4.5 into the associated table in Microsoft SQL Server Management Studio Express. Remember you need to perform this recovery operation each time after you perform this deleting query in this project `QueryLINQSQL`.

Our LINQ to SQL project is successful! Do not forget to close the Microsoft SQL Server Management Studio Express when this recovery job is done.

4.6.3 LINQ to SQL Implementations

Quite a few real projects that use LINQ to SQL queries will be developed in the following chapters, and those projects are categorized based on the following chapters:

- LINQ to SQL Select query projects: Chapters 5, 8, and 9
- LINQ to SQL Insert query projects: Chapters 6, 8, and 9
- LINQ to SQL Update query projects: Chapters 7, 8, and 9
- LINQ to SQL Delete query projects: Chapters 7, 8, and 9

Refer to those chapters to get more detailed information and related code developments for those projects.

4.7 LINQ TO ENTITIES

As we mentioned in the introduction to the LINQ section, LINQ to Entities belongs to LINQ to ADO.NET and it is a subcomponent of LINQ to ADO.NET. LINQ to Entities queries are performed under the control of the ADO.NET 3.5 Entity Framework (ADO.NET 3.5 EF) and ADO.NET 3.5 Entity Framework Tools (ADO.NET 3.5 EFT). ADO.NET 3.5 EF enables developers to work with data in the form of domain-specific objects and properties, such as customers and customer addresses, without having to think about the underlying database tables and columns where this data is stored. To access and implement ADO.NET 3.5 EF and ADO.NET 3.5 EFT, developers need to understand the Entity Data Model, which is the core of ADO.NET 3.5 EF. LINQ allows developers to formulate set-based queries in their application code, without having to use a separate query language. Through the Object Services infrastructure of Entity Framework, ADO.NET exposes a common conceptual view of data, including relational data, as objects in the .NET environment. This makes the object layer an ideal target for LINQ support.

This LINQ technology, LINQ to Entities, allows developers to create flexible, strongly typed queries against the Entity Framework object context by using LINQ expressions and the LINQ standard query operators directly from the development environment. The queries are expressed in the programming language itself and not as string literals embedded in the application code, as is usually the case in applications written in the Microsoft .NET Framework 2.0. Syntax errors as well as errors in member names and data types will be caught by the compiler and reported at compile time, reducing the potential for type problems between the Entity Data Model and the application.

LINQ to Entities queries use the Object Services infrastructure. The `ObjectContext` class is the primary class for interacting with an Entity Data Model as CLR objects. The developer constructs a generic `ObjectQuery` instance through the `ObjectContext`. The `ObjectQuery` generic class represents a query that returns an instance or collection of typed entities. The returned entity objects are updatable and are located in the object context. This is also true for entity objects that are returned as members of anonymous types.

4.7.1 Object Services Component

Object Services is a component of the Entity Framework that enables you to query, insert, update, and delete data, expressed as strongly typed common language runtime (CLR) objects that are instances of entity types. Object Services supports both LINQ and Entity SQL queries against types defined in an Entity Data Model (EDM). Object Services materializes returned data as objects and propagates object changes back to the persisted data store. It also provides facilities for tracking changes, binding objects to controls, and handling concurrency. Object Services is implemented by classes in the `System.Data.Objects` and `System.Data.Objects.DataClasses` namespaces.

4.7.2 ObjectContext Component

The `ObjectContext` class encapsulates a connection between the .NET Framework and the database. This class serves as a gateway for Create, Read, Update, and Delete operations, and it is the primary class for interacting with data in the form of objects that are instances of entity types defined in an EDM. An instance of the `ObjectContext` class encapsulates the following:

- A connection to the database, in the form of an `EntityConnection` object
- Metadata that describes the model, in the form of a `MetadataWorkspace` object
- An `ObjectStateManager` object that manages objects persisting in the cache

The Entity Framework tools consume a conceptual schema definition language (CSDL) file from a relational database and generate the object-layer code. This code is used to work with entity data as objects and to take advantage of Object Services functionality. This generated code includes the following data classes:

- A class that represents the `EntityContainer` for the model and is derived from `ObjectContext`
- Classes that represent entities and inherit from `EntityObject`

4.7.3 ObjectQuery Component

The `ObjectQuery` generic class represents a query that returns a collection of zero or more typed entities. An object query always belongs to an existing object context. This context provides the connection and metadata information required to compose and execute the query.

4.7.4 LINQ to Entities Flow of Execution

Queries against the Entity Framework are represented by command tree queries, which execute against the object context. LINQ to Entities converts LINQ queries to command tree queries, executes the queries against the Entity Framework, and returns objects that can be used by both the Entity Framework and LINQ. The following is the process for creating and executing a LINQ to Entities query:

1. Construct an `ObjectQuery` instance from `ObjectContext`.
2. Compose a LINQ to Entities query in C# by using the `ObjectQuery` instance.
3. LINQ Standard Query Operators and expressions in query are converted to command trees.
4. The query, in command tree representation, is executed against the data store. Any exceptions thrown on the data store during execution are passed directly up to the client.
5. Query results are materialized back to the client.

Let's take a little detailed discussion for each of these steps.

Construct an `ObjectQuery` Instance The `ObjectQuery` generic class represents a query that returns a collection of zero or more typed entities. An object query is typically constructed from an existing object context, instead of being manually constructed, and always belongs to that object context. This context provides the connection and metadata information that is required to compose and execute the query. The `ObjectQuery` generic class implements the `IQueryable` generic interface, whose builder methods enable LINQ queries to be incrementally built.

Compose a LINQ to Entities Query Instances of the `ObjectQuery` generic class, which implements the generic `IQueryable` interface, serve as the data source for LINQ to Entities queries. In a query, you specify exactly the information that you want to retrieve from the data source. A query can also specify how that information should be sorted, grouped, and shaped before it is returned. In LINQ, a query is stored in a variable. This query variable takes no action and returns no data; it only stores the query information. After you create a query you must execute that query to retrieve any data.

LINQ to Entities queries can be composed in two different syntaxes: query expression syntax and method-based query syntax. We have provided a very detailed discussion about the query expression syntax and the method-based query syntax with real example codes in Sections 4.5.1.1 and 4.5.1.2. Refer to those sections to get a clear picture for these two syntaxes.


```
IQueryable<string> FacultyInfo = from fi in Faculties
                                where fi.faculty_id == "B78880"
                                select fi.faculty_name;
```

Figure 4.49 Example of expression used in LINQ to Entities.

Convert the Query to Command Trees To execute a LINQ to Entities query against the Entity Framework, the LINQ query must be converted to a command tree representation that can be executed against the Entity Framework.

LINQ to Entities queries are comprised of LINQ Standard Query Operators (such as Select, Where, and OrderBy) and expressions. LINQ Standard Query Operators are not defined by a class but rather are static methods in a class. In LINQ, expressions can contain anything allowed by types within the System.Expressions namespace and, by extension, anything that can be represented in a lambda function. This is a superset of the expressions that are allowed by the Entity Framework, which are by definition restricted to operations allowed on the database and supported by ObjectQuery.

In the Entity Framework, both operators and expressions are represented by a single type of hierarchy, which are then placed in a command tree. The command tree is used by the Entity Framework to execute the query. If the LINQ query cannot be expressed as a command tree, an exception will be thrown when the query is being converted. The conversion of LINQ to Entities queries involves two subconversions: the conversion of the Standard Query Operators and the conversion of the expressions. In general, expressions in LINQ to Entities are evaluated on the server, so the behavior of the expression should not be expected to follow CLR semantics. An example of an expression used in LINQ to Entities is shown in Figure 4.49.

Execute the Query After the LINQ query is created by the user, it is converted to a representation that is compatible with the Entity Framework (in the form of command trees), which is then executed against the store. At query execution time, all query expressions (or components of the query) are evaluated on the client or on the server. This includes expressions that are used in result materialization or entity projections.

A query expression can be executed in two ways. LINQ queries are executed each time the query variable is iterated, not when the query variable is created; this is referred to as deferred execution. The query can also be forced to execute immediately, which is useful for caching query results. The example shown in Figure 4.50 uses **select** to return all the rows from *Faculty* and displays the faculty names. Iterating over the query variable in the **foreach** loop causes the query to execute.

When a LINQ to Entities query is executed, some expressions in the query might be executed on the server and some parts might be executed locally on the client. Client-side evaluation of an expression takes place before the query is executed on the server. If an expression is evaluated on the client, the result of that evaluation is substituted for the expression in the query, and the query is then executed on the server. Because queries are executed on the data store, the data store configuration overrides the behavior specified in the client. Null value handling and numerical precision are examples of this. Any exceptions thrown during query execution on the server are passed directly up to the client.

```

ObjectQuery<Faculty> faculties = cse_dept.Faculty;
IQueryable<string> FacultyNames = from f in faculties
                                   select f.faculty_name;

Console.WriteLine("Faculty Names:");
foreach (var fName in FacultyNames)
{
    Console.WriteLine(fName);
}

```

Figure 4.50 Example of executing the query.

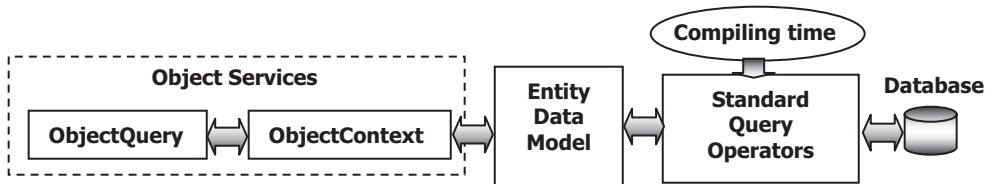


Figure 4.51 Simplified structure of LINQ to Entities.

Materialize the Query Materialization is the process of returning query results back to the client as CLR types. In LINQ to Entities, query results data records are never returned; there is always a backing CLR type, defined by the user or by the Entity framework, or generated by the compiler (anonymous types). All object materialization is performed by the Entity Framework. Any errors that result from an inability to map between the Entity Framework and the CLR will cause exceptions to be thrown during object materialization.

Query results are usually returned as one of the following:

- A collection of zero or more typed entity objects or a projection of complex types in the Entity Data Model
- CLR types supported by the Entity Data Model
- Inline collections
- Anonymous types
- IGrouping instances
- IQueryable instances

A simplified structure of LINQ to Entities is shown in Figure 4.51.

We have provided a very detailed discussion about the structure and components used in LINQ to Entities query. Next we need to illustrate these by using some examples.

4.7.5 Implementation of LINQ to Entities

In order to use LINQ to Entities query to perform data actions against databases, one needs to have a clear picture about the infrastructure and fully understanding about components used in LINQ to Entities. In Section 3.4.8 in Chapter 3, we provided a very

detailed discussion about the ADO.NET 3.5 Entity Framework and ADO.NET 3.5 Entity Data Model, including the Entity Data Model Wizard, Entity Data Model Designer, and Entity Model Browser with a real example project EDMModel. Review that section to get more details on the implementation of LINQ to Entities. A complete example project EDMModel that uses LINQ to Entities query can be found in the folder DBProjects\Chapter 5 located at the accompanying ftp site (see Chapter 1).

4.8 LINQ TO XML

LINQ to XML was developed with Language-Integrated Query over XML in mind and takes advantage of standard query operators and adds query extensions specific to XML. LINQ to XML is a modernized in-memory XML programming API designed to take advantage of the latest .NET Framework language innovations. It provides both DOM and XQuery/XPath like functionality in a consistent programming experience across the different LINQ-enabled data access technologies.

There are two major perspectives for thinking about and understanding LINQ to XML. From one perspective you can think of LINQ to XML as a member of the LINQ Project family of technologies with LINQ to XML providing an XML Language-Integrated Query capability along with a consistent query experience for objects, relational database (LINQ to SQL, LINQ to DataSet, LINQ to Entities), and other data access technologies as they become LINQ-enabled. From another perspective you can think of LINQ to XML as a full-feature in-memory XML programming API comparable to a modernized, redesigned Document Object Model XML programming API plus a few key features from XPath and XSLT.

LINQ to XML is designed to be a lightweight XML programming API. This is true from both a conceptual perspective, emphasizing a straightforward, easy-to-use programming model, and from a memory and performance perspective. Its public data model is aligned as much as possible with the W3C XML Information Set.

4.8.1 LINQ to XML Class Hierarchy

First let's have a global picture about the LINQ to XML Class Hierarchy shown in Figure 4.52.

The following important points should be noted when studying this class hierarchy:

1. Although XElement is low in the class hierarchy, it is the fundamental class in LINQ to XML. XML trees are generally made up of a tree of XElements. XAttributes are name/value pairs associated with an XElement. XDocuments are created only if necessary, such as to hold a Document Type Definitions (DTD) or top-level XML processing instruction (XProcessingInstruction). All other XNodes can only be leaf nodes under an XElement or possibly an XDocument (if they exist at the root level).
2. XAttribute and XNode are peers derived from a common base class XObject. XAttributes are not XNodes because XML attributes are really name value/pairs associated with an XML element not nodes in the XML tree. Contrast this with W3C DOM.
3. XText and XCDATA are exposed in this version of LINQ to XML, but as discussed above, it is best to think of them as a semihidden implementation detail except when exposing text

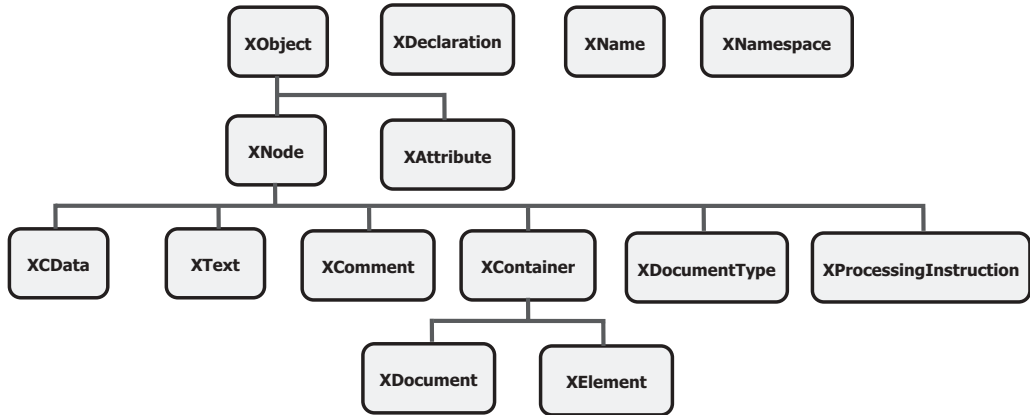


Figure 4.52 The LINQ to XML class hierarchy.

nodes is necessary. As a user, you can get back the value of the text within an element or attribute as a string or other simple value.

4. The only XNode that can have children is an XContainer, meaning either an XDocument or XElement. An XDocument can contain an XElement (the root element), an XDeclaration, an XDocumentType, or an XProcessingInstruction. An XElement can contain another XElement, an XComment, an XProcessingInstruction, and text (which can be passed in a variety of formats, but will be represented in the XML tree as text).

In addition to this class hierarchy, some other important components applied in XML also play key roles in LINQ to XML. One of them is the XML names.

XML names, often a complex subject in XML programming APIs, are represented simply in LINQ to XML. An XML name is represented by an XNamespace object [which encapsulates the XML namespace Uniform Resource Identifier (URI)] and a local name. An XML namespace serves the same purpose that a namespace does in your .NET Framework-based programs, allowing you to uniquely qualify the names of your classes. This helps ensure that you don't run into a name conflict with other users or built-in names. When you have identified an XML namespace, you can choose a local name that needs to be unique only within your identified namespace.

4.8.2 Manipulate XML Elements

LINQ to XML provides a full set of methods for manipulating XML. You can insert, delete, copy, and update XML content. Before we can continue to discuss these data actions, first we need to show you how to create a sample XML element file using LINQ to XML.

4.8.2.1 Creating XML from Scratch

LINQ to XML provides a powerful approach to creating XML elements. This is referred to as *functional construction*. Functional construction lets you create all or part of your XML tree in a single statement. For example, to create a **faculties XElement**, you could use the code shown in Figure 4.53.

```

XElement faculties = new XElement("faculties",
    new XElement("faculty",
        new XElement("faculty_name", "Patrick Tones"),
        new XElement("phone", "750-378-0144"),
        new XElement("title", "Associate Professor"),
        new XElement("office", "MTC-387"),
        new XElement("college", "Main University"),
        new XElement("email", "ptones@college.edu"),
        new XElement("faculty_id", "P68042")));

```

Figure 4.53 Sample XML file created using LINQ to XML.

By indenting, the `XElement` constructor resembles the structure of the underlying XML. Functional construction is enabled by an `XElement` constructor that takes a **params** object. An example of the Functional construction is shown below.

```
public XElement(XName faculty_name, params object[] contents)
```

The *contents* parameter is extremely flexible, supporting any type of object that is a legitimate child of an `XElement`. Parameters can be any of the following:

- A *string*, which is added as text content. This is the recommended pattern to add a string as the value of an element; the LINQ to XML implementation will create the internal `XText` node.
- An `XText`, which can have either a string or `CData` value, added as child content. This is mainly useful for `CData` values; using a *string* is simpler for ordinary string values.
- An `XElement`, which is added as a child element.
- An `XAttribute`, which is added as an attribute.
- An `XProcessingInstruction` or `XComment`, which is added as child content.
- An `IEnumerable`, which is enumerated, and these rules are applied recursively.
- Anything else, `ToString()` is called and the result is added as text content.
- *Null*, which is ignored.

The term `CDATA`, meaning *character data*, is used for distinct, but related purposes in the markup languages SGML and XML. The term indicates that a certain portion of the document is general *character data*, rather than noncharacter data or character data with a more specific, limited structure.

In the above example showing functional construction, a string (“atrick Tones”) is passed into the **faculty_name** `XElement` constructor. This could have been a variable [e.g., `new XElement("aculty_name", facultyName)`], it could have been a different type besides string [e.g., `new XElement("quantity", 55)`], and it could have been the result of a function call like the one shown in Figure 4.54.

It could also have even been the `IEnumerable<XElement>`. For example, a common scenario is to use a query within a constructor to create the inner XML. The code shown in Figure 4.55 reads faculties from an array of **Person** objects into a new XML element **faculties**.

Notice how the inner body of the XML, the repeating **faculty** element, and, for each **faculty**, the repeating **phone** were generated by queries that return an **IEnumerable**.

```

{
  ...
  XElement qty = new XElement("quantity", GetQuantity());
  ...
}
public int GetQuantity() { return 55; }

```

Figure 4.54 Sample functional construction.

```

class Person
{
  public string faculty_name;
  public string[] PhoneNumbers;
}
var persons = new[] {new Person {
  faculty_name = "Patrick Tones",
  PhoneNumbers = new[] { "750-555-0144", "750-555-0145" }},
new Person { faculty_name = "Gretchen Rivas",
  PhoneNumbers = new[] { "750-555-0163" }}};
XElement faculties = new XElement("faculties", from f in persons
  select new XElement("faculty",
    new XElement("fname", f.faculty_name),
    from p in f.PhoneNumbers
    select new XElement("phone", p)));
Console.WriteLine(faculties);

```

Figure 4.55 Sample query using LINQ to XML.

When an objective of your program is to create an XML output, functional construction lets you begin with the end in mind. You can use functional construction to shape your goal output document and either create the subtree of XML items inline or call out to functions to do the work.

Functional construction is instrumental in *transforms*, which belongs to XML Transformation. Transformation is a key usage scenario in XML, and functional construction is well-suited for this task.

Now let's use this sample XML file to discuss data manipulations using LINQ to XML.

4.8.2.2 Insert XML

You can easily add content to an existing XML tree. To add another *phone* XElement, one can use the **Add()** method shown in section **A** in Figure 4.56.

This code fragment will add the **mobilePhone** XElement as the *last* child of faculty. If you want to add to the beginning of the children, you can use **AddFirst()**. If you want to add the child in a specific location, you can navigate to a child before or after your target location by using **AddBeforeSelf()** or **AddAfterSelf()**. For example, if you wanted **mobilePhone** to be the second **phone**, you could do the coding that is shown in section **B** in Figure 4.56.

```

A XElement mobilePhone = new XElement("phone", "750-555-0168");
    faculty.Add(mobilePhone);

B XElement mobilePhone = new XElement("phone", "750-555-0168");
    XElement firstPhone = faculty.Element("phone");
    firstPhone.AddAfterSelf(mobilePhone);

C XElement mobilePhone = new XElement("phone", "750-555-0168");
    Console.WriteLine(mobilePhone.Parent); // will print out null

D faculty.Add(mobilePhone);
    Console.WriteLine(mobilePhone.Parent); // will print out faculty

E faculty2.Add(mobilePhone);

F faculty2.Add(new XElement(mobilePhone));

```

Figure 4.56 Some sample code using LINQ to XML to insert XML.

```

A faculty.Element("phone").ReplaceNodes("750-555-0155");
B faculty.SetElement("phone", "750-555-0155");
C faculty.SetElement("office", "MTC-119");
D faculty.SetElement("office", null);

```

Figure 4.57 Some sample code using LINQ to XML to update XML.

Let's take a better look at what is happening behind the scenes when adding an element child to a parent element. When you first create an XElement, it is *unparented*. If you check its **Parent** property, you will get back **null**, which is shown in section **C** in Figure 4.56.

When you use the **Add()** method to add this child element to the parent, LINQ to XML checks to see if the child element is unparented, if so, LINQ to XML *parents* the child element by setting the child's **Parent** property to the XElement that **Add()** was called on. Section **D** in Figure 4.56 shows this situation. This is a very efficient technique that is extremely important since this is the most common scenario for constructing XML trees.

To add **mobilePhone** to another faculty, such as **faculty2**, refer to the codes shown in section **E** in Figure 4.56. Again, LINQ to XML checks to see if the child element is parented. In this case, the child is already parented. If the child *is* already parented, LINQ to XML *clones* the child element under subsequent parents. This situation can be illustrated by the code shown in section **F** in Figure 4.56.

4.8.2.3 Update XML

To update XML, you can navigate to the XElement whose contents you want to replace, and then use the **ReplaceNodes()** method. For example, if you wanted to change the phone number of the first phone XElement of a **faculty**, you could do the code shown in section **A** in Figure 4.57.

```

A faculty.Element("phone").Remove();
B faculty.Elements("phone").Remove();
C faculties.Element("faculty").Element("office").RemoveNodes();
D faculty.SetElement("phone", null);

```

Figure 4.58 Some sample code using LINQ to XML to delete XML.

The method `SetElement()` is designed to work on simple content. With the `SetElement()`, you can operate on the parent. For example, we could have performed the same update we demonstrated above on the first phone number by using the code that is shown in section **B** in Figure 4.57.

The results would be identical. If there had been no phone numbers, an `XElement` named **“phone”** would have been added under **faculty**. For example, you might want to add an **office** to the **faculty**. If an **office** is already there, you can update it. If it does not exist, you can insert it. This situation is shown in section **C** in Figure 4.57.

Also, if you use `SetElement()` with a value of **null**, the selected `XElement` will be deleted. You can remove the office element completely by using the code shown in section **D** in Figure 4.57. Attributes have a symmetric method called `SetAttribute()`, which has the similar functionality as `SetElement()`.

4.8.2.4 Delete XML

To delete XML elements, navigate to the content you want to delete and call the `Remove()` method. For example, if you want to delete the first phone number for a **faculty**, enter the code shown in section **A** in Figure 4.58.

The `Remove()` method also works over an `IEnumerable`, so you could delete all of the phone numbers for a **faculty** in one call as shown in section **B** in Figure 4.58.

You can also remove all of the content from an `XElement` by using the `RemoveNodes()` method. For example you could remove the content of the first **faculty**’s first **office** with the statement as shown in section **C** in Figure 4.58.

Another way to remove an element is to *set* it to **null** using the `SetElement()` method, which we talked about in the last section, Update XML. An example code is shown in section **D** in Figure 4.58.

4.8.3 Manipulate XML Attributes

There is substantial symmetry between working with `XElement` and `XAttribute` classes. However, in the LINQ to XML class hierarchy, `XElement` and `XAttribute` are quite distinct and do not derive from a common base class. This is because XML attributes are not nodes in the XML tree; they are unordered name/value pairs associated with an XML element. LINQ to XML makes this distinction, but in practice, working with `XAttribute` is quite similar to working with `XElement`. Considering the nature of an XML attribute, where they diverge is understandable.


```
<faculties>
  <faculty>
    <faculty_name>Patrick Tones</faculty_name>
    <phone type="home">750-555-0144</phone>
    <phone type="work">750-555-0145</phone>
  </faculty>
</faculties>
```

Figure 4.59 Sample XML attributes.

```
XElement faculty = new XElement("faculty",
    new XElement("faculty_name", "Patrick Tones"),
    new XElement("phone",
        new XAttribute("type", "home"),
        "750-555-0144" ),
    new XElement("phone",
        new XAttribute("type", "work"),
        "750-555-0145"));
```

Figure 4.60 Sample code to create an XAttribute.

4.8.3.1 Add XML Attributes

Adding an XAttribute is very similar to adding a simple XElement. In the sample XML shown in Figure 4.59, notice that each phone number has a *type* attribute that states whether this is a home, work, or mobile phone number:

You create an XAttribute by using functional construction the same way you would create an XElement with a simple type. To create a **faculty** using functional construction, enter the codes shown in Figure 4.60.

Just as you use the SetElement() method to update, add, or delete elements with simple types, you can do the same using the SetAttribute(XName, object) method on XElement. If the attribute exists, it will be updated. If the attribute does not exist, it will be added. If the value of the **object** is **null**, the attribute will be deleted.

4.8.3.2 Get XML Attributes

The primary method for accessing an XAttribute is by using the Attribute(XName) method on XElement. For example, to use the *type* attribute to obtain the contact's home phone number, one can use the piece of code shown in section **A** in Figure 4.61.

Notice how the Attribute(XName) works similarly to the Element(XName) method. Also, notice that there are identical explicit cast operators, which lets you cast an XAttribute to a variety of simple types.

4.8.3.3 Delete XML Attributes

If you want to delete an attribute you can use the Remove() or SetAttribute(XName, object) method passing **null** as the value of object. For example, to delete the type attribute from the first phone using the Remove() method, use the code shown in section **B** in Figure 4.61.

```

A foreach (p in faculty.Elements("phone"))
    {
        if ((string)p.Attribute("type") == "home")
            Console.WriteLine("Home phone is: " + (string)p);
    }

B faculty.Elements("phone").First().Attribute("type").Remove();
C faculty.Elements("phone").First().SetAttribute("type", null);

```

Figure 4.61 Sample code to get and delete an XAttribut.

Alternatively you can use the `SetAttribute()` method with a **null** argument to perform this deleting operation. An example code is shown in section **C** in Figure 4.61. We have provided a very detailed discussion about the basic components on manipulating XML elements and attributes, now let's go a little deeper on the query XML with LINQ to XML.

4.8.4 Query XML with LINQ to XML

The major differentiator for LINQ to XML and other in-memory XML programming APIs is Language-Integrated Query. LINQ provides a consistent query experience across different data models as well as the ability to mix and match data models within a single query. This section describes how to use LINQ with XML. The following section contains a few examples of using LINQ across data models.

Standard query operators form a complete query language for `IEnumerable<T>`. Standard query operators show up as extension methods on any object that implements `IEnumerable<T>` and can be invoked like any other method. This approach, calling query methods directly, can be referred to as explicit dot notation. In addition to standard query operators there are query expressions for the five common query operators:

- Where
- Select
- SelectMany
- OrderBy
- GroupBy

Query expressions provide an ease-of-use layer on top of the underlying explicit dot notation similar to the way that `foreach` is an ease-of-use mechanism that consists of a call to `GetEnumerator()` and a `while` loop. When working with XML, you will probably find both approaches useful. An orientation of the explicit dot notation will give you the underlying principles behind XML Language-Integrated Query, and help you to understand how query expressions simplify things.

The LINQ to XML integration with Language-Integrated Query is apparent in three ways:

1. Leveraging standard query operators
2. Using XML query extensions
3. Using XML transformation

```

<faculties>
  <!-- contact -->
  <faculty_name>Patrick Tones</faculty_name>
  <phone type="home">750-555-0144</phone>
  <phone type="work">750-555-0145</phone>
  <office>MTC-319</office>
  <title>Associate Professor</title>
  <email>ptones@college.edu</email>
  <!-- contact -->
  <faculty_name>Greg River</faculty_name>
  <office>MTC-330</office>
  <title>Assistant Professor</title>
  <email>griver@college.edu</email>
  <!-- contact -->
  <faculty_name>Scott Money</faculty_name>
  <phone type="home">750-555-0134</phone>
  <phone type="mobile">750-555-0177</phone>
  <office>MTC-335</office>
  <title>Professor</title>
  <email>smoney@college.edu</email>
</faculties>

```

Figure 4.62 Sample code to create an XElement.

The first is common with any other LINQ-enabled data access technology and contributes to a consistent query experience. The last two provide XML-specific query and transform features.

4.8.4.1 Standard Query Operators and XML

LINQ to XML fully leverages standard query operators in a consistent manner, exposing collections that implement the IEnumerable interface. We have provided a very detailed discussion about the Standard Query Operators in Sections 4.1.2, 4.1.3, and 4.1.4. Review those sections for details on how to use standard query operators. In this section we will cover two scenarios that occasionally arise when using standard query operators.

First let's create a XElement with multiple elements that can be queried by using a single Select Standard Query Operator. Enter the code shown in Figure 4.62 to create this sample XElement.

In this XElement, the faculty information is directly created under the root <faculties> element rather than under each separate <faculty> elements. In this way, we flatten out our faculty list and make it simple to be queried.

To use the Standard Query Operator Select to perform the LINQ to XML query, you can use a piece of sample code shown in Figure 4.63. Notice that we used an array initializer to create the sequence of children that will be placed directly under the faculties element.

4.8.4.2 XML Query Extensions

XML-specific query extensions provide you with the query operations you would expect when working in an XML tree data structure. These XML-specific query extensions are

```

new XElement("faculties",
    from c in faculties.Elements("faculty")
    select new object[]
    {
        new XComment("faculty"),
        new XElement("faculty_name", (string)c.Element("faculty_name")),
        c.Elements("phone"),
        new XElement("office", c.Element("office"))
    });

```

Figure 4.63 Sample code to perform the query to an XElement.

analogous to the XPath axes. For example, the Elements() method is equivalent to the XPath * (star) operator. The following sections describe each of the XML-specific query extensions in turn.

The Elements query operator returns the child elements for each XElement in a sequence of XElements (IEnumerable<XElement>). For example, to get the child elements for every faculty in the faculty list, you could do the following:

```

foreach (XElement fi in faculties.Elements("faculty").
    Elements())
{
    Console.WriteLine(fi);
}

```

Note that the two Elements() methods used in this example are different, although they do identical things. The first Elements() is calling the XElement method Elements(), which returns an IEnumerable<XObject>, containing the child elements in the single XElement faculties. The second Elements() method is defined as an extension method on IEnumerable<XObject>. It returns a sequence containing the child elements of every XElement in the list.

If you want all of the children with a particular name, you can use the Elements(XName) overload. A piece of sample codes is shown below:

```

foreach (XElement pi in faculties.Elements("faculty").
    Elements("phone"))
{
    Console.WriteLine(pi);
}

```

This would return all phone numbers related to all children.

4.8.4.3 Using Query Expressions with XML

There is nothing unique in the way that LINQ to XML works with query expressions so we will not repeat information here. The following shows a few simple examples of using query expressions with LINQ to XML.

The query shown in section **A** in Figure 4.64 retrieves all of the offices from the faculties, orders them by faculty_name, and then returns them as **string** (the result of this query is IEnumerable<string>).

The query shown in section **B** in Figure 4.64 retrieves all faculty members from faculty that have the faculty_id that starts from B and have an area code of 750 ordered by the faculty_name. The result of this query is IEnumerable<XElement>.

```

A from f in faculties.Elements("faculty")
    where (string) f.Element("office") == "MTC-3.*"
    orderby (string) f.Element("faculty_name")
    select (string) f.Element("faculty_name");

B from f in faculties.Elements("faculty"), p in f.Elements("phone")
    where (string) f.Element("faculty_id") == "B.*" && p.Value.StartsWith("750")
    orderby (string) f.Element("faculty_name")
    select f;

C from s in students.Elements("student"), average = students.Elements("student").
    Average(x => (int) x.Element("gpa"))
    where (int) s.Element("gpa") > average
    select s;

```

Figure 4.64 Sample code to perform the query using query expressions with XML.

Another example shown in section **C** in Figure 4.64 retrieving the students that have a gpa that is greater than the average gpa.

4.8.4.4 Using XPath and XSLT with LINQ to XML

LINQ to XML supports a set of “bridge classes” that allow it to work with existing capabilities in the System.Xml namespace, including XPath and XSLT. Note that System.Xml supports only the 1.0 version of these specifications in “Orcas.”

Extension methods supporting XPath are enabled by referencing the System.Xml.XPath namespace by adding this namespace typing: using System.Xml.XPath; in the namespace declaration section on the code window of each project.

This brings into scope CreateNavigator overloads to create XPathNavigator objects, XPathEvaluate overloads to evaluate an XPath expression, and XPathSelectElement[s] overloads that work much like SelectSingleNode and XPathSelectNodes methods in the System.Xml DOM API. To use namespace-qualified XPath expressions, it is necessary to pass in a NamespaceResolver object, just as with DOM.

For example, to display all elements with the name “phone”, the following codes can be developed:

```

foreach (var phone in faculties.XPathSelectElements("//
    phone"))
{
    Console.WriteLine(phone);
}

```

Likewise, XSLT is enabled by referencing the System.Xml.Xsl namespace by typing: using System.Xml.Xsl in the namespace declaration section on the code window of each project. That allows you to create an XPathNavigator using the XDocument CreateNavigator() method and pass it to the Transform() method.

4.8.4.5 Mixing XML and Other Data Models

LINQ provides a consistent query experience across different data models via standard query operators and the use of lambda expressions that will be discussed in the next section. It also provides the ability to mix and match LINQ-enabled data models/APIs

```

XElement Faculty Faculties = new XElement("Faculties",
    from f in db.Faculties
    where f.faculty_id == "B*"
    select new XElement("Faculty",
        new XAttribute("facultyName", f.faculty_name),
        new XElement("Office", f.office),
        new XElement("Title", f.title),
        new XElement("Phone", f.phone),
        new XElement("Email", f.email)));
Console.WriteLine(Faculty Faculties);

```

Figure 4.65 Sample code to perform the query using mixing XML.

```

facultyUpdates>
  <facultyUpdate>
    <faculty_id>D55990</faculty_id>
    <phone>750-555-0103</phone>
  </facultyUpdate>
  <facultyUpdate>
    <faculty_id>E23456</faculty_id>
    <phone>750-555-0143</phone>
  </facultyUpdate>
</facultyUpdates>

```

Figure 4.66 Sample XML.

within a single query. This section provides a simple example of two common scenarios that mix relational data with XML, using our CSE_DEPT sample database.

Reading from a Database to XML Figure 4.65 shows a simple example of reading from the CSE_DEPT database (using LINQ to SQL) to retrieve the faculties from the Faculty table, and then transforming them into XML.

Reading XML and Updating a Database You can also read XML and put that information into a database. For this example, assume that you are getting a set of faculty member updates in XML format. For simplicity, the update records contain only the phone number changes.

First, let's create a sample XML, which is shown in Figure 4.66.

To accomplish this update, you query for each facultyUpdate element and call the database to get the corresponding Faculty record. Then, you update the Faculty column with the new phone number. A piece of sample code to fulfill this functionality is shown in Figure 4.67.

At this point, we have finished the discussion about the LINQ to XML. Next we will have a closer look at the C# 3.0 language enhancement for LINQ.

4.9 C# 3.0 LANGUAGE ENHANCEMENT FOR LINQ

C# 3.0 introduces several language extensions that build on C# 2.0 to support the creation and use of higher order, functional style class libraries. The extensions enable construc-

```

foreach (var fi in facultyUpdates.Elements("facultyUpdate"))
{
    Faculty faculty = db.Faculties.
    First(f => f.faculty_id == (string)fi.Element("faculty_id"));
    faculty.Phone = (string)fi.Element("phone");
}
db.SubmitChanges();

```

Figure 4.67 Piece of sample code to read and update database.

tion of compositional APIs that have equal expressive power of query languages in domains such as relational databases and XML.

Compared with C# 2.0, significant enhancements have been added into C# 3.0, and these enhancements are mainly developed to support the Language-Integrated Query. LINQ is a series of language extensions that supports data querying in a type-safe way; it is released with the latest version Visual Studio, Visual Studio.NET 2008. The data to be queried, which we have discussed in the previous sections in this chapter, can take the form of objects (LINQ to Objects), databases (LINQ-enabled ADO.NET, which includes LINQ to SQL, LINQ to DataSet, and LINQ to Entities), XML (LINQ to XML), and so on.

In addition to those general LINQ topics, special improvements on LINQ are made for C# and involved in C# 3.0. The main components of these improvements include:

- Lambda expressions
- Extension methods
- Implicitly typed local variables
- Query expressions

Let's have a detailed discussion of these topics one by one.

4.9.1 Lambda Expressions

Lambda expressions are a language feature that is similar in many ways to anonymous methods. In fact, if lambda expressions had been developed and implemented into the language first, there would have been no need for anonymous methods. The basic idea of using lambda expressions is that you can treat code as data. In the early version C#, such as C# 1.0, it is very common to pass strings, integers, reference types, and so on to methods so that the methods can work on those values. Anonymous methods and lambda expressions extend the range of the values to include code blocks. This concept is common in functional programming.

The syntax of lambda expressions can be expressed as a comma-delimited list of parameters with the lambda operator (\Rightarrow) followed by an expression. For more complicated lambda expressions, a statement block can be followed after the lambda operator. A simple example of lambda expression used in C# looks like:

```
x => y
```

where *x* on the left side of the lambda operator is the input parameter and the *y* on the right side of the lambda operator is the output parameter. The data type of both the input and the output parameters should be explicitly indicated by the delegate. Therefore the lambda expressions are closely related to delegate. This lambda expression can be read as *input x and output y*. The syntax of this kind of simple lambda expressions can be written as:

```
(param1, param2, ..., paramN) => output
```

A parenthesis should be used to cover all input parameters.

For more complicated lambda expressions, a statement block should be adopted. An example of this kind of syntax is shown below:

```
(x, y) => { if (x > y) return x; else return y; }
```

Note that the data type of both the input and the output parameters must be identical with those types defined in the delegate. For example, in the previous sample expression *x => y*, if the input *x* is defined as a string, and the output is defined as an integer by the delegate, the output must be converted to an integer type by using the following lambda expression:

```
x => y.Length
```

where *Length* is a method to convert the input from a string to an integer.

Another example of using lambda expressions to perform LINQ query is:

```
IEnumerable<Faculty> faculty =  
EnumerableExtensions.Where(faculties, f => f.faculty_name ==  
"Ying Bai");
```

Here the SQO method *Where()* is used as a filter in this query. The input is an object with a type of *faculties*, and the output is a string variable. The compiler is able to infer that “*f*” refers to a faculty because the first parameter of the *Where()* method is *IEnumerable<Faculty>*, such that *T* must, in fact, be **Faculty**. Using this knowledge, the compiler also verifies that **Faculty** has a *faculty_name* member. Finally, there is no return key word specified. In the syntactic form, the return member is omitted but this is merely syntactic convenience. The result of the expression is still considered to be the return value.

Lambda expressions also support a more verbose syntax that allows you to specify the types explicitly, as well as execute multiple statements. An example of this kind of syntax is:

```
return EnumerableExtensions.Where(faculties, (Faculty f) =>  
{string id =faculty_id; return f.faculty_id = id;});
```

Here the *EnumerableExtensions* class is used to allow us to access and use the static method *Where()* since all SQO methods are static methods defined in either *Enumerable* or *Queryable* classes. As you know, a static method is defined as a class method and can be accessed and used by each class in which that method is defined. Is that possible for us to access a static method from an instance of that class? Generally, this will be considered as a stupid question since that is impossible. Is there any way to make it pos-

sible? The answer is maybe. To get that question answered correctly, let's go to the next topic.

4.9.2 Extension Methods

Regularly, static methods can only be accessed and used by classes in which those static methods are defined. For example, all SQO methods, as we discussed in Sections 4.1.3 and 4.1.4, are static methods defined in either Enumerable or Queryable classes and can be accessed by those classes directly. But those static methods cannot be accessed by any instance of those classes. Let's use an example to make this story clear.

Figure 4.68 shows a piece of code that defines both class and instance methods.

In this example, the method `ConvertToUpper()` is an instance method and `ConvertToLower()` is a class method. To call these methods, different calling strategy must be utilized. To call and execute the instance method `ConvertToUpper()`, one must first create a new instance of the class `Conversion`, and then call that method. To call and execute the class method `ConvertToLower()`, one can directly call it with the class name prefixed in front of that method. Figure 4.69 shows a piece of code to call these two methods.

In some situations, the query would become very complicated if one wants to call those static methods from any instance of those classes. To solve this complex issue, extension methods are developed to simplify the query structures and syntax.

To declare an extension method from existing static method, just add the keyword **this** to the first argument of that static method. For example, to make the class method

```
public static class Conversion
{
    public string ConvertToUpper(string input)
    {
        return input.ToUpper();
    }
    public static string ConvertToLower(string input)
    {
        return input.ToLower();
    }
}
```

Figure 4.68 Example of defining class and instance method.

```
// call instance method ConvertToUpper.
// first create a new instance of the class Conversion
Conversion conv = new Conversion();
string instResult = conv.ConvertToUpper("conversion");
// call class method ConvertToLower.
string classResult = Conversion.ConvertToLower("CONVERSION");
```

Figure 4.69 Example of calling class and instance method.

```

public static class Conversion
{
    // declare the class method ConvertToLower to extension method.
    public static string ConvertToLower(this string input)
    {
        return input.ToLower();
    }
}

```

Figure 4.70 Declare the class method ConvertToLower to extension method.

```

public static class Main()
{
    // declare an anonymous type variable.
    faculty = new { faculty_id = "B78880", faculty_name = "Ying Bai" };
    Console.WriteLine("faculty information {0}, {1}", faculty.faculty_id + ". " + faculty.faculty_name);
}

```

Figure 4.71 Declare an anonymous type variable.

ConvertToLower() an extension method, add the keyword **this** to the first argument of that method, as shown in Figure 4.70.

Now the class method ConvertToLower() has been converted to an extension method and can be accessed by any instance of the class Conversion.

The extension methods have the following important properties:

1. The extension method will work as an instance method of any object with the same type as the extension method's first argument's data type.
2. The extension methods can only be declared in static classes.
3. Both the class and the extension method are prefixed by the keyword **static**.

Refer to Figure 4.70. The extension method ConvertToLower() has a data type of string since the first argument's type is string. This method is declared in a static class Conversion, and both class and this method are prefixed by the keyword **static**.

4.9.3 Implicitly Typed Local Variables

In LINQ query, there's another language feature known as implicitly typed local variables (or *var* for short) that instructs the compiler to infer the type of a local variable. As you know, with the addition of anonymous types to C#, a new problem becomes a main concern, which is that if a variable is being instantiated that is an unnamed type, as in an anonymous type, what type variable would you assign it to? LINQ queries belong to strongly typed queries with two popular types: `IEnumerable<T>` and `IQueryable<T>`, as we discussed at the beginning of this chapter. Figure 4.71 shows an example of this kind of variable with an anonymous type.

A compiling error will be encountered when this piece of code is compiled since the data type of the variable `faculty` is not indicated. In C# 3.0 language enhancement for

```

public static class Main()
{
    // declare an anonymous type variable.
    var faculty = new { faculty_id = "B78880", faculty_name = "Ying Bai" };
    Console.WriteLine("faculty information {0}, {1}", faculty.faculty_id + ". " + faculty.faculty_name);
}

```

Figure 4.72 Declare an anonymous type variable using implicitly typed local variable.

```

Faculty faculty = new Faculty();
faculty.faculty_id = "B78880";
faculty.faculty_name = "Ying Bai";
faculty.office = "MTC-211";
faculty.title = "Associate Professor";

```

Figure 4.73 Example of using the object initializer.

LINQ, a new terminology, implicitly typed local variable *var*, is developed to solve this kind of anonymous type problem. Refer to Figure 4.72, where the code written in Figure 4.71 is rewritten.

This time there would be no error if you compile this piece of code since the keyword *var* informs the compiler to implicitly infer the variable type from the variable's initializer. In this example, the initializer for this implicitly typed variable *faculty* is a string collection. This means that all implicitly typed local variables are statically type checked at the compile time, therefore an initializer is required to allow the compiler to implicitly infer the type from it.

The implicitly typed local variables mean that those variables are just local within a method, for example, the *faculty* is valid only inside the *main()* method in the previous example. It is impossible for them to escape the boundaries of a method, property, indexer, or other block because the type cannot be explicitly stated, and *var* is not legal for fields or parameter types.

Another important terminology applied in C# 3.0 language enhancement for LINQ is the object initializers. Object initializers basically allow the assignment of multiple properties or fields in a single expression. For example, a common pattern for object creation is shown in Figure 4.73.

In this example, there is no constructor of *Faculty* that takes a *faculty_id*, name, office, and title; however, there are four properties, *faculty_id*, *faculty_name*, *office*, and *title*, which can be set once an instance *faculty* is created. Object initializers allow to create a new instance with all necessary initializations being performed at the same time as the instantiation process.

4.9.4 Query Expressions

To perform any kind of LINQ query, such as LINQ to Objects, LINQ to ADO.NET, or LINQ to XML, a valid query expression is needed. The query expressions implemented in C# 3.0 have a syntax that is closer to SQL statements and are composed of some clauses. One of the most popular query expressions is the **foreach** statement. As this

```

var query_variable = from [identifier] in [data source]
                    let [expression]
                    where [boolean expression]
                    order by [[expression](ascending/descending)], [optionally repeat]
                    select [expression]
                    group [expression] by [expression] into [expression]

foreach (var range_variable in query_variable)
{
    //pick up or retrieve back each element from the range_variable...
}

```

Figure 4.74 Typical syntax of query expression.

foreach is executed, the compiler converts it into a loop with calls to methods such as GetEnumerator() and MoveNext(). The main advantage of using the foreach loop to perform the query is that it provides a significant simplicity in enumerating through arrays, sequences, and collections and return the terminal results in an easy way. A typical syntax of query expression is shown in Figure 4.74.

Generally, a query expression is composed of two blocks. The top block in Figure 4.74 is the from-clause block and the bottom block is the query-body block. The from-clause block only takes charge of the data query information (no query results), but the query-body block performs the real query and contains the real query results.

Referring to syntax represented in Figure 4.74, the following components should be included in a query expression:

- A query variable must be defined first in either explicitly (IEnumerable<T>) or implicitly (var).
- A query expression can be represented in either query syntax or method syntax.
- A query expression must start with a **from** clause, and must end with a **select** or **group** clause. Between the first **from** clause and the last **select** or **group** clause, it can contain one or more of these optional clauses: **where**, **orderby**, **join**, **let**, and even additional **from** clauses.

In all LINQ queries (including LINQ to DataSet), all of clauses will be converted to the associated SQO methods, such as From(), Where(), OrderBy(), Join(), Let(), and Select(), as the queries are compiled. Refer to Table 4.1 to get the most often used Standard Query Operators and their definitions.

In LINQ, a query variable is always strongly typed, and it can be any variable that stores a query instead of the results of a query. More specifically, a query variable is always an enumerable type that will produce a sequence of elements when it is iterated over in a **foreach** loop or a direct call to its method IEnumerator.MoveNext.

A very detailed discussion about the query expression has been provided in Sections 4.5.1.1 and 4.5.1.2 in this Chapter. Refer to those sections to get more details on this topic.

Before we can finish this chapter, a real query example implemented in our project is shown in Figure 4.75.

4.10 CHAPTER SUMMARY

Language-Integrated Query (LINQ), which is built on .NET Frameworks 3.5, is a new technology released with Visual Studio.NET 2008 by Microsoft. LINQ is designed to

```

static void Main()
{
    IEnumerable<Faculty> faculty = db.Faculties.Where(f => f.faculty_id == "D.*",
        f => f.college == "U.*",
        f => f.title == "Associate Professor");

    // Execute the query to produce the results
    foreach (Faculty fi in faculty)
    {
        Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}", f.faculty_name, f.title, f.office, f.phone, f.email);
    }
}

```

Figure 4.75 Real example of query expression.

query general data sources represented in different formats, such as Objects, DataSet, SQL Server database, Entities, and XML. The innovation of LINQ bridges the gap between the world of objects and the world of data.

An introduction to LINQ general programming guide is provided in the first part of this chapter. Some popular interfaces widely used in LINQ, such as IEnumerable, IEnumerable<T>, IQueryable, and IQueryable<T>, and Standard Query Operators (SQO) including the deferred and nondeferred SQO, are discussed in that part.

An introduction to LINQ Query is given in the second section in this chapter. Following this introduction, a detailed discussion and analysis about LINQ implemented for different data sources is provided based on the sequence listed below.

1. Architecture and components of LINQ
2. LINQ to Objects
3. LINQ to DataSet
4. LINQ to SQL
5. LINQ to Entities
6. LINQ to XML
7. C# 3.0 language enhancement for LINQ

Both literal introductions and actual examples are provided for each part listed above to give readers not only a general and global picture about LINQ technique applied for different data, but also practical and real feeling about the program codes developed to realize the desired functionalities.

Twelve real projects are provided in this chapter to help readers to understand and follow up on all techniques discussed in this chapter.

After finishing this chapter, readers should be able to:

- Understand the basic architecture and components implemented in LINQ.
- Understand the functionalities of Standard Query Operators.
- Understand general interfaces implemented in LINQ, such as LINQ to Objects, LINQ to DataSet, LINQ to SQL, LINQ to Entities, and LINQ to XML.
- Understand the C# 3.0 language enhancement for LINQ.
- Design and build real applications to apply LINQ queries to perform data actions to all different data sources.

- Develop and build applications to apply C# 3.0 language enhancement for LINQ to perform all different queries to data sources.

Starting with the next chapter, we will concentrate on the database programming with Visual C#.NET using real projects.

HOMEWORK

I. True/False Selections

- ___ 1. LINQ queries are built based on .NET Frameworks 3.5.
- ___ 2. Most popular interfaces used for LINQ queries are IEnumerable, IEnumerable<T>, IQueryable, and IQueryable<T>.
- ___ 3. IEnumerable interface is used to convert data type of data source to IEnumerable<T>, which can be implemented by LINQ queries.
- ___ 4. IEnumerable interface is inherited from the class IQueryable.
- ___ 5. All Standard Query Operator methods are static methods defined in the IEnumerable class.
- ___ 6. IEnumerable and IQueryable interfaces are mainly used for the nongeneric collections supported by the earlier versions of C#, such as C# 1.0 or earlier.
- ___ 7. All LINQ query expressions can only be represented as query syntax.
- ___ 8. All LINQ query expressions will be converted to the Standard Query Operator methods during the compile time by CLR.
- ___ 9. The query variable used in LINQ queries contains both the query information and the returned query results. ___
- ___ 10. LINQ to SQL, LINQ to DataSet, and LINQ to Entities belong to LINQ to ADO.NET.

II. Multiple Choices

1. The difference between the interfaces IEnumerable and IEnumerable<T> is that the former is mainly used for _____, but the latter is used for _____.
 - a. Nongeneric collections, generic collections
 - b. Generic collections, nongeneric collections
 - c. All collections, partial collections
 - d. .NET Frameworks 2.0, .NET Frameworks 3.5
2. The query variable used in LINQ queries contains _____.
 - a. Query information and query results
 - b. Query information
 - c. Query results
 - d. Standard Query Operator
3. All Standard Query Operator (SQO) methods are defined as _____; this means that these methods can be called either as class methods or as instance methods.
 - a. Class methods
 - b. Instance methods
 - c. Variable methods
 - d. Extension methods

4. One of the SQO methods, the AsEnumerable() operator method, is used to convert the data type of the input object from _____ to _____.
 - a. IQueryable<T>, IEnumerable<T>
 - b. IEnumerable<T>, IEnumerable<T>
 - c. Any, IEnumerable<T>
 - d. All of the above
5. LINQ to Objects is used to query any sequences or collections that are either explicitly or implicitly compatible with _____ sequences or _____ collections.
 - a. IQueryable, IQueryable<T>
 - b. IEnumerable, IEnumerable<T>
 - c. Deferred SQO, non-deferred SQO
 - d. Generic, nongeneric
6. LINQ to DataSet is built on the _____ architecture. The codes developed by using that version of ADO.NET will continue to function in a LINQ to DataSet application without modifications.
 - a. ADO.NET 2.0
 - b. ADO.NET 3.0
 - c. ADO.NET 3.5
 - d. ADO.NET 4.0
7. Two popular LINQ to SQL Tools, _____ and _____, are widely used in developing applications of using LINQ to SQL.
 - a. Entity Data Model, Entity Data Model Designer
 - b. IEnumerable, IEnumerable<T>
 - c. SQLMetal, Object Relational Designer
 - d. IQueryable, IQueryable<T>
8. LINQ to SQL query is performed on classes that implement the _____ interface. Since the _____ interface is inherited from the _____ with additional components, therefore the LINQ to SQL queries have additional query operators.
 - a. IEnumerable<T>, IEnumerable<T>, IQueryable<T>
 - b. IEnumerable<T>, IQueryable<T>, IEnumerable<T>
 - c. IQueryable<T>, IEnumerable<T>, IQueryable<T>
 - d. IQueryable<T>, IQueryable<T>, IEnumerable<T>
9. LINQ to Entities queries are performed under the control of the _____ and the _____.
 - a. .NET Frameworks 3.5, ADO.NET 3.5
 - b. ADO.NET 3.5 Entity Framework, ADO.NET 3.5 Entity Framework Tools
 - c. IEnumerable<T>, IQueryable<T>
 - d. Entity Data Model, Entity Data Model Designer
10. To access and implement ADO.NET 3.5 EF and ADO.NET 3.5 EFT, developers need to understand the _____, which is a core of ADO.NET 3.5 EF.
 - a. SQLMetal
 - b. Object Relational Designer
 - c. Generic collections
 - d. Entity Data Model

11. Lambda expressions, which are represented by _____, are a language feature that is similar in many ways to _____ methods.
 - a. =>, Standard Query Operator
 - b. =>, anonymous
 - c. =>, Generic collection
 - d. =>, IQueryable
12. Extension methods are defined as those methods that can be called as either _____ methods or _____ methods.
 - a. Class, instance
 - b. IEnumerable<T>, IQueryable<T>
 - c. Generic, nongeneric
 - d. Static, dynamic
13. In LINQ queries, the data type **var** is used to define a(n) _____, and the real data type of that variable can be inferred by the _____ during the compiling time.
 - a. Generic variable, debugger
 - b. Implicitly typed local variable, compiler
 - c. Nongeneric variable, builder
 - d. IEnumerable<T> variable, loader
14. In LINQ queries, the query expression must start with a _____ clause, and must end with a _____ or _____ clause.
 - a. begin, select, end
 - b. select, where, orderby
 - c. from, select, group
 - d. **query** variable, **range** variable, **foreach** loop
15. The DataContext is a class that is used to establish a _____ between your project and your database. In addition to this role, the DataContext also provides the function to _____ operations of the Standard Query Operators to the SQL statements that can be run in real databases.
 - a. Relationship, perform
 - b. Reference, translate
 - c. Generic collections, transform
 - d. Connection, convert

III. Exercises

1. Explain the architecture and components of LINQ, and illustrate the functionality of these using a block diagram.
2. Explain the execution process of a LINQ query using the foreach statement.
3. Explain the definitions and functionalities of the Standard Query Operator methods.
4. Explain the relationship between the LINQ query expressions and Standard Query Operator methods
5. Explain the definitions and functionalities of IEnumerable, IEnumerable<T>, IQueryable, and IQueryable<T> interfaces.
6. Explain the components and procedure used to perform LINQ to SQL queries.


```
List<string> fruits = new List<string> { "apple", "banana", "mango", "orange",  
                                       "blueberry", "grape", "strawberry" };  
  
var query = from fruit in fruits  
            where fruit.Length < 6  
            select fruit;  
  
foreach (string f in query)  
    Console.WriteLine(f);
```

Figure 4.76

7. A query used for LINQ to Objects, which is represented by a query syntax, is shown in Figure 4.76. Try to convert this query expression to a method syntax.
8. Illustrate the procedure of creating each entity class for each data table in our sample database CSE_DEPT.mdf by using the Object Relational Designer, and adding a connection to the selected database using the DataContext class or the derived class from the DataContext class.
9. Explain the difference between the class method and the instance method, and try to illustrate the functionality of an extension method and how to build an extension method by using an example.
10. List three steps of performing the LINQ to DataSet queries.

Chapter 5

Data Selection Query with Visual C#.NET

Compared to Visual Studio 2005, Visual Studio 2008 adds more new components to simplify data accessing, inserting, and updating functionalities for database development and applications. First of all, Visual Studio 2005 was built based on the .NET Framework 2.0, but Visual Studio 2008 is based on .NET Framework 3.5. Quite a number of new features such as Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF), and Language Integrated Query (LINQ) have been added to Visual Studio 2008. In addition to these features, Visual Studio 2008 also adds design time tools and more server controls for richer User Interfaces (UIs) and better communication between the client-side code and the server. These new components and features are very helpful and the runtime features have been available in Community Technology Previews (CTPs) for a while. Of all those features, one of the most important features added by Visual Studio 2008 is the LINQ for the database access and data source applications. Because of that, Visual Studio.NET 2008 greatly reduces the programming load and the number of query program codes to provide significant assistance to people who are new to database programming with Visual Studio.

Starting from Visual Studio 2005, Microsoft provides quite a few design tools and wizards to help users build and develop database programming easily and efficiently. The most popular design tools and wizards are:

- Data Components in the Toolbox Window
- Wizards in the Data Source Window

These design tools and wizards are still implemented in Visual Studio 2008, and they can be accessed and used by any .NET-compatible programming language such as Visual C++, Visual Basic, Visual J#, and Visual C#. The Toolbox window in Visual Studio 2008 contains data components that enable you to quickly and easily build simple database applications without needing to touch very complicated coding issues. Combine these data components with wizards, which are located in the Data Source wizard and related to ADO.NET, and one can easily develop binding relationships between the data source and controls on the Visual C# windows form object. Furthermore one can build simple

Visual C# project to navigate, scan, retrieve, and manipulate data stored in the data source with a few lines of codes.

This chapter is divided to two parts: Part I provides a detailed description and discussion on how to use Visual Studio 2008 tools and wizards to build simple but efficient database applications without touching complicated coding in the Visual C# environment. In Part II, a more in-depth discussion on how to develop advanced database applications while using runtime objects is presented. More complicated coding technology is provided in this part. The data query using the LINQ technology is discussed in both parts with project examples. Five real examples are provided in detail to enable readers to have a clear picture of the development of professional database applications in simple and efficient ways. This chapter concentrates only on the data query applications.

In this chapter, you will:

- Learn and understand the most useful tools and wizards used in developing data query applications.
- Learn and understand how to connect a database with different components provided in data providers, and configure this connection with wizards.
- Learn and understand how to use BindingSource object to display database tables' contents using DataGridView.
- Learn and understand how to bind a DataSet (data source) to various controls in the windows form object.
- Learn and understand how to configure and edit DataAdapter to build special queries.
- Learn and understand how to retrieve data using the LINQ technology from the data source to simplify and improve the efficiency of the data querying.
- Build and execute simple dynamic data query commands to retrieve desired data.

To successfully complete this chapter, you need to understand topics such as the fundamentals of databases, which was introduced in Chapter 2, and ADO.NET, which was discussed in Chapter 3. Also three sample databases developed in Chapter 2, which are CSE_DEPT.accdb, CSE_DEPT.mdf, and CSE_DEPT of the Oracle Database 10g, will be used through this chapter.

PART I DATA QUERY WITH VISUAL STUDIO DESIGN TOOLS AND WIZARDS

Before we consider the Visual Studio 2008 tools and wizards, a preview of a completed sample database application is necessary. This preview can give readers a feeling of how a database application works and what it can do. The database used for this project is Access 2007.

5.1 COMPLETED SAMPLE DATABASE APPLICATION EXAMPLE

This sample application is composed of five forms, titled LogIn, Selection, Faculty, Student, and Course forms. This example is designed to map the Computer Science and

Table 5.1 Relationship between the Form and Data Table

Visual C# Form	Tables in Sample Database
LogIn	LogIn
Faculty	Faculty
Course	Course
Student	Student, StudentCourse

**Figure 5.1** LogIn form.

Engineering (CSE) Department in a university and allow users to scan and browse all information about the department, including the faculty, courses taught by selected faculty, students, and courses taken by the associated student.

Each form, except the Selection form, is associated with one or two data tables in a sample database CSE_DEPT.acddb, which was developed in Chapter 2. The relationship between the form and tables is shown in Table 5.1.

Controls on each form are bound to the associated fields in certain data tables located in the CSE_DEPT database. As the project runs, a data query will be executed via a dynamic SQL statement that is built during the configuration of each TableAdapter in the Data Source wizard. The retrieved data will be displayed on the associated controls that have been bound to those data fields.

Go to the accompanying site at ftp://ftp.wiley.com/public/sci_tech_med_/practical_database and browse to the folder DBProjects\Chapter 5 to find the project, SampleWizards Solution\SampleWizards Project, to locate an executable file: SampleWizards Project.exe. Double-click on this file to run it.

As the project runs, a login form will be displayed to ask users to enter username and password, which shown in Figure 5.1. Enter **ybai** and **reback** as username and password. Then click on the LogIn button to call the LogIn TableAdapter to execute a query to pick up a record that matches the username and password entered by the user from the LogIn table located in the CSE_DEPT database.

If a matched record is found based on the username and password, this means that the login is successful and the next window form, Selection, will be displayed to allow the user to select and retrieve the desired information for the selected faculty, course, or student, which is shown in Figure 5.2.

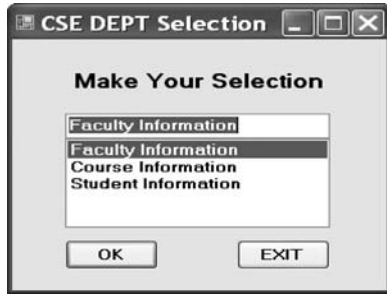


Figure 5.2 Selection form.



Figure 5.3 Faculty form.

Select the default information—Faculty Information—by clicking on the OK button, and the Faculty form appears as shown in Figure 5.3.

The faculty information query is controlled by two ComboBox controls, Faculty Name and Query Method, and two Button controls, Select and Back. By using the Faculty Name ComboBox control, the user can select the desired faculty name to retrieve back all related information. By using the Query Method ComboBox, one can select the desired query method, either the TableAdapter or the LINQ method. All faculty names in the CSE department are listed in a combobox control on the form. To query all information for the selected faculty, click on the Select button to execute a prebuilt dynamic query. All information of the selected faculty, which is stored in the Faculty table in the database, will be fetched from the database and reflected on five label controls in the Faculty form, as shown in Figure 5.3.

The faculty photo will also be displayed in a PictureBox control in the form.

The Back button is used to return to the Selection form to enable users to make other selections to obtain the associated information.

Click on the Back button to return to the Selection form, and then select the Course Information item to open the Course form. Select the desired faculty name from the ComboBox control, and click on the Select button to retrieve all courses that are represented by the related course ID and taught by this faculty. All retrieved courses are displayed in the Course ListBox, as shown in Figure 5.4.

Figure 5.4 Course form.

Note that when you select the specified course ID by clicking on it from the Course list, all information related to that selected course—such as the course title, course schedule, classroom, credits, and course enrollment—will be reflected on each associated textbox control under the Course Information frame control.

Two query methods are available to this query, TableAdapter or LINQ. One can select any method by clicking on it from the Query Method ComboBox control.

Click on the Back button to return to the Selection form and select the Student Information to open the Student form. You can continue to work on this form to see what will happen to this form.

In the following sections, we will show how to design and build this demo project step by step by using Visual C# 2008 and SQL Server 2005 database. It is very easy to develop a similar project using the different database such as the Microsoft Access and Oracle. The only thing you need to do is to select the different Data Source when you connect your project to the database you desired.

5.2 VISUAL STUDIO.NET 2008 DESIGN TOOLS AND WIZARDS

When developing and building a Windows application that needs to interface to a database, a powerful and simple way is to use design tools and wizards provided by Visual Studio. The size of the coding process can be significantly reduced, and the procedures can also be greatly simplified. Now let's first take a look at the components in the Toolbox window.

5.2.1 Data Design Tools in Toolbox Window

Each database-related Windows application contains three components that can be used to develop a database application using the data controls in the Toolbox: DataSet,

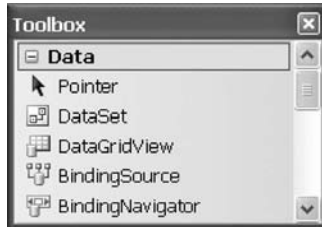


Figure 5.5 Data components in Toolbox window.

BindingSource, and TableAdapter. Two other useful components are the DataGridView and the BindingNavigator. All of these components, except the TableAdapter, are located in the Toolbox window as shown in Figure 5.5.

Compared with Visual Studio 2003, in which only three components—DataConnection, DataAdapter, and DataSet—were used to perform data operations for a data-driven Visual C# application, Visual Studio 2008 made some modifications.

5.2.1.1 DataSet

A DataSet object can be considered as a container, and it is used to hold data from one or more data tables. It maintains the data as a group of data tables with optional relationships defined between those tables. The definition of the DataSet class is a generic idea, which means that it is not tied to any specific type of database. Data can be loaded into a DataSet by using a TableAdapter from many different databases such as Microsoft Access, Microsoft SQL Server, Oracle, Microsoft Exchange, Microsoft Active Directory, or any OLE DB or ODBC-compliant database when your application begins to run or the Form_Load() method is called if one used an DataGridView object.

Although not tied to any specific database, the DataSet class is designed to contain relational tabular data as one would find in a relational database. Each table included in the DataSet is represented in the DataSet as a DataTable. The DataTable can be considered as a direct mapping to the real table in the database. For example, the LogIn data table, LogInDataTable, can be mapped to the real table LogIn in the CSE_DEPT database. The relationship between any table is realized in the DataSet as a DataRelation object. The DataRelation object provides the information that relates a child table to a parent table via a foreign key. A DataSet can hold any number of tables with any number of relationships defined between tables. From this point of view, a DataSet can be considered as a mini-database engine. It can contain all information on tables it holds such as the column name and data type, all relationships between tables, and, more important, it contains most management functionalities of the tables such as browse, select, insert, update, and delete data from tables.

A DataSet is a container and it keeps its data or tables in memory as XML files. In Visual Studio.NET 2003, when one wanted to edit the structure of a DataSet, one had to edit an XML Schema or XSD file. Although there is a visual designer, the terminology and user interface were not consistent with a DataSet and its constituent objects.

With Visual Studio 2008, one can easily edit the structure of a DataSet and make any changes to the structure of that DataSet by using the Dataset Designer in the Data Source window. More important, one can graphically manipulate the tables and queries

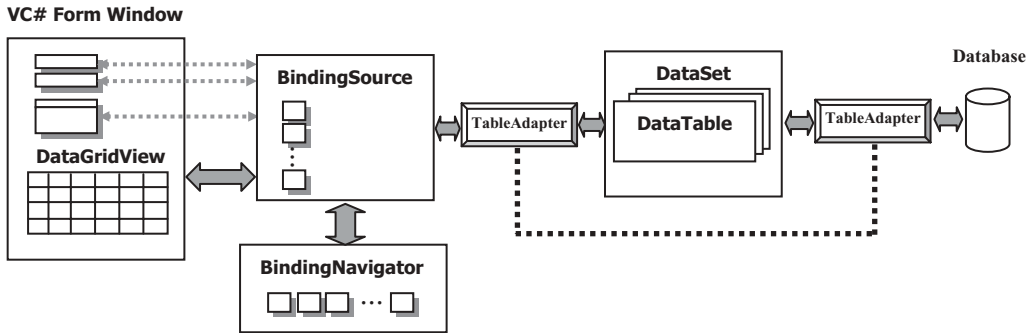


Figure 5.6 Relationship between data components.

in a manner more directly tied to the DataSet rather than having to deal with an XML Schema (XSD).

Therefore, the DataSet object is a very powerful component that can contain multiple data tables with all information related to those tables. By using this object, one can easily browse, access, and manipulate data stored in it. We will explore this component in more detail in the following sections when a real project is built.

When you build a data-driven project and set up a connection between your C# project and a database using the ADO.NET, the DataTables in the DataSet can be populated with data from your database by using data query methods or the Fill() method. From this point of view, you can consider the DataSet as a *data source*, and it contains all mapped data from the database you connected to your project.

Refer to Figure 5.6 for a global picture of the DataSet and other components in the Toolbox window to obtain more detailed ideas on this subject.

5.2.1.2 DataGridView

The next useful data component defined in the Toolbox window is the DataGridView. Like its name, you can consider the DataGridView as a view container, and it can be used to bind data from your database and display the data in a tabular or a grid format. You can use the DataGridView control to show read-only views of a small amount of data, or you can scale it to show editable views of very large sets of data. The DataGridView control provides many properties that enable you to customize the appearance of the view and properties that allow you to modify the column headers and the data displayed in the grid format. You can also easily customize the appearance of the DataGridView control by choosing among different properties. Many types of data stores can be used as a database, or the DataGridView control can operate with no data source bound to it.

By default, a DataGridView control has the following properties:

- Automatically displays column headers and row headers that remain visible as users scroll the table vertically.
- Has a row header that contains a selection indicator for the current row.
- Has a selection rectangle in the first cell.
- Has columns that can be automatically resized when the user double-clicks on the column dividers.

- Automatically supports visual styles on Windows XP and the Windows Server 2003 family when the `EnableVisualStyles` method is called from the application's Main method.

Refer to Figure 5.6 to get a relationship between the `DataGridView` and other data components. A more detailed description on how to use the `DataGridView` control to bind and display data in Visual C# will be provided in Section 5.5.

5.2.1.3 *BindingSource*

The `BindingSource` component has two functionalities. First, it provides a layer of indirection when binding the controls on a form in the data source. This is accomplished by binding the `BindingSource` component to your data source, and then binding the controls on your form to the `BindingSource` component. All further interactions with the data, including navigating, sorting, filtering, and updating, are accomplished with calls to the `BindingSource` component. Second, the `BindingSource` component can act as a strongly typed data source. Adding a type to the `BindingSource` component with the `Add` method creates a list of that type.

Basically, the `BindingSource` control works as a bridge to connect the data bound controls on your Visual C# forms with your data source (`DataSet`). The `BindingSource` control can also be considered as a container object that holds all mapped data from the data source. As a data-driven project runs, the `DataSet` will be filled with data from the database by using a `TableAdapter`. Also the `BindingSource` control will create a set of data that are mapped to those filled data in the `DataSet`. The `BindingSource` control can hold this set of mapped data and create a one-to-one connection between the `DataSet` and the `BindingSource`. This connection is very useful when you perform data binding between controls on the Visual C# form and data in the `DataSet`. Basically, you set up a connection between your controls on the Visual C# form and those mapped data in the `BindingSource` object. As your project runs and the data are needed to be reflected on the associated controls, a request to `BindingSource` is issued and the `BindingSource` control will control the data accessing to the data source (`DataSet`) and data updating in those controls. For instance, the `DataGridView` control will send a request to the `BindingSource` control when a column sorting action is performed, and the latter will communicate with the data source to complete this sorting.

When performing a data binding in Visual Studio, you need to bind the data referenced by the `BindingSource` control to the `DataSource` property of your controls on the forms.

5.2.1.4 *BindingNavigator*

The `BindingNavigator` control allows the user to scan and browse all records stored in the data source (`DataSet`) one by one in sequence. The `BindingNavigator` component provides a standard UI with buttons and arrows to enable users to navigate to the first and the previous records as well as the next and the last records in the data source. It also provides textbox controls to display how many records existed in the current data table and the current displayed record's index.

As shown in Figure 5.6, the `BindingNavigator` is also bound to the `BindingSource` component as other components are. When the user clicks on either the Previous or the Next button on the `BindingNavigator` UI, a request is sent to the `BindingSource` for the

previous or the next record, and, in turn, this request is sent to the data source for picking up the desired data.

5.2.1.5 *TableAdapter*

From Figure 5.6, one can find that a *TableAdapter* is equivalent to an adapter, and it just works as a connection media between the database and *DataSet* and between the *BindingSource* and the *DataSet*. This means that the *TableAdapter* has double functionalities when it works as different roles for the different purposes. For example, as you develop your data-driven applications using the design tools, the data in the database will be populated to the mapped tables in the *DataSet* using the *TableAdapter*'s *Fill()* method. The *TableAdapter* also works as an adapter to coordinate the data operations between the *BindingSource* and the *DataSet* when the data-bound controls in Visual C# need to be filled or updated.

Prior to Visual Studio 2005, the Data Adapter was the only link between the *DataSet* and the database. If a change is needed to the data in the *DataSet*, you need to use a different Data Adapter for each table in the *DataSet* and have to call the Update method of each Data Adapter.

The *TableAdapter* was introduced in Visual Studio 2005 and you cannot find this component in the Toolbox window. The *TableAdapter* belongs to the designer-generated component that connects your *DataSet* objects with their underlying databases, and it will be created automatically when you add and configure new data sources via design tools such as the Data Source Configuration Wizard.

The *TableAdapter* is similar to the *DataAdapter* in that both components can handle the data operations between *DataSet* and the database, but the *TableAdapter* can contain multiple queries to support multiple tables from the database, allowing one *TableAdapter* to perform multiple queries to your *DataSet*. Another important difference between the *TableAdapter* and the Data Adapter is that each *TableAdapter* is a unique class that is automatically generated by Visual Studio to work with only the fields you have selected for a specific database object.

The *TableAdapter* class contains queries used to select data from your database. Also it contains different methods to allow users to fill the *DataSet* with some dynamic parameters in your project with data from the database. You can also use the *TableAdapter* to build different SQL statements such as Insert, Update, and Delete based on the different data operations. A more detailed exploration and implementation of *TableAdapter* with a real example will be provided in the following sections.

5.2.2 Data Design Wizards in Data Source Window

Starting with Visual Studio 2005, two new Integrated Development Environment (IDE) features, the Data Sources Window and the Data Source Configuration Wizard, are added to assist you in setting up data access by using the new classes, such as *DataConnector* and *TableAdapter*.

The Data Sources window is used to display the data sources or available databases in your project. You can use the Data Sources window to directly create a user interface (consisting of data-bound controls) by dragging items from the Data Sources window

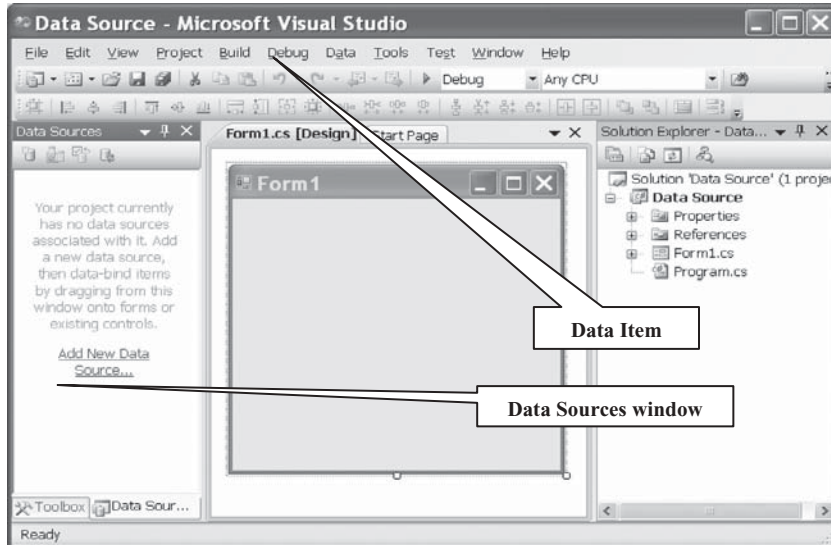


Figure 5.7 Data Sources window.

onto Visual C# forms in your project. Each item inside the Data Sources window has a drop-down control list where you can select the type of control to create prior to dragging it onto a form. You can also customize the control list with additional controls, such as controls that you have created. A more detailed description on how to use the Data Sources window to develop a data-driven project is provided in Section 5.4.

5.2.2.1 Add New Data Source

The first time you create a new data-driven application project in the Visual C# 2008 environment, there is no data source that has been added to your project, and, therefore, the Data Source window is blank with no data source in there. For example, you can create a new Visual C# 2008 Windows application by selecting File|New| Project menu items and select the DataSource as the project name. After this new project is created and opened, you can find the Data Sources window by clicking on the Data menu item from the menu bar, which is shown in Figure 5.7.

To open the Data Sources window, click the Data>Show Data Sources item. Because you have no previous database connected to this new project, the opened Data Sources window is blank. To add a new data source or database to this new project, you can click on the Add New Data Source link from the Data Sources window.

Once you click on the Add New Data Source link from the Data Sources window to add a new data source, the Data Source Configuration Wizard will be displayed. You need to use this wizard to select the database you want connected to your new project.

5.2.2.2 Data Source Configuration Wizard

The opened Data Source Configuration Wizard is shown in Figure 5.8. By using the Data Source Configuration Wizard, you can select the data source or database you want

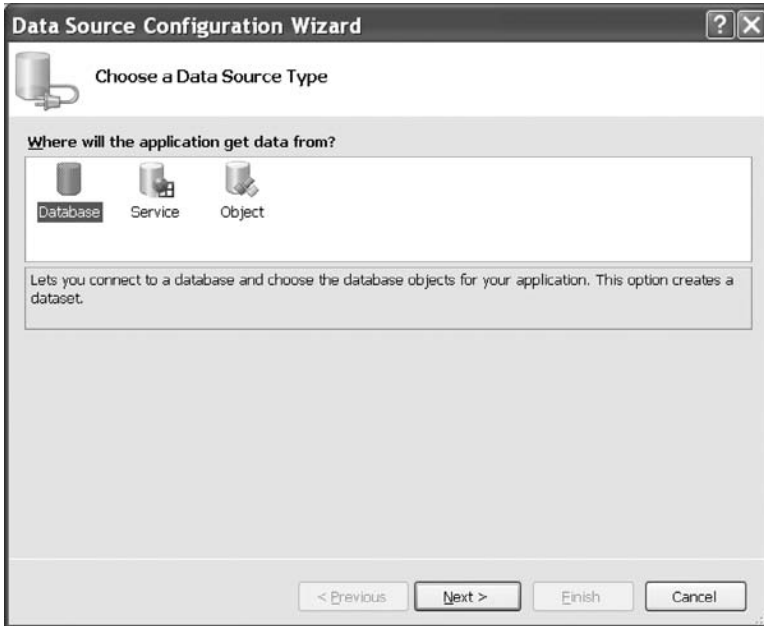


Figure 5.8 Data Source Configuration Wizard.

connected to your new project. The Data Source Configuration Wizard supports three types of data sources. The first option, Database, allows you to select a data source from a database server on your local computer or on a network server. The examples for this kind of data sources are Microsoft Access 2007, SQL Server 2005 Express, SQL Server 2005, or Oracle Database 10g XE. This option also allows you to choose either an .MDF SQL Server database file or a Microsoft Access .MDB or .ACCDB file. The difference between a SQL Server database and a SQL Server database file is that the former is a complete database that integrates the database management system with data tables to form a body or a package, but the latter is only a database file. The second option, Web Service, enables you to select a data source that is located at a Web service. The third option, Object, allows you to bind your user interface to one of your own database classes.

Click on the Next button after you have selected your desired source, and the next step in the Data Source Configuration Wizard allows you to either select an existing data connection or create a new connection to your data source, which is shown in Figure 5.9.

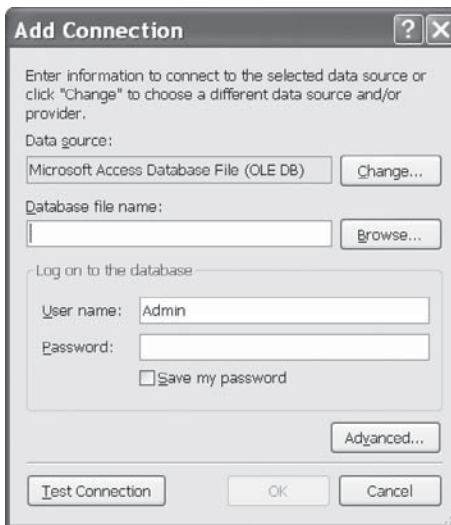
When you run this wizard for the first time, there is no preexisting database connections available, but on subsequent uses of the wizard you can reuse previously created connections. To make a new connection, click on the New Connection button, and the Add Connection dialog is displayed, which is shown in Figure 5.10a.

You can select different types of the data source by clicking on the Change button. The Change Data Source dialog is displayed as you do that, which is shown in Figure 5.10b. Six popular data sources can be chosen based on your application:

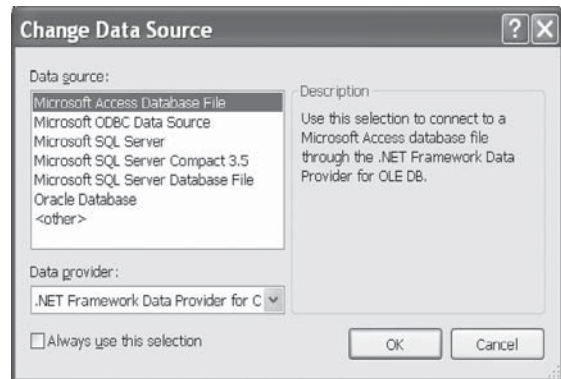
1. Microsoft Access Database File
2. Microsoft ODBC Data Source
3. Microsoft SQL Server



Figure 5.9 Next step in the Data Source Configuration Wizard.



(a)



(b)

Figure 5.10 Add Connection and Change Data Source dialogs.

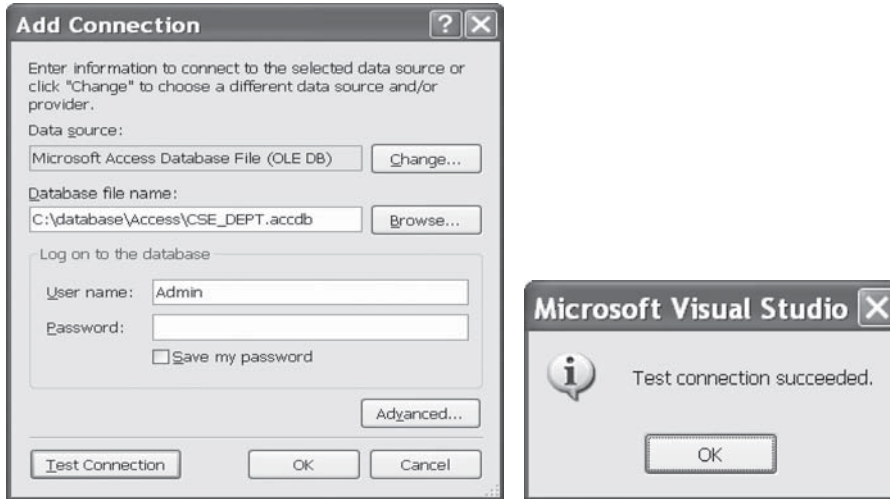


Figure 5.11 Add Connection Dialog and the Test Connection MessageBox.

4. Microsoft SQL Server Compact 3.5
5. Microsoft SQL Server Database File
6. Oracle Database.

The second option is to allow users to select any kind data source that is compatible with a Microsoft ODBC data source. The fifth option is for users who select a SQL Server 2005 Express as the data source.

For example, if you want to connect your new project with a Microsoft Access database named CSE_DEPT.accdb, you need to keep the default Data Source selection, Microsoft Access Database File selected, and then click on the OK button to return to the Add Connection dialog. Click on the Browse button to locate and select the database file CSE_DEPT.accdb. You can click on the Test Connection button to test your connection. A *Test connection succeeded* message will be displayed if your connection is correct, which is shown in Figure 5.11. Click on the OK button to close the Test Connection MessageBox.

The next step in this wizard allows you to save the connection string to the application configuration file named app.config in your new Visual C# 2008 project. You can save this connection string for your further usage if you want to use the same connection again for your other applications later.

When you click on the OK and then the Next button to continue to the next step, a message box will be displayed to ask you if you want to save this data source into your new project, which is shown in Figure 5.12.

The advantage of saving a data source into your project is that you can integrate your project with the data source together to make a complete application. In this way, you are free to worry about any connection problem between your project and your data source, and they are one body and easy to be portable. The disadvantage is that the size of your project will be increased, and more memory space is needed to save your application.

The next configuration step, which is shown in Figure 5.13, allows you to select the database objects for this data source. Although you can select any number of tables,

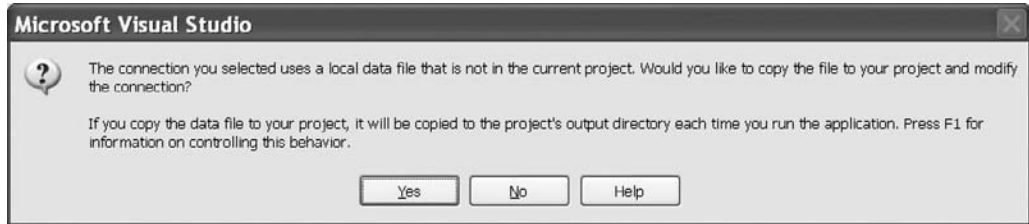


Figure 5.12 Message to ask you to save the data source.



Figure 5.13 Select database objects in the configuration wizard.

views, and functions, it is highly recommended to select all tables and views. In this way, you can access any table and view any data in all tables later.

When you finish selecting your database objects, all selected objects should have been added into your new instance of your DataSet class. In this example, it is CSE_DEPTDataSet located at the bottom of this dialog at the DataSet name: box shown in Figure 5.13. Basically, the data in all tables in your database CSE_DEPT.accdb should have been copied to those mapped tables in your DataSet object CSE_DEPTDataSet. Click on the Finish Button to complete this Data Source Configuration process. You can review any data from any data table in the DataSet by using the Preview Data button later, and the wizard will build your SELECT statements for you automatically.

An important issue is that as you finished this Data Source Configuration and close this Wizard, the connection you set between your application and your database is closed. You need to use a valid data query or data manipulation method such as the Fill() to

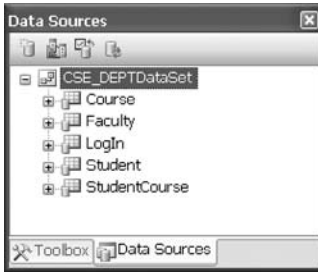


Figure 5.14 Added data source.

reopen this connection, if you want to perform any data action between your application and your database later as your project runs. This technology is called Disconnection mode and is widely implemented in data-driven applications today.

After the Data Source Configuration is finished, a new data source is added into your project. Basically it is added into the Data Source window, which is shown in Figure 5.14. The data source added into your project is basically a DataSet object that contains all data tables that are mappings to those tables in your real database. As shown in Figure 5.14, the Data Source window displays the data source and tables as a tree view, and each table is connected to this tree via a node. If you click on a node (represented by a small + symbol surrounded with a box), all columns in the selected table will be displayed.

Even after the data source is added into your project, the story is not finished and you still have some controllability over this data source. This means that you can still make some modifications to the data source, that is, modifications to the tables and data source related methods. To do this job, you need to know something about another component, DataSet Designer, which is also located in the Data Source window.

5.2.2.3 DataSet Designer

The DataSet Designer is a group of visual tools used to create and edit a typed DataSet and the individual items that make up that DataSet. The DataSet Designer provides visual representations of the objects contained in the DataSet. By using the DataSet Designer, you can create and modify TableAdapters, TableAdapter Queries, DataTables, DataColumnns, and DataRelations.

To open the DataSet Designer, right-click on any place inside the Data Source window; then select the Edit DataSet with Designer. A sample DataSet Designer is shown in Figure 5.15.

In this sample database, we have five tables: LogIn, Faculty, Course, Student, and StudentCourse. To edit any item, just right-click on the associated component for which you want to modify from each table to open a dialog. For example, if you want to edit the LogIn table, right-click on that table and a pop-up window will be displayed with multiple editing selections. You can add new queries, new relationships, new keys, even new columns to the LogIn table. Also you can modify or edit any built-in method of the TableAdapter (the LogInTableAdapter in this example), such as Select, Update, Insert, or Delete.

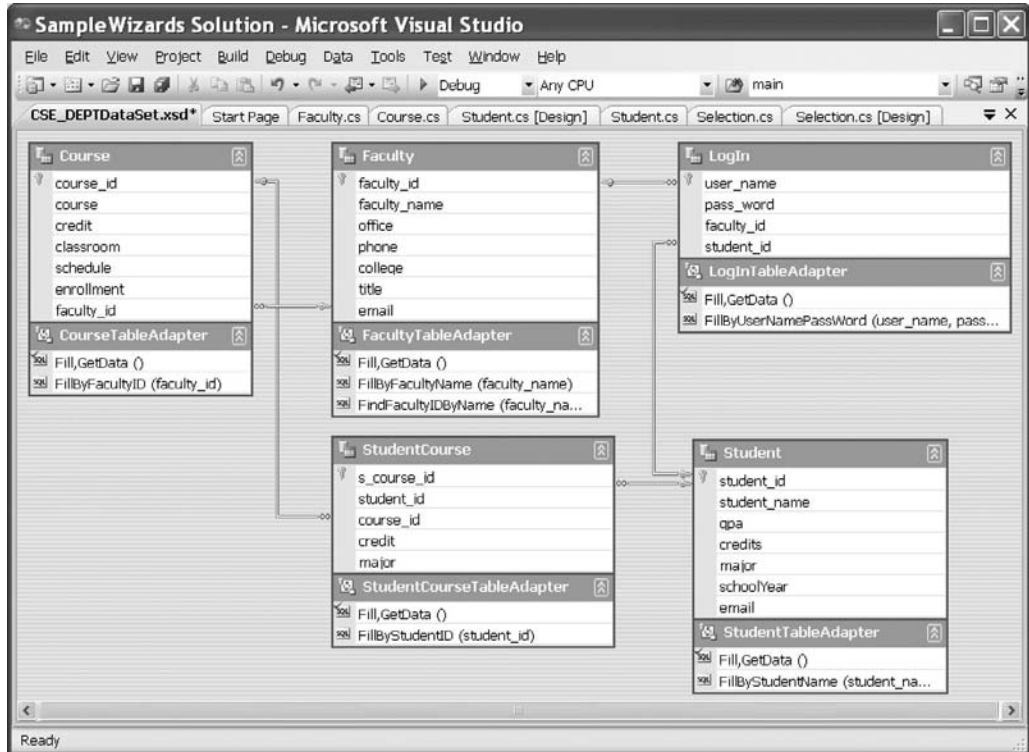


Figure 5.15 Sample DataSet Designer.

In addition to multiple editing abilities mentioned above, you can perform the following popular data operations using the DataSet Designer:

- **Configure:** Configure and build data operations such as building a data query by modifying the default methods of the TableAdapter such as Fill() and GetData().
- **Delete:** Delete the whole table.
- **Rename:** Rename the table.
- **Preview Data:** View the contents of the table in a grid format.

The Preview Data is a very powerful tool and it allows users to preview the contents of a data table. Figure 5.16 shows an example of data table—Faculty table. To open the preview for this Faculty table, right-click on the Faculty table, and select the Preview Data item from the pop-up menu, and then click on the Preview button.

Based on above discussions, it can be seen that the DataSet Designer is a powerful tool to help users design and manipulate the data source added into their projects. But that is not enough! The reason I say that is because the DataSet Designer has one more important function: It allows users to add any missing table to a project.

Of course, you should not have this kind of problem in the normal situation as you develop a data-driven application using the DataSet Designer. But in some cases, you may have forgotten to add a data table, or you add a wrong table (according to my experience, this has happened a lot to students who selected the wrong data source). You

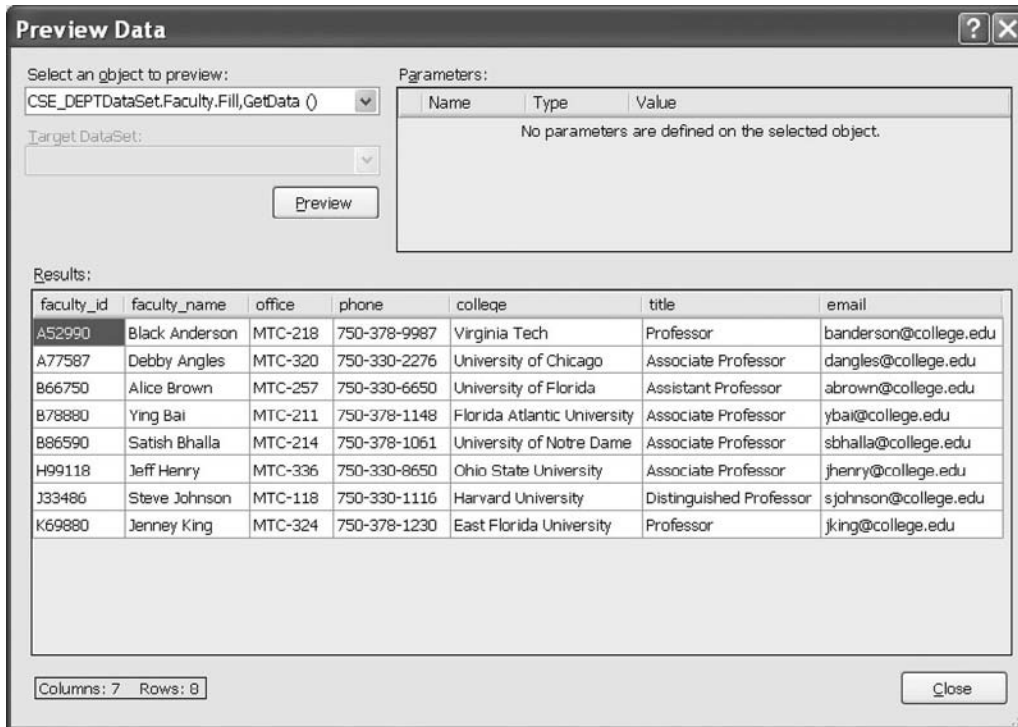


Figure 5.16 Example of Preview Data for Faculty table.

need to use this function to add that missed table by first deleting the wrong table and then adding the correct one.

To add a missed table, just right-click on a blank area of the designer surface and choose `AddDataTable`. You can also use this function to add a `TableAdapter`, `Query`, or a `Relation` to this `DataSet`.

A more detailed exploration of `DataSet Designer` will be provided in Section 5.4.3.

5.3 BUILD A SAMPLE DATABASE PROJECT—SELECTWIZARD WITH SQL SERVER DATABASE

So far we have introduced most design tools and wizards located at both the Visual Studio 2008 Toolbox and the Data Source window. Now we will illustrate how to utilize those tools and wizards to build a data-driven application by using a real example. First, let's build a Visual C# project named **SelectWizard**, which means that we want to build this project with design tools and wizards provided by Visual Studio 2008.

5.3.1 Application User Interfaces

We made a demo of a database project using a similar sample data-driven application in Section 5.1. This project is composed of five forms, titled `LogIn`, `Selection`, `Faculty`,

Table 5.2 Relationship between the Form and Data Table

VB Form	Tables in Sample Database
LogIn	LogIn
Faculty	Faculty
Course	Course
Student	Student, StudentCourse

Student, and Course. The project is designed to map a Computer Science and Engineering Department in a university and allow users to scan and browse all information about the department, including faculty, courses taught by selected faculty, students, and courses taken by the associated students.

Each form, except the Selection form, is associated with one or two data tables in a sample database CSE_DEPT.mdf, which was developed in Chapter 2. The relationship between each form and table is shown in Table 5.2. For your convenience, here we list this table again.

Controls on each form are bound to the associated fields in certain data tables located in the CSE_DEPT database. As the project runs, a data query will be executed via a dynamic SQL statement that is built during the configuration of each TableAdapter in the Data Source wizard or a LINQ. The retrieved data will be reflected on the associated controls that have been bound to those data fields.

The database used in this sample project, which was developed in Chapter 2, is SQL Server 2005 Express database since it is compatible with SQL Server 2005 database, and more important, it is free and can be easily downloaded from the Microsoft Knowledge Base site. You can use any kind of database such as Microsoft Access, SQL Server 2005, or Oracle for this sample project. The only thing you need to do is to select the desired data source when you add and connect that data source to your project.

Let's begin to develop this sample project with five forms.

5.3.1.1 LogIn Form

First, create a new Visual C# 2008 project with the name SelectWizard. Open Visual Studio 2008 and go to the File|New|Project item to open the New Project window. In the Project types pane, expand the Visual C# item and make sure that you select the Windows item under the Visual C# item. Select the Windows Forms Application icon from the Templates pane since we want to create a C# Windows application, and enter SelectWizard into the Name textbox as the project's name. Select a desired folder as the location to save this project by browsing the Location textbox. In our case, we use the folder Book6\Chapter 5 under the root drive as our destination for this project. Keep all other default selections unchanged because we want to create a Solution SelectWizard to hold this new project. Click on the OK button to create this new project and add it into our new Solution. Your finished New Project window should match the one shown in Figure 5.17.

When the new project is created, the default windows form is opened, which is shown in Figure 5.18. Four windows are created and displayed as this new project is created, they are:

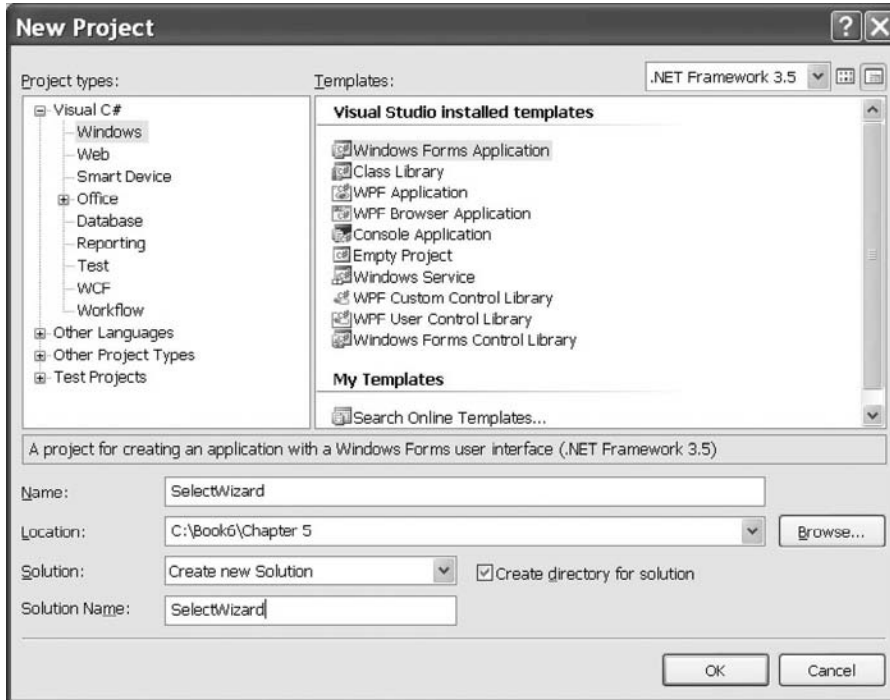


Figure 5.17 Finished New Project window.

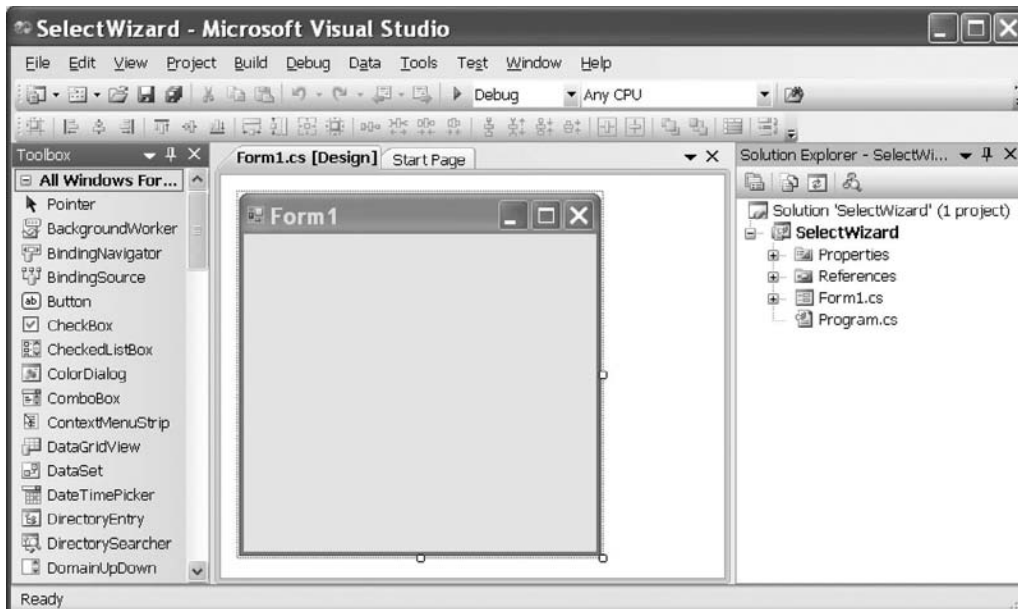


Figure 5.18 Create a new project window.

- Windows Form Designing window: Contains Windows Form object and File Form object.
- Toolbox window: Contains all tools and components to be used to develop Visual C# projects.
- Solution Explorer window: Contains all files and projects created and added into the current solution.
- Properties window: Contains all properties of the selected controls or objects.

Four folders are created and added into the Solution Explorer window for this new project; they are:

- Properties: Contains all properties of the selected object.
- References: Contains all namespaces of the system classes, methods, and events.
- Form1.cs: Contains all files of the graphic user interface and resources.
- Program.cs: Contains the entry point of the project.

Perform the following modifications to this form:

- Change the File Name from Form1.cs to LogIn.cs.
- Change the windows form object's Name property from Form1 to LogInForm.
- Change the Text property of the windows form to CSE DEPT LogIn Form.

Note that when you perform the first operation to change the File Name, a MessageBox will be displayed to ask you whether you prefer to change the name of all references related to this file. The difference is that if you select Yes, you want to change all references related to this file, including the Windows Form object's name, from default Form1 to LogIn. Click on Yes to this MessageBox to confirm that we want to make this change.

When you perform the second name change, you may find that the name of the Windows Form object has already been renamed to LogIn. Yes, that is the result of the first change you made. In order to distinguish this Windows Form object with the File Form object LogIn.cs, we prefer to add Form after the LogIn for the Windows Form object, thus making its name LogInForm.

Add the controls shown in Table 5.3 into the LogInForm window.

The finished LogInForm window should match the one shown in Figure 5.19.

You should select the LogIn button, cmdLogIn, as the default button by choosing this button from the AcceptButton property. Also you need to select the CenterScreen as the StartPosition property of the form.

Table 5.3 Objects and Controls in the LogIn Form

Type	Name	Text	TabIndex
Label	Label1	Welcome to CSE Department	0
Label	Label2	User Name	1
Textbox	txtUserName		2
Label	Label3	Pass Word	3
Textbox	txtPassWord		4
Button	cmdLogIn	LogIn	5
Button	cmdCancel	Cancel	6



Figure 5.19 LogInForm window.

Table 5.4 Objects and Controls in the Selection Form

Type	Name	Text	TabIndex	DropDownStyle
Label	Label1	Make Your Selection	0	
ComboBox	ComboSelection	Faculty Information	1	Simple
Button	cmdOK	OK	2	
Button	cmdExit	Exit	3	
Form	SelectionForm	CSE DEPT Selection		

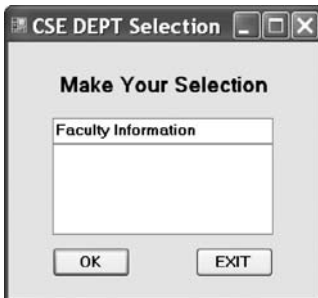


Figure 5.20 Selection form.

5.3.1.2 Selection Form

This form allows users to select the different Windows forms to connect to the different data tables and furthermore to browse data from the associated table. No data table is connected to this form.

To create this form, go to Project/Add Windows Form menu item to open the Add New Items window. Select the Windows Form from the Templates pane and enter Selection.cs into the Name textbox as the name for this new form window, and click on the Add button to add this new form into your project. Add the objects and controls shown in Table 5.4 into this form.

You should select the OK button, cmdOK, as the default button by choosing this button from the `AcceptButton` property of the form. Also you need to select the `CenterScreen` as the `StartPosition` property of the form.

The completed Selection form should match the one shown in Figure 5.20.

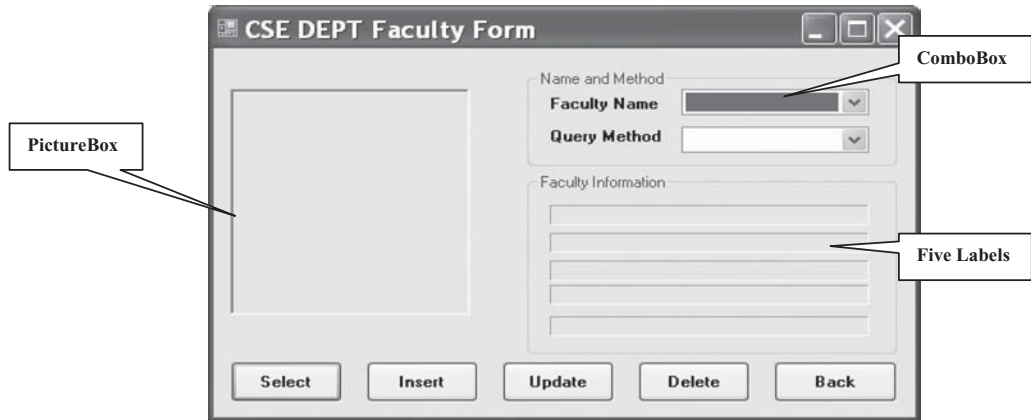


Figure 5.21 Faculty form.

Table 5.5 Objects and Controls in the Faculty Form

Type	Name	Text	TabIndex	DropDownStyle
PictureBox	PhotoBox			
GroupBox	FacultyBox	Name and Method	0	
Label	Label1	Faculty Name	0.0	
ComboBox	ComboName		0.1	DropDownList
Label	Label2	Query Method	0.2	
ComboBox	ComboMethod		0.3	DropDownList
GroupBox	FacultyInfoBox	Faculty Information	1	
Label	TitleLabel		1.0	
Label	OfficeLabel		1.1	
Label	PhoneLabel		1.2	
Label	CollegeLabel		1.3	
Label	EmailLabel		1.4	
Button	cmdSelect	Select	2	
Button	cmdInsert	Insert	3	
Button	cmdUpdate	Update	4	
Button	cmdDelete	Delete	5	
Button	cmdBack	Back	6	
Form	FacultyForm	CSE DEPT Faculty Form		

5.3.1.3 Faculty Form

The Faculty form contains controls that are related to faculty information stored in the database CSE_DEPT, which is the sample database we built in Chapter 2.

Right-click on the project SelectWizard from the Solution Explorer window and select Add/Windows Form item from the pop-up menu to add a new form with a file name Faculty.cs. The finished Faculty form is shown in Figure 5.21.

You should choose the **Select** button, cmdSelect, as the default button by choosing this button from the **AcceptButton** property of the form. Also you need to select the **CenterScreen** as the **StartPosition** property of the form.

The objects and controls shown in Table 5.5 need to be added into this form. In this chapter, we only use the **Select** and **Back** button to make data query to the data source. Other buttons will be used in the following chapters.

Table 5.6 Objects and Controls in the Course Form

Type	Name	Text	TabIndex	DropDownStyle
GroupBox	NameBox	Name and Method	0	
Label	Label1	Faculty Name	0.0	
ComboBox	ComboName		0.1	DropDownList
Label	Label2	Query Method	0.2	
ComboBox	ComboMethod		0.3	DropDownList
GroupBox	CourseBox	Course List	1	
ListBox	CourseList		1.0	
GroupBox	CourseInfoBox	Course Information	2	
Label	CourseTitleLabel	Course	2.0	
TextBox	txtName		2.1	
Label	ScheduleLabel	Schedule	2.2	
TextBox	txtSchedule		2.3	
Label	ClassRoomLabel	Classroom	2.4	
TextBox	txtClassRoom		2.5	
Label	CreditsLabel	Credits	2.6	
TextBox	txtCredits		2.7	
Label	EnrollLabel	Enrollment	2.8	
TextBox	txtEnroll		2.9	
Button	cmdSelect	Select	3	
Button	cmdInsert	Insert	4	
Button	cmdBack	Back	5	
Form	CourseForm	CSE DEPT Course Form		

5.3.1.4 Course Form

This form is used to interface to the Course table in our data source to retrieve course information associated with a specific faculty member selected by the user. Recall that in Chapter 2 we developed a sample database CSE_DEPT.mdf, and the Course table is one of the five tables built into that database. A one-to-many relationship exists between the Faculty and the Course table, which is connected by using a primary key `faculty_id` in the Faculty table and a foreign key `faculty_id` in the Course table. We will use this relationship to retrieve data from the Course table based on the `faculty_id` in both tables.

Go to Project|Add Windows Form menu item to add a new windows form with a file name `Course.cs`. Add the following objects and controls, which are shown in Table 5.6, into this form window.

You should choose the **Select** button, `cmdSelect`, as the default button by choosing it from the `AcceptButton` property of the form. Also you need to select the `CenterScreen` as the `StartPosition` property of the form.

The finished Course form should match the one shown in Figure 5.22.

The objects shown in Table 5.6 need to be added into this Course form.

In this chapter, we only use the **Select** and the **Back** button to make data query from the data source. The **Insert** button will be used in Chapter 6.

5.3.1.5 Student Form

The Student form is used to collect and display student information, including the courses taken by the student. As we mentioned in Section 5.1, the Student form needs two data tables in the database; one is the Student table and the other is the StudentCourse table. This is a typical example of using two data tables for one graphical user interface (form).

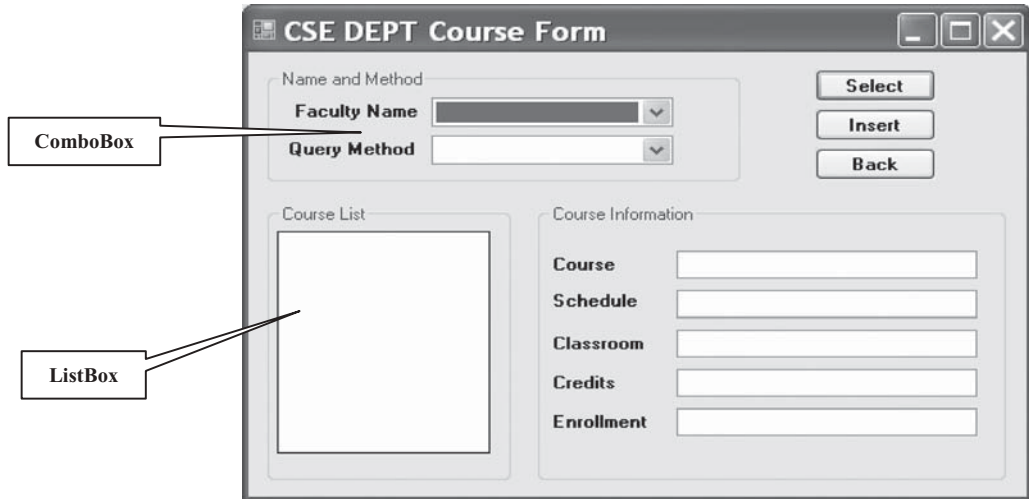


Figure 5.22 Course form.

Table 5.7 Objects and Controls in the Student Form

Type	Name	Text	TabIndex	DropDownStyle
PictureBox	PhotoBox			
GroupBox	NameMethodBox	Name and Method	0	
Label	Label1	Student Name	0.0	
ComboBox	ComboName		0.1	DropDownList
Label	Label2	Query Method	0.2	
ComboBox	ComboMethod		0.3	DropDownList
GroupBox	CourseSelectedBox	Course Selected	1	
ListBox	CourseList		1.0	
GroupBox	StudentInfoBox	Student Information	2	
Label	Label2	Student ID	2.0	
TextBox	txtID		2.1	
Label	Label3	School Year	2.2	
TextBox	txtSchoolYear		2.3	
Label	Label4	GPA	2.4	
TextBox	txtGPA		2.5	
Label	Label5	Major	2.6	
TextBox	txtMajor		2.7	
Label	Label6	Credits	2.8	
TextBox	txtCredits		2.9	
Label	Label7	Email	2.10	
TextBox	txtEmail		2.11	
Button	cmdSelect	Select	3	
Button	cmdInsert	Insert	4	
Button	cmdBack	Back	5	
Form	StudentForm	CSE DEPT Student Form		

Go to Project\Add Windows Form menu item to add a new Windows form with a file name **Student.cs**. Add the objects and controls listed in Table 5.7 into this form.

Make sure that you set up the following properties for controls and objects in this form:

Figure 5.23 Student form.

- Make the Select button the default button by selecting this button from the `AcceptButton` property of the form.
- Select the `CenterScreen` as the `StartPosition` property of the form.
- Set the `BorderStyle` property of the `ListBox` control, `CourseList`, to `FixedSingle`.

Also in this Student form, we use `TextBoxes` to replace `Labels` to bind and display the student's information. The courses taken by the student are reflected and displayed in a `ListBox` control, `CourseList`.

Your finished Student form should match the one shown in Figure 5.23. Go to the `File/Save All` menu item to save all forms we developed and built into these sections. Next we need to select and add our desired data source to this project and connect our project with our database to perform the data query operations.

5.4 ADD AND UTILIZE VISUAL STUDIO.NET WIZARDS AND DESIGN TOOLS

After the graphical user interfaces are created, we need to add a data source to this new project and set up a connection between our project and the database. In Section 5.2.2, we discussed in detail about how to add a new data source and how to configure it. In this section, we will illustrate these with the real Visual C# 2008 project, `SelectWizard`, we created in the last section.

5.4.1 Add and Configure a New Data Source

Open the project `SelectWizard` if it is not opened and select the `LogIn` form window. Go to `Data/Show Data Sources` menu item to open the `Data Source` window. Currently, this window is blank since we have not added and connected any data source to this project.

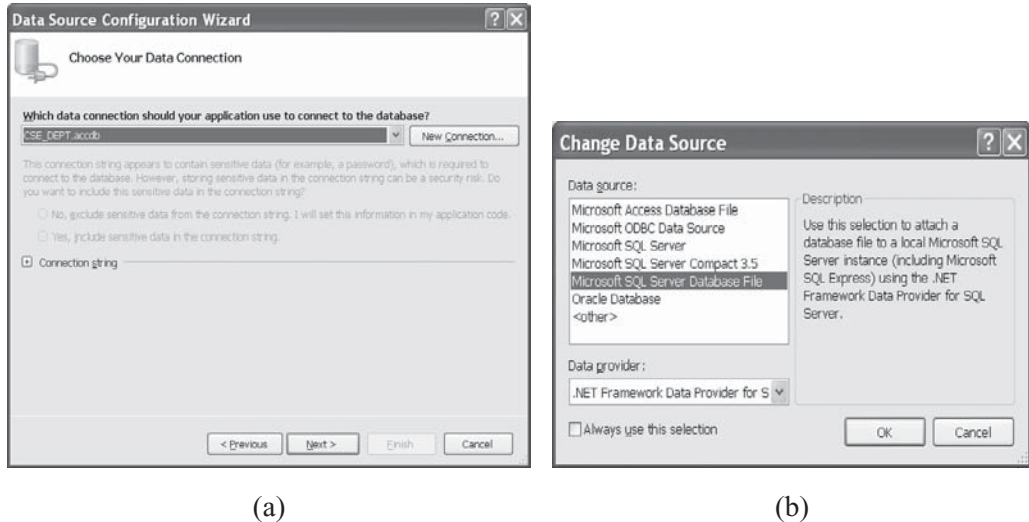


Figure 5.24 Change Data Source dialog.

Click on the link Add New Data Source as we did in Section 5.2.2.1 to add a new data source to the project.

On the opened Data Source Configuration Wizard, keep the default selection Database unchanged and click on Next to go to the next step, which is shown in Figure 5.24a.

Click on the New Connection button to open the Add Connection dialog. This dialog allows you to select your database with two specifications: database type and database name, and both are determined by the actual database you want to use for this project. As we mentioned before, we want to use an SQL Server Express database for this project. Thus, first, we need to change the default database type from the Microsoft Access Database File to the Microsoft SQL Server Database File by clicking on the Change button, which is located next to the Data Source box.

On the opened Change Data Source dialog, select the Microsoft SQL Server Database File, which is shown in Figure 5.24b. Click on OK to select this database type and return to the Add Connection dialog.



Microsoft SQL Server is different with Microsoft SQL Server Database File. The former is used for the real server/client database, but the latter is used for a local SQL Server instance, such as SQL Server Express. This means that you can install both an SQL Server and an SQL Client in your local computer by using SQL Server Express 2005. Your computer works as both a server and a client, and you access the SQL Server database file from that local server.

Next we need to select the database file. Click on the Browse button to locate your database file. You should have developed and built your database files using Microsoft SQL Server 2005 Express in Chapter 2. Basically this database file should be located at C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data in your computer. In

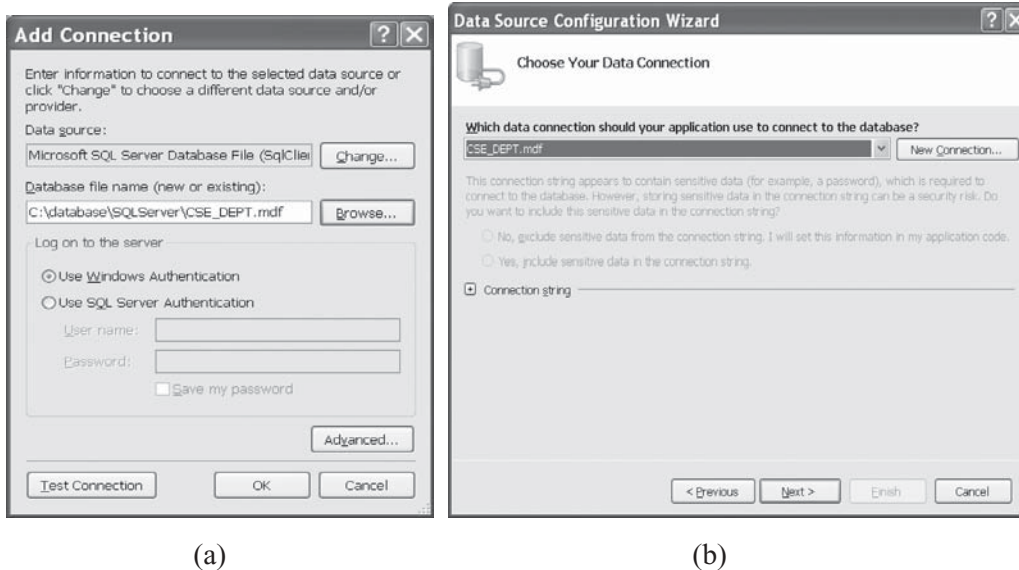


Figure 5.25 Add Connection and Data Source Configuration Wizard.

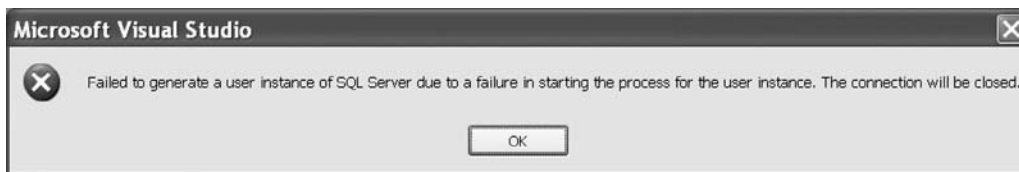


Figure 5.26 Error connection message.

this example, the database name is CSE_DEPT.mdf with five data tables. You can find this database file in the folder Database\SQLServer located at the accompanying ftp site (see Chapter 1). You can copy this database file and paste it in a folder at your local computer, then select this database file and click on the Open button to add it into your project. In this case, we store this database file in the folder database\SQLServer in the root drive. A completed Add Connection dialog is shown in Figure 5.25a.

For database security, we used the default Windows Authentication mode. You can select the SQL Server Authentication mode if you like for your database connection.

Now we can test this database connection by clicking the Test Connection button to confirm that this connection to our database works fine. But before we can do that, we want to make it clear that a potential bug exists in this database connection, and it is not easy to debug.

If you installed an old version of SQL Server such as SQL Server 2000 or 2005 Express SP1 on your computer, and later on you removed that old software from your computer and installed a new version of SQL Server such as SQL Server 2005 Express SP2 or SQL Server 2008, an error message would be displayed when you click on this Test Connection button to test your database connection, which is shown in Figure 5.26.

The problem is due to the old version of SQL Server database engine. Basically, it is the old instance of the SQL Server database that was created when you used it in your data-driven applications. That old instance is still in your machine even though you removed the old database server. Sometimes this bug can be solved by uninstalling the old SQL Server a few times, but that is often difficult. Yes, bug existed in the SQL Server 2005. One solution is to neglect checking the database instance by resetting the User Instance property of the database server to False.



This SQL Server database bug possibly exists in any machine in which an old SQL Server was installed and then removed in order to allow a new version of SQL Server to be installed in the same machine. It is highly recommended to upgrade an old version of SQL Server instead of removing the old version and installing a new one.

Now let's perform this operation to solve this potential bug. Still in the Add Connection dialog, click on the Advanced button to open the Advanced Properties for this connection. Scroll down in this dialog until you find the User Instance property. Change this property from the default value, True, to False since we do not want to check this new database instance when the database is connected to our project. Click on the OK button to close the Advanced Properties dialog. Click on the Test Connection button again, and this time a *Connection successful* message is displayed, and it indicates that the database connection is successful.

Click on the OK button to return to the Data Source Configuration Wizard, which is shown in Figure 5.25b. Click on the Next button to go to the next step.

A message box will pop-up to ask if you would like to add this data source to your project. As we discussed in Section 5.2.2.2, click on Yes to save the data source into your project.

The next window shows a message to ask if you would like to save this connection string for future usage; select the checkbox to save it and click on the Next button to continue.

The next step allows you to select different database objects. Generally, all data tables are necessary because we need to use those data to perform data operations between our Visual C# 2008 project and those tables in the connected database. The View object provides a full view of tables, and it allows users to open and scan all data using the Preview Data functionality. For other objects, they are not necessary for developing simple data-driven applications. Stored Procedures are used to integrate a sequence of queries into a procedure to speed up the data query operations. The Functions objects provide some special functions to facilitate the building of data-driven applications. It would not be a problem if you select all of them. So just check all checkboxes to select all of them, as shown in Figure 5.27.

You may already find that a new DataSet with a name of CSE_DEPTDataSet is created and is located in the DataSet name box. Click on the Finish button to complete this data source configuration.

After you finish this Data Source Configuration, a new instance of the DataSet with a name of CSE_DEPTDataSet is added into the project, which is shown in Figure 5.28.



Figure 5.27 Select the database objects.

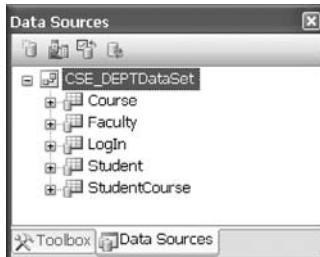


Figure 5.28 New data source CSE_DEPTDataSet.

Five data tables, LogIn, Faculty, Course, Student, and StudentCourse, are included in this DataSet instance. These five data tables are only mappings or copies of those real tables in the database. The connection you set up between your project and your database is closed as this wizard is finished. You need to call some data query or manipulation methods to reopen this connection as you perform some data queries or actions later in your application.

5.5 QUERY AND DISPLAY DATA USING THE DATAGRIDVIEW CONTROL

Now we have added a data source into our Visual C# 2008 project. Before we can develop this data-driven project, we want to consider a popular and important tool provided by the Toolbox window; it is DataGridView. We discussed this issue in Section 5.2.1.2. The

DataGridView is a view container and can be used to bind data from your database and display the data in a tabular or a grid format in your Visual C# form windows.

To use this tool, add a new blank form to the project SelectWizard, and name this new form Grid. Go to Project|Add Windows Form to open the Add New Item dialog; enter Grid.cs into the Name box and click on the Add button.

Change the name of the form window of the new added form from Grid to GridForm. Keep the form selected, and then open the Data Source window (if it is not opened) by clicking on the Data menu item from the menu bar. You can view data from any table in your data source window. The two popular views are Full Table view and Detail view for specified columns.

Here we use the Faculty table as an example to illustrate how to use these two views.

5.5.1 View Entire Table

To view the full Faculty table, click on the Faculty table from the Data Source window, click the drop-down arrow, and select the DataGridView item. Then drag the Faculty table to the GridForm window, which is shown in Figure 5.29.

As soon as you drag the Faculty table to the GridForm, a set of graphical components is created and added into your form automatically, which include the browsing arrows and the Addition, Delete, and Save buttons. This set of components helps you to view data from the selected table. To make a full table view, make sure to set two properties of the DataGridView, `AutoSizeColumnsMode` and `AutoSizeRowsMode`, to `AllCells`. In this way, all the data can be displayed on this grid view tool.

Besides those graphical components, a set of design tools, which includes the `cSE_DEPTDataSet`, `facultyBindingSource`, `facultyTableAdapter`, `facultyBindingNavigator`, and

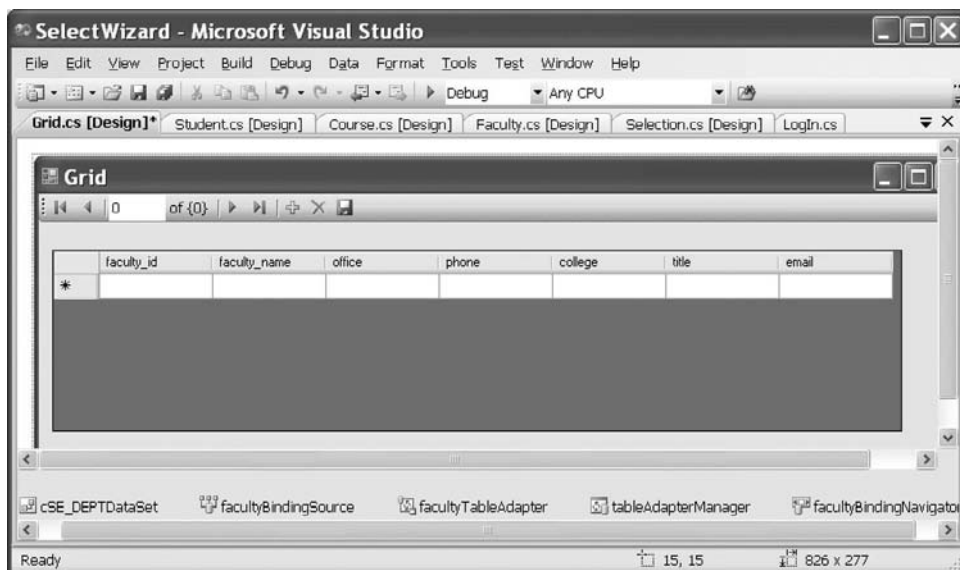


Figure 5.29 DataGridView tool.

tableAdapterManager, is also added into this project when you drag the Faculty table onto the GridForm window. As we discussed in Section 5.2.1, these tools are used to help users to effectively access, translate, and manipulate data between the Visual C# project and the DataSet.



One important point to be noted is that these new added design tools (cSE_DEPTDataSet, facultyTableAdapter, ...) are basically instances, not the classes, of these design tools. In Visual C#, they are called fields since these instances are class level, and they are very similar to the Form level variables in Visual Basic.NET and the class variables in Java. This is the reason why the first letter of each of these instances is lowercase. This is a significant difference between Visual Basic.NET and Visual C#.NET. In Visual Basic.NET, all of these new added tools are classes, not instances. You must keep this in mind when you develop data-driven applications in Visual C#.

Now you can run your project by clicking on the Start button. But wait a moment! One more thing before you run your project: Check whether you have selected your GridForm as the startup object. There is a difference if you select a startup object for a project using Visual C#.NET or Visual Basic.NET. In Visual Basic.NET, one can do that by going to the Project Properties window and selecting the startup form from the Startup object box. But in Visual C#.NET, you cannot do this job in that way. As you know, the Program.cs in the Solution Explorer window contains the entry point of each Visual C# project. Basically this Program.cs contains the Main() method from which the startup object can be defined.

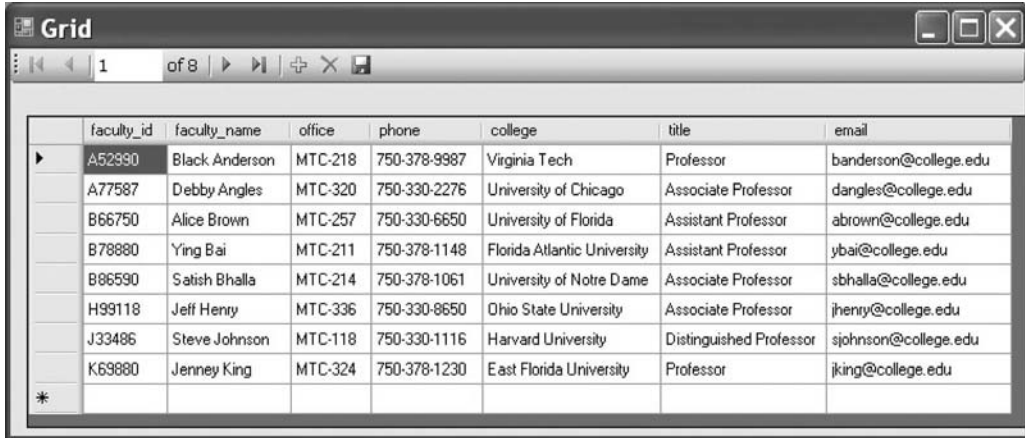
To check the startup object for our current project, SelectWizard, open the Program.cs from the Solution Explorer window by double clicking it. You can find that one instruction,

```
Application.Run(new LogInForm());
```

is located inside this Main() method. The startup object is defined by calling a system method Run() with the constructor of the LogInForm as the argument. To change the startup object for our current project, modify this argument to GridForm(), which means that we want to call the constructor of the GridForm class to create a new instance of the GridForm class, and start our project from this form. Now run our project by going to Debug menu item and clicking on the Start Debugging. The running result is shown in Figure 5.30.

By using this grid view tool, you can not only view data from the Faculty table but also add new data into and delete data from the table by clicking on the Add (+) or Delete (x) button. If you want to add new data, just type the new data in a new line after you click on the Add button. If you want to delete data, move to the data you want to delete by clicking the browsing arrow on the top of the form window and then click on the Delete button. One thing you need to know is that these modifications can only take effect to data in your data tables in the DataSet. It has nothing to do with data in your database.

When you drag the Faculty table from the data source window to the GridForm, what is happening behind this dragging? Let's take a closer look at this issue. To do that, click on the Close button to stop our project's running.



	faculty_id	faculty_name	office	phone	college	title	email
▶	A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.edu
	A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
	B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@college.edu
	B78980	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Assistant Professor	ybai@college.edu
	B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
	H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
	J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
	K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu
*							

Figure 5.30 Entire table view for the Faculty table.

First, from the point of view of the graphical user interface, as we mentioned, five instances of design tools are created and added into this project. Second, from the point of view of the program codes, some instructions have been automatically added into this project, too. Open the code window of the GridForm by clicking on it from the Solution Explorer window and then clicking on the View Code button. Browse to the method GridForm_Load() and you can find that a line of instruction is in there:

```
this.facultyTableAdapter.Fill(this.cSE_DEPTDataSet.Faculty);
```

The function of this instruction is that the Fill() method, which belongs to the FacultyTableAdapter class, is called to load data from the Faculty table in the database into the Faculty table in the DataSet and also into your DataGridView window. The Fill() is a very powerful method, and it performs an equivalent operation as an SQL SELECT statement did. To make this clearer, open the Data Source window, and right-click on any place inside that window. Select the Edit the DataSet with Designer item to open the DataSet Designer Wizard. Right-click on the bottom line, in which the Fill() and the GetData() methods are shown, on the Faculty table, and then select the Configure item to open the TableAdapter Configuration Wizard. You will find that a complete SQL SELECT statement is already in there:

```
SELECT faculty_id, faculty_name, title, office, college, phone,  
email FROM dbo.Faculty
```

This statement will be executed when the Fill() method is called by the faculty-TableAdapter as the GridForm_Load() event method runs when you start this project. The data returned from executing this statement will fill the grid view tool in the Faculty form.

5.5.2 View Each Record or Specified Columns

To view each record from the Faculty table, first delete the grid view tool from the Faculty form. Then go to the Data Source window and click the Faculty table. Click on the drop-



Figure 5.31 Grid view for specified columns.

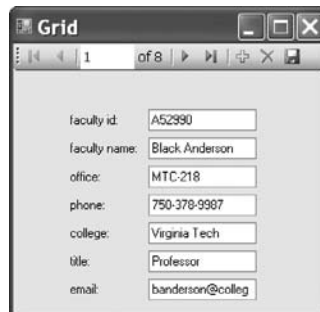


Figure 5.32 Running status of the grid view for each record.

down arrow and select the Detail item. Drag the Faculty table from the Data Source window to the Faculty form window. All column headers in the Faculty table are displayed, which is shown in Figure 5.31.

Now click on the Start button to run your project, and the first record in the Faculty table is displayed in this grid tool, which is shown in Figure 5.32. To view each record, you can click the forward arrow on the top of the form to scan all records from the top to the bottom of the Faculty table.

If you only want to display some specified columns from the Faculty table, go to the Data Source window and select the Faculty table. Expand the table to display the individual columns, and drag the desired column from the Data Source window onto the

GridForm window. For each column you drag, an individual data-bound control is created on the GridForm, accompanied by an appropriately titled label control. When you run this project, the first record with the specified columns will be retrieved and displayed on the form, and you can scan all records by clicking the forward arrow.

Well, the DataGridView is a powerful tool and allows users to view all data from a table. But generally we do not want to perform this kind of data view as a in-line SQL statement did. The so-called in-line SQL statement must be already defined in full before your project runs. In other words, you cannot add any parameter into the query statement after your project runs, and all query parameters must be defined before your project runs. But running SQL statements dynamically is a very popular style for today's database operations, and in the following sections we will implement this technology and the LINQ method to perform the data query from our five tables in the database CSE_DEPT.mdf with the project SelectWizard.

5.6 USE DATASET DESIGNER TO EDIT THE STRUCTURE OF DATASET

After a new data source is added into a new project, the next step is to edit the DataSet structure based on your applications if you want to develop a dynamic SQL statement. The following DataSet structures can be edited by using the DataSet Designer:

- Build user-defined query in an SQL statement format.
- Modify the method of the TableAdapter to match the users' preference.

Now let's begin to develop a dynamic SQL statement or a user-defined query with a real example. We still use the sample project SelectWizard developed in the previous sections. We start from the LogIn table.

Open the Data Source window and right-click any place inside the window, and select Edit DataSet with Designer to open the DataSet Design Wizard. Locate the LogIn table and right-click on the last line, in which two methods—Fill() and GetData()—are displayed, and select the AddQuery item from the pop-up menu. Of course, you can select other items such as Configure to modify an existing query. But right now we do not want to configure any existing query; instead we just want to add a new query to perform our specified data query.

On the opened TableAdapter Configuration Wizard, click on the Query Builder button to open the Query Builder window to build our desired dynamic query. The opened Query Build window is shown in Figure 5.33.

Query Builder provides a graphical user interface (GUI) for creating SQL queries, and it is composed of graphical panes and test panes. The top two panes are graphical panes and the third pane is the text pane. You can select desired columns from the top graphical pane, and each column you select will be added into the second graphical pane. By using the second graphical pane, you can install desired criteria to build user-defined queries. The query you built will be translated and presented by a real SQL statement in the text pane.

By default, all columns in the LogIn table are selected in the top graphical pane. You can decide which column you want to query by checking the associated checkbox in front of each column. In this application, we prefer to select all columns from the top graphical

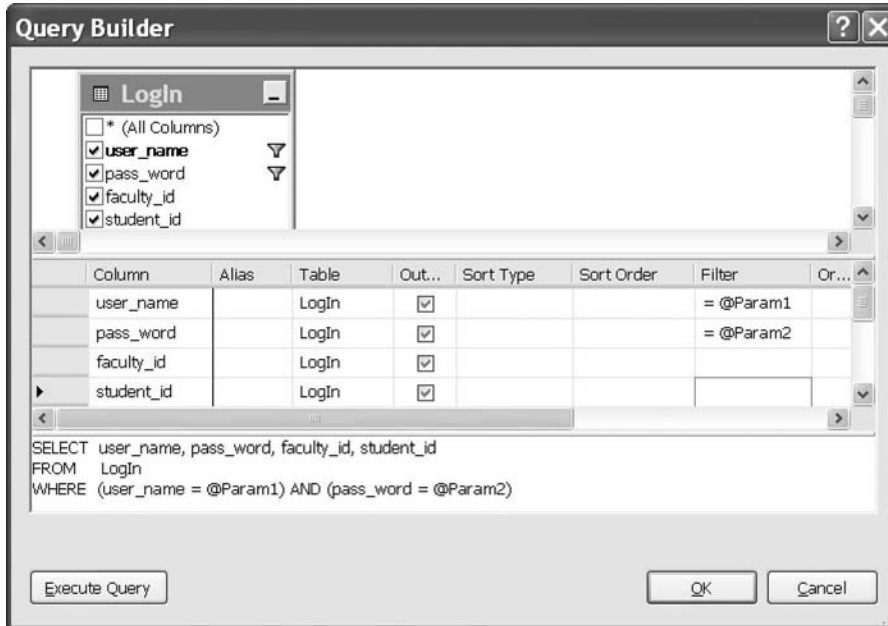


Figure 5.33 Query Build window.

pane. The selected columns will be displayed in the second graphical pane, which is also shown in Figure 5.33.

Since we try to build a dynamic SQL query for the LogIn table, what we want to do is as follows. When the project runs, the username and password are entered by the user, and those two items will be embedded in an SQL SELECT statement that is sent to the data source, basically to the LogIn table, to check if the username and password entered by the user can be found from the LogIn table. If a match is found, that matched record will be read back from the DataSet to the BindingSource via the TableAdapter, and then reflected on the bound textbox control on the Visual C# 2008 form window.

The problem is that when we build this query, we do not know the values of username and password, which will be entered by the user as the project runs. In other words, these two parameters are dynamic parameters. So in order to build a dynamic query with two dynamic parameters, we need to use two question marks (?) to temporarily replace those two parameters in the SQL SELECT statement. We do this by typing a question mark in the Filter column for the user_name and pass_word rows in the second graphical pane, which is shown in Figure 5.33. The two question marks will become two dynamic parameters represented by =@Param1 and =@Param2, respectively, after you press the Enter key from the keyboard when you finish typing the two question marks. This is the typical representation method for the dynamic parameters used in the SQL Server database query.

Now let's go to the text pane, and you can find that a WHERE clause is attached at the end of the SELECT statement, which is shown in Figure 5.33. The clause

```
WHERE (user_name = @Param1) AND (pass_word = @Param2)
```

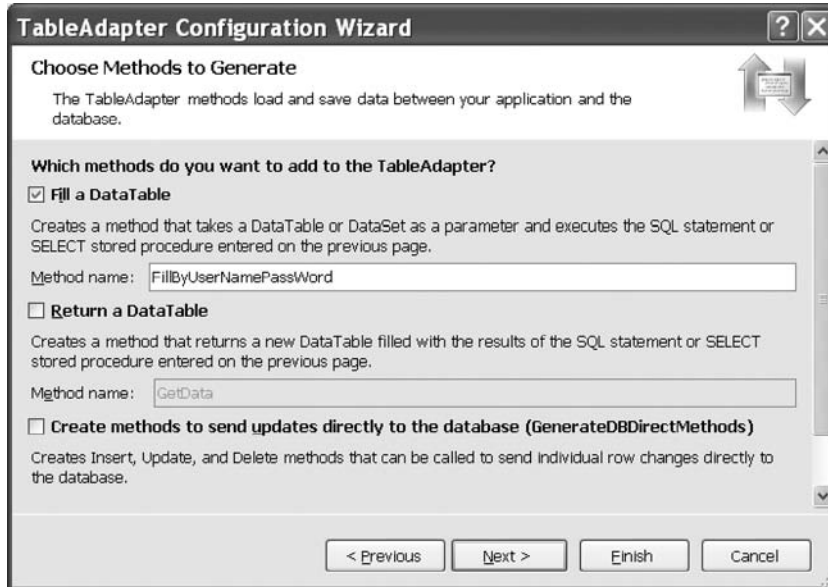


Figure 5.34 Choose Methods to Generate window.

is used to set up a dynamic criteria for this SELECT statement. Two dynamic parameters, Param1 and Param2, will be replaced by the actual username and password entered by the user as the project runs. You can consider the @ symbol as a * in C++, which works as an address. So we leave two addresses that will be filled later by two dynamic parameters, username and password, as the project runs.

Click on the OK button to continue to the next step. The next window shows the complete query we built from the last step in the text format to ask your confirmation, and you can make any modification if you want. Click on the Next button to go to the next step.

The next window provides you with three options; the first one allows you to modify the Fill() method to meet your specified query for your application. The second option allows you to modify the GetData() method, which returns a new data table filled with the executing results of the SQL statement above. The third one allows you to add other SQL statements such as Insert, Update, and Delete methods.

For this application, we only need to use and modify the Fill() method. To do that, attach the word ByUserNamePassWord to the end of the Fill method in the Method name textbox, which is shown in Figure 5.34. Also uncheck the following two checkboxes, Return a DataTable and Create, to send updates directly to the database. We will use this modified Fill() method in our project to run the dynamic SQL statement we built in the last step. Click on the Next button to go to the next page.

The next window shows the result of your TableAdapter configuration. If everything is going smoothly, all statements and methods should be created and modified successfully, as shown in Figure 5.35. Click on the Finish button to close this configuration.

Before we can begin to do our coding, we need to bind data to controls on the LogIn form to set up the connection between each control on the form and each data on the data source.

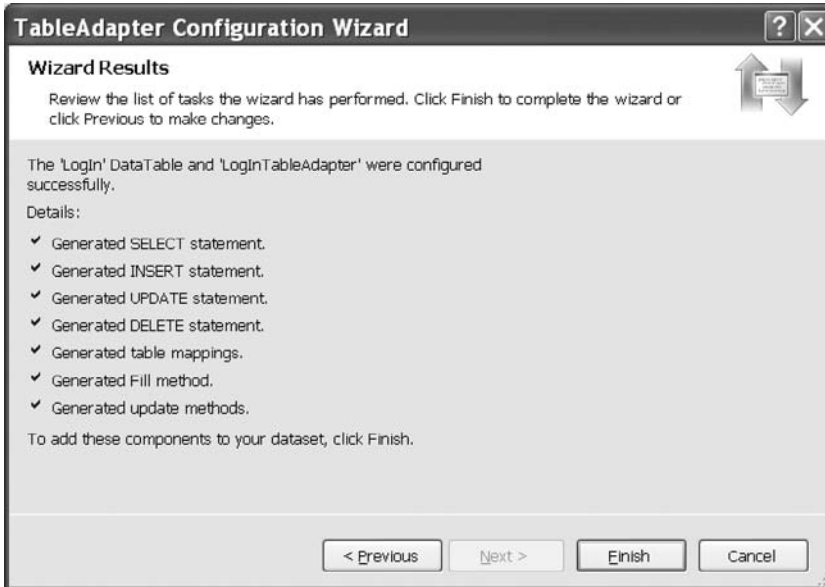


Figure 5.35 Result of the TableAdapter Configuration Wizard.

5.7 BIND DATA TO ASSOCIATED CONTROLS IN LOGIN FORM

Open the Solution Explorer window and select the LogIn Form from that window, then click on the View Designer button to open its graphical user interface. Now we want to use the BindingSource to bind controls in the LogIn form, that is, the User Name and Pass Word TextBoxes, to the associated data fields in the LogIn table in the data source.

Click the User Name TextBox, then go to the DataBindings property located in the top section of the property window. Expand the property to display the individual items, and then select the Text item. Click on the drop-down arrow to expand the following items:

- Other Data Sources
 - Project Data Sources
 - CSE_DEPTDataSet
 - LogIn
 - user_name

The expansion result is shown in Figure 5.36.

Then select the user_name column by clicking on it. In this way, we finish the data binding and set up a connection between the User Name TextBox control on the LogIn form and the user_name column in the LogIn data table.

You can find that three objects, cSE_DEPTDataSet, logInBindingSource, and logInTableAdapter, added into the project and displayed at the bottom of the window after you finish this binding.

Well, was that easy? Yes. Perform the similar operations for the Pass Word TextBox to bind it with the pass_word column in the LogIn table in the data source. But one point you need to note is: When we perform the data binding for the User Name TextBox,



Figure 5.36 DataBindings property.

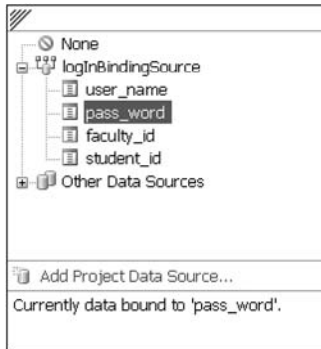


Figure 5.37 New created BindingSource object.

there is no prebuilt or default BindingSource object available because we have not performed any data binding before, and the User Name is the first control you want to bind. But after you finish the first binding, a new BindingSource object, logInBindingSource, is created. You need to use this created BindingSource object to handle all other data binding jobs for all other controls on the LogIn form.



When you perform the first data binding, there is no default BindingSource object available since you have not performed any binding before. You can browse to the desired data column and select it to finish this binding. Once you finish the first binding, a new BindingSource object is created, and all the following data bindings should use that new created BindingSource to perform all data bindings.

Let's perform the data binding for the Pass Word TextBox now.

Click on the Pass Word TextBox to select it, and then go to the DataBindings property, select the Text item and then click on the drop-down arrow. This time you will find that a new BindingSource object, logInBindingSource, shows up (Figure 5.37). Expand

this new object and select the `pass_word` column by clicking on it. The data binding is done.

Some readers may have noted that when we call the `FillByUsernamePassword()` method, we fill the LogIn form with four columns; `user_name`, `pass_word`, `faculty_id`, and `student_id` from the LogIn table. In fact, we only fill two textbox controls on the form, `txtUserName` and `txtPassWord`, with two associated columns in the LogIn table, `user_name` and `pass_word`. We only need to know if we can find the matched username and password entered by the user from the LogIn table. If both matched items can be found from the LogIn table, the login is successful and we can continue to the next step. Two bound controls on the form, `txtUserName` and `txtPassWord`, will be filled with the identical values stored in the LogIn table. It looks like this does not make sense. In fact, we do not want to retrieve any column from the LogIn table. Instead, we only want to find the matched items of username and password from the LogIn table. If we can find matched username and password, we do not care whether we fill the `faculty_id` and `student_id` or not. If no matched items can be found, this means that the login has failed and a warning message should be displayed.

Before we can go ahead with our coding, one we need to point out the displaying style of the password in the textbox control `txtPassWord`. Generally, the password letters will be represented by a sequence of stars (*) when users enter them as the project is running. To make this happen in our project, we need to set the `PasswordChar` property of the textbox control `txtPassWord` to a star (*).



To check the matched username and password entered by the user from the data source, one can use *Return a Single Value to Query Data* for LogIn table. But here in order to simplify this check, we use the `Fill()` method to fill four columns in a mapped data table in the `DataSet`. Then we can check whether this `Fill()` is successful. If it is, the matched data items have been found. Otherwise no matched data items are found.

Now it is the time for us to develop codes that are related to the objects we created in the previous steps such as the `BindingSource` and `TableAdapter` to complete the dynamic query. The operation sequences of the LogIn form are as follows:

1. When the project runs, the user needs to enter the username and password to two textbox controls, `txtUserName` and `txtPassWord`.
2. Then the user will click on the LogIn button on the form to execute the LogIn button click method.
3. The LogIn button click method will first create some local variables or objects that will be used for the data query and a new object for the next form.
4. Then the method will call the `FillByUsernamePassword()` method to fill the LogIn form.
5. If this `Fill` is successful, which means that the matched data items for username and password have been found from the LogIn table, the next window form, `SelectionForm`, will be displayed for the next step.
6. Otherwise, a warning message is displayed.

As we discussed in Section 5.5.1, these new created design tools, `cSE_DEPTDataSet`, `logInTableAdapter`, and `logInBindingSource`, are not the classes but the instances of design tools. Therefore we can directly use these instances to develop our code. Keeping this in mind, now let's begin to develop the codes for the LogIn form.

5.8 DEVELOP CODES TO QUERY DATA USING FILL() METHOD

Select the `LogIn.cs` from the Solution Explorer window and click on the View Designer button to open its graphical user interface. Double click on the LogIn button to open its Click method.

First, we need to create a local object `selForm`, which is an instance of the `SelectionForm` class and then enter the codes shown in Figure 5.38 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. A new namespace is created by the Visual C#, and the name of this namespace is equal to the name of our project, `SelectWizard`. By using the namespace technology, it is much easier to distinguish the different variables, methods, delegates, and events that have the same name but are located at different spaces.
- B. This line indicates that our `LogIn` form class is derived from the system class `Form`.
- C. The constructor of our `LogIn` form class contains a built-in method, `InitializeComponent()`. This method is used to initialize all new created instances and variables in this form. Starting Visual C# 2008, this method is moved to the `LogIn.Designer.cs` file.

```

SelectWizard.LogInForm
cmdLogIn_Click

namespace SelectWizard
{
    public partial class LogInForm : Form
    {
        public LogInForm()
        {
            InitializeComponent();
        }
        private void cmdLogIn_Click(object sender, EventArgs e)
        {
            SelectionForm selForm = new SelectionForm();
            logInTableAdapter.ClearBeforeFill = true;
            logInTableAdapter.FillByUserNamePassWord(cSE_DEPTDataSet.LogIn, txtUserName.Text, txtPassWord.Text);
            if (cSE_DEPTDataSet.LogIn.Count == 0)
            {
                MessageBox.Show("No matched username/password found!");
                txtUserName.Clear();
                txtUserName.Focus();
                txtPassWord.Clear();
            }
            else
            {
                selForm.Show();
                this.Hide();
            }
        }
    }
}

```

Figure 5.38 Coding of the LogIn button Click method.

- D. Our LogIn button's Click method contains two arguments: The sender indicates the current object that triggers this method, and the second argument e contains the additional information for this event.
- E. As this method is triggered and executed, first we need to create an instance of our next form window, SelectionForm.
- F. Before filling the LogIn data table, clean up that table in the DataSet. As we mentioned in Section 5.2.1.1, the DataSet is a table holder and it contains multiple data tables. But these data tables are only mappings to those real data tables in the database. All data can be loaded into these tables in the DataSet by using the TableAdapter when the project runs. Here a property ClearBeforeFill, which belongs to the TableAdapter, is set to True to perform this cleaning job for that mapped LogIn data table in the DataSet.
- G. Now we need to call the Fill() method we created in Section 5.6, exactly the FillByUsernameAndPassword(), to fill the LogIn data table in the DataSet. Because we have already bound two textbox controls on the LogIn form, txtUserName and txtPassword, with two columns in the LogIn data table in the DataSet, user_name and pass_word, by using the logInBindingSource, so these two filled columns in the LogIn data table will also be reflected in those two bound textbox controls, txtUserName and txtPassword, when this Fill() method is executed.
 This Fill() method has three arguments; the first one is the data table, in this case it is the LogIn table that is held by the DataSet, CSE_DEPTDataSet. The following two parameters are dynamic parameters that were temporarily replaced by two question marks when we modified this Fill() method in Section 5.6. Now we can use two real parameters, txtUserName.Text and txtPassword.Text, to replace those two question marks to complete this dynamic query.
- H. If no matched username and password can be found from the LogIn table in the database, the Fill() method cannot be executed to fill the LogIn table in the DataSet. This situation can be detected by checking the Count property of the LogIn table in the DataSet. This Count property represents the number of rows that have been successfully filled to the LogIn table in the DataSet. A zero value means that no matched username and password has been found and this fill has failed. A warning message is displayed if this happens and some cleaning jobs are performed for two textboxes in the LogIn form. By checking this property, we will know if this Fill is successful or not, or if a matched username and password has been found from the database.
- I. Otherwise if a matched username and password is found from the LogIn table in the database and the login process is successful, the next window form, SelectionForm, will be displayed to allow users to continue to the next step. After displaying the next form, the current form, LogIn form, should be hidden by calling the Hide() method. The keyword *this* represents the current form.

The coding for the Cancel button Click method is very simple. The Application.Exit() method should be called to terminate our project if this button is clicked by the user.

Before we can test this piece of code by running the project, make sure that the LogIn form has been selected as the Startup form. To confirm this, double click on the Program.cs folder from the Solution Explorer window to open the Main() method. Make sure that the argument of the Application.Run() method is new LogInForm(). This means that a new instance of LogInForm class is created and displayed as this Run() method is executed.

Another important issue is that in order to run this login process properly, make sure to remove the LogInForm_Load() method and its content since a default Fill() method



Figure 5.39 Running status of the project.



Figure 5.40 Warning message.

is included in this method, and both username and password textboxes will have been already filled before the LogIn button is clicked on by the user when the project runs; this method is executed first when the project runs.

Click on the Start button to run the project, and the running LogIn form should match the one shown in Figure 5.39.

Enter a valid username such as “jhenry” to the User Name textbox and a valid password such as “test” to the Pass Word textbox, then click on the LogIn button. The FillByUsernameAndPassword() method will be called to fill the LogIn table in the data source. Because we entered correct username and password, this fill will be successful and the next form, SelectionForm, will be shown up.

Now try to enter a wrong username or password; then click on the LogIn button and a MessageBox will be displayed, which is shown in Figure 5.40, to ask the user to handle this situation.

In this section, we used the LogIn form and LogIn table to show readers how to perform a dynamic data query and fill a mapped data table in the DataSet by using the Visual Studio 2008 design tools and wizards. The coding is relatively simple and easy to follow. In the next section, we will show the readers how to use another method provided by the TableAdapter to pick up a single value from the database.

5.9 USE RETURN A SINGLE VALUE TO QUERY DATA FOR LOGIN FORM

Many people have experienced forgetting either their username or their password when they try to logon to a specific website to get some information, to order some merchandises, or pay bills for their monthly utilities or cell phones. In this section, we show users how to retrieve a single data value from the database. This method belongs to the TableAdapter class.

We still use the LogIn form and LogIn table as an example. Suppose you forget your password, but you want to login to this project by using the LogIn form with your username. By using this example, you can retrieve your password by using your username.

The DataSet Designer allows us to edit the structure of the DataSet. As we discussed in Section 5.6, by using this Designer, you can configure an existing query, add a new query, and add a new column and even a new key to a database. The Add Query method allows us to add a new data query with a SQL SELECT statement, which returns a single value.

Open the LogIn form window from the Solution Explorer window and open the Data Source window by clicking on the Data menu item from the menu bar. Right-click on any place inside that window and select the Edit DataSet with Designer, then locate the LogIn table and right-click on the last line of that table, which contains our modified method FillByUserNamePassWord() we did in the last section. Then select Add Query to open the TableAdapter Query Configuration Wizard.

On the opened wizard, keep the default selection Use SQL Statements, which means that we want to build a query with SQL Statements, then click on the Next button and choose SELECT, which returns a single value radio button. Click on Next to go to the next window and click on the Query Builder button to build our query.

On the opened Query Builder dialog, perform the following operations to create this single data query:

- Click on the first row from the second pane to select it.
- Then right-click on this row and select Delete from the pop-up menu to delete this row.
- Go to the top pane and select the pass_word and user_name columns from the LogIn table by checking two checkboxes related to those two columns.
- Go to the second pane and uncheck the checkbox for the user_name column from the Output column since we do not want to use it as the output, but instead we need to use it as a criterion to filter this query.
- Still in the second pane, right-click on the Group By column and select Delete from the pop-up menu to remove this Group By choice.
- Type a question mark on the Filter field in the user_name column, and press the Enter key on your keyboard. Your finished Query Builder should match the one shown in Figure 5.41.

The SQL statement

```
SELECT pass_word FROM LogIn WHERE (user_name = @Param1)
```

indicates that we want to select a password from the LogIn table based on the username, which is a dynamic parameter, and this parameter will be entered by the user when the project runs. Click on the OK button to go to the next window.

The next window is used to confirm your terminal SQL statement. Click on Next to go to the next window. This window allows you to choose a function name for this query. Change the default name to a meaningful name such as PassWordQuery, then click on the Next button. A successful Wizard Result will be displayed if everything is fine. Click on the Finish button to complete this configuration.

Now let's do our coding for the LogIn form. For testing purposes, we need to add a temporary button with the name = cmdPW and the Text = Password to the LogIn form. Then select and open the LogIn form from the Solution Explorer window, double-click

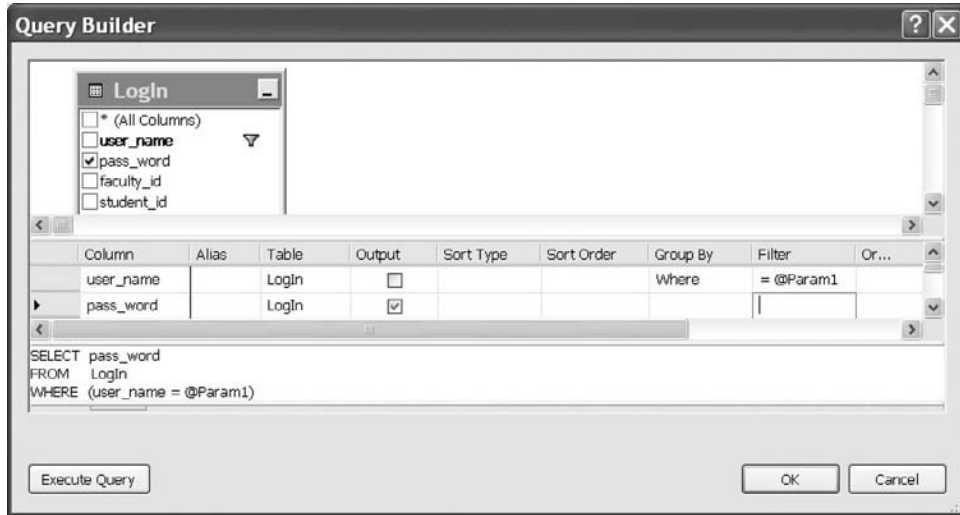


Figure 5.41 Finished Query Builder.

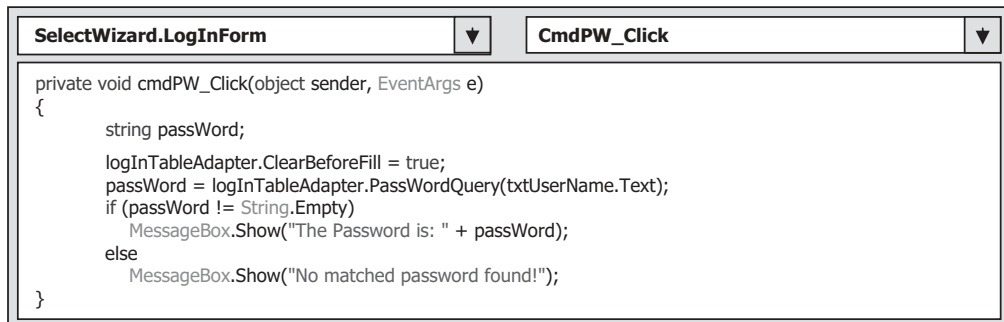


Figure 5.42 Codes for the cmdPW button method.

on the Password button to open its method, and enter the codes shown in Figure 5.42 into this method.

Let's have a little closer look at this piece of code.

- A.** A local string variable `passWord` is created, and it is used to hold the returned queried single value of the `pass_word`.
- B.** The query method we just built in this section, `PassWordQuery()`, with a dynamic parameter `username` that is entered by the user is called to retrieve back the matched `pass_word`.
- C.** If this query found a valid password from the `Login` table based on the username entered by the user, that password will be returned and displayed in a `MessageBox`.
- D.** If this query cannot find any matched `pass_word`, a blank string will be returned and assigned to the variable `passWord`. A `Messagebox` with a warning message will be displayed if this situation happens.



Figure 5.43 Running status of the LogIn form.



Figure 5.44 Returned password.



Figure 5.45 Coding for the getLogInForm method.

Now let's run the project to test this query. Click on the Start button to run the project, and your running project should match the one shown in Figure 5.43.

Enter a username such as "ybai" to the User Name box and click on the PassWord button. The returned password is displayed in a message box, which is shown in Figure 5.44. Well, it looks like a fun! Does it not?

Now you can remove the temporary PassWord button and its method from this LogIn form if you like since we do not need it any more for this project.

Before we can move to the next section, we need to do one more thing, which is to add an accessing method for the LogInForm to allow other methods in the project to access the LogInForm easily when they need to perform some actions on it. For example, to close the LogInForm from other form windows, you need to call this method to access the LogInForm and call its Close() method.

Open the Code Window of the LogInForm and add a new method named getLogInForm() with the code shown in Figure 5.45 into the LogInForm class.

When this method is called, a reference, **this**, is returned to the calling method. Generally, the reference **this** represents the current form object.

In the following sections, we will show the readers how to develop more professional data-driven projects by using more controls and methods. We still use the SelectWizard example project and continue with the SelectionForm.

5.10 CODING FOR SELECTION FORM

As we discussed in Section 5.8, if the login process is successful, the SelectionForm window should be displayed to allow users to continue to the next step. Figure 5.46 shows an opened SelectionForm window.

All information in the ComboBox control is associated with a form window. Furthermore it is associated with a group of data stored in a data table in the database.

The operation steps for this form are summarized as follows:

1. When this form is opened, three pieces of information will be displayed in a ComboBox control to allow users to make a selection to browse the information related to that selection.
2. When the user clicks the OK button, the selected form should be displayed to enable the user to browse the related information.

Based on the operation step 1, the codes for displaying three pieces of information should be located in the constructor of the SelectionForm since this constructor should be called first as an instance of the SelectionForm is created.

Open the SelectionForm window and click on the View Code button to open its code window. Enter the following codes, which are shown in Figure 5.47, into the constructor of the SelectionForm.

Let's see how this piece of code works.

- A. Three instances are created first, and each one is associated with a Form class.
- B. The Add() method of the ComboBox class is used to attach all three pieces of information to this ComboBox. The reference **this** represents the current form object, an instance of the SelectionForm class, and the property SelectedIndex is set to zero to select the first information as the default one.

According to operation step 2 above, when users click on the OK button, the related form selected by the user should be displayed to allow users to browse information from that form. Click the View Designer button to open the graphical user interface of the SelectionForm object. Then double-click on the OK button to open its cmdOK_Click method and enter following codes shown in Figure 5.48 into this method.

Let's see how this piece of code works. The function for this piece of coding is straightforward and easy to be understood, which is explained as follows:

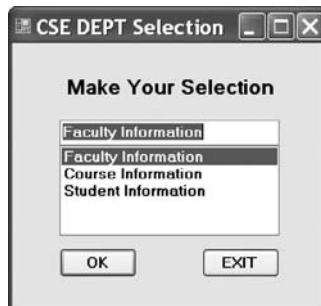


Figure 5.46 Selection Form.


```

SelectWizard.SelectionForm SelectionForm
namespace SelectWizard
{
    public partial class SelectionForm : Form
    {
        FacultyForm facultyForm = new FacultyForm();
        CourseForm courseForm = new CourseForm();
        StudentForm studentForm = new StudentForm();

        public SelectionForm()
        {
            InitializeComponent();
            this.ComboSelection.Items.Add("Faculty Information");
            this.ComboSelection.Items.Add("Course Information");
            this.ComboSelection.Items.Add("Student Information");
            this.ComboSelection.SelectedIndex = 0;
        }
    }
}

```

Figure 5.47 Coding for the Selection Form.

```

SelectWizard.SelectionForm CmdOK_Click()
private void cmdOK_Click(object sender, EventArgs e)
{
    if (this.ComboSelection.Text == "Faculty Information")
        facultyForm.Show();
    else if (this.ComboSelection.Text == "Course Information")
        courseForm.Show();
    else if (this.ComboSelection.Text == "Student Information")
        studentForm.Show();
    else
        MessageBox.Show("Invalid Selection!");
}

```

Figure 5.48 Coding for the OK button Click method.

- A.** Open the FacultyForm window if the user selected Faculty Information.
- B.** Open the StudentForm window if the user selected Student Information.
- C.** Open the CourseForm window if the user selected Course Information.
- D.** An error message is displayed if no information is selected.

The last coding for this form is the Exit button. Open the graphical user interface of the SelectionForm, double-click on the Exit button to open its cmdExit_Click() method. Enter the codes into this method as shown in Figure 5.49.

This piece of code looks a little complicated. Let's see how it works.

- A.** First of all, we must create a new instance of the LogInForm class since we need to close all opened form windows if they are still open before we can exit this project. The point to be noted is that this instance is not one we created and applied in our LogInForm window, but it is a new instance and has no relationship with the one we used before in the LogInForm.

```

SelectWizard.SelectionForm  cmdExit.Click()
private void cmdExit_Click(object sender, EventArgs e)
{
A   LogInForm logForm = new LogInForm();
B   logForm = logForm.getLogInForm();
C   logForm.Close();
D   courseForm.Close();
   facultyForm.Close();
E   studentForm.Close();
   Application.Exit();
}

```

Figure 5.49 Coding for the Exit button Click method.

- B.** In order to access and use our original LogInForm object, we need to call the getLogInForm() method we built in the LogInForm class in the last section, and assign this returned object to our new instance. In this way, we make our new created instance of the LogInForm have the same reference as the original instance of the LogInForm had. Now we can use this instance to access any method attached to the original LogInForm object.
- C.** To close the LogInForm window, the Close() method is called.
- D.** Similarly, the Close() methods attached to other classes, such as FacultyForm, CourseForm, and StudentForm, are executed to close the associated form object. The point is that you do not need to create any new instance for each of those classes since those instances are created in this SelectionForm as the class variables or called fields in Visual C# 2008.
- E.** Finally the system method Exit() is called to terminate the whole project.

Suppose the user selected the first information—Faculty Information. A Faculty form window will be displayed, and it is supposed to be connected to a Faculty data table in the database. If the user selected a faculty name from the ComboBox control and clicked the Select button on that form (refer to Figure 5.21), all information related to that faculty should be displayed on that form, exactly on five labels and a picturebox.

Now let's first see how to perform the data binding to bind controls on the Faculty form to the associated columns in the database.



One of important issues in Object-Oriented Programming is how to access an instance of a class, which has been previously created and used by some other files or classes, by any other class or object. A good solution is to set up a common reference or point inside the class that will be instanced and used in multiple times later in the project. A retrieving or get method is created in that class, too. By using this get method, any other class or object can access this instance.

5.11 BIND DATA TO ASSOCIATED CONTROLS IN FACULTY FORM

Open the Faculty form window from the Solution Explorer window and perform the following data bindings:



Figure 5.50 Expansion for data binding.

1. Select the TitleLabel by clicking on it; then go to the **DataBindings** property, select the **Text** item, and click on the drop-down arrow. Expand the following items:

- Other Data Sources
- Project Data Sources
- CSE_DEPTDataSet
- Faculty

Then select the **title** column from the Faculty table by clicking on it. In this way, we finish the binding between the label control TitleLabel on the Faculty form and the title column in the Faculty table. As soon as this data binding is finished, immediately you can find that three instances are created and displayed under the form: cSE_DEPTDataSet, facultyBindingSource, and facultyTableAdapter.

2. Continue to select the next label from the Faculty Information GroupBox, which is the OfficeLabel, go to the **DataBindings** property and then select the **Text** item, and click on the drop-down arrow. This time you will find that a new object facultyBindingSource is created. As we discussed in Section 5.7, as soon as you finish one data binding, a new object of the data-binding source will be created and served for the form in which the binding source is located. Now we need to use this data-binding source to bind our OfficeLabel control. Expand this binding source until you find the Faculty table, then click the **office** column to finish this binding. An example of this expansion is shown in Figure 5.50.
3. In the similar way, you can finish the data binding for the rest of the three label controls: PhoneLabel, CollegeLabel, and EmailLabel. The binding relationship is PhoneLabel → phone column, CollegeLabel → college column, and EmailLabel → email column in the Faculty table.

Next, we need to use the DataSet Designer to build our data query with the SQL **SELECT** statement involved and modify the name of the FillBy() method for the facultyTableAdapter.

Open the Data Source window by clicking on the Data>Show Data Sources menu item from the menu bar. Right-click on any place inside that window and select **Edit DataSet** with the Designer item to open the DataSet Designer Wizard. Locate the Faculty table, then right-click on the last line of the Faculty table, and select the **Add Query** item from the pop-up menu to open the TableAdapter Configuration Wizard.

On the opened Wizard, click on **Next** to keep the default command type. Use SQL statements and click on another **Next** to keep the default query type **SELECT**, which returns rows for the next dialog. Then click on the **Query Builder** button to open the

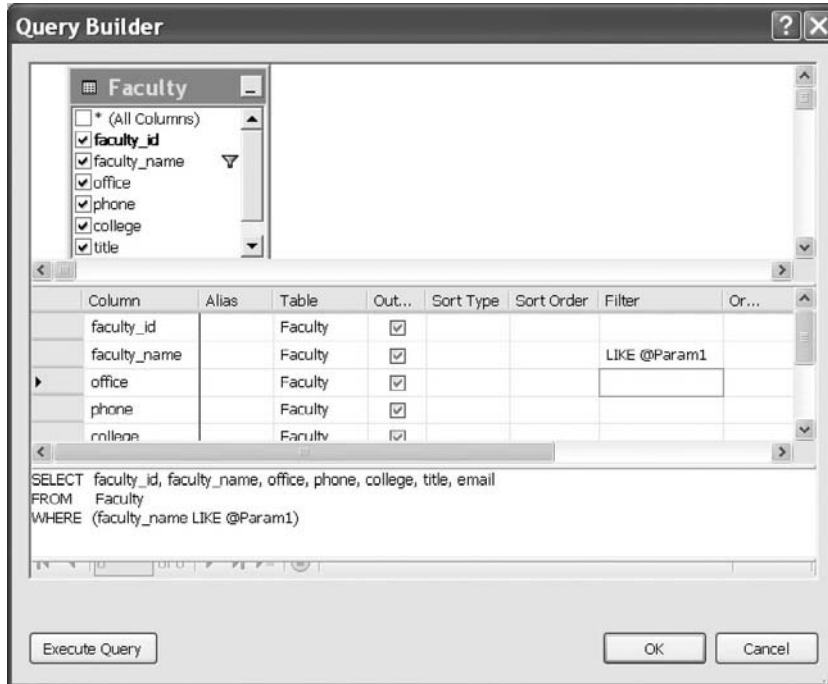


Figure 5.51 Example of the Query Builder.

Query Builder dialog. In the middle graphical pane, move your cursor to the Filter column along the faculty_name line, then type a question mark and press the Enter key on your keyboard. In this way, we add a WHERE clause with a dynamic parameter that is represented by LIKE @Param1 in the SQL Server database. Note that the keyword LIKE is similar to an equals symbol used in the assignment operator in Microsoft Access query. In SQL Server data query, the LIKE is used instead of the equals symbol. Your finished Query Builder should match the one shown in Figure 5.51.

Click on the OK and the Next buttons to modify the name of the FillBy() method. Attach FacultyName to the end of the FillBy, so the modified name for this FillBy() method is FillByFacultyName(). Uncheck the Return a DataTable checkbox since we do not want to return any table. Click on the Next and then the Finish buttons to complete this configuration.

Now let's develop the code for querying the faculty information using this Faculty form with the Faculty data table in the database.

5.12 DEVELOP CODES TO QUERY DATA FROM FACULTY TABLE

In this section, we divide the coding job into two parts. Querying data from the Faculty table using the SQL Select method is discussed in part 1, and retrieving data using the LINQ method is provided in part 2. Furthermore, we only take care of the coding for

the Select and the Back buttons' click methods, and the coding for all other buttons will be discussed and coded in the following sections.

5.12.1 Develop Codes to Query Data Using SQL SELECT Method

As we mentioned above, the pseudocode or the operation sequence of this data query can be described as follows:

- After the project runs, the user has completed the login process and selected the Faculty Information item from the Selection Form.
- The Faculty form will be displayed to allow users to select the desired faculty name from the Faculty Name ComboBox control.
- Then the user can click on the Select button to make a query to the Faculty data table to get all the information related to that desired faculty member.

The main coding job is performed within the Select button click method. But before we can do that coding, we need to add all faculty names into the Faculty Name ComboBox control. In this way, as the project runs, the user can select a desired faculty from that box. Since these faculty names should be displayed first as the project runs, we need to do this coding in the Form_Load method.

In the opened Solution Explorer window, choose the FacultyForm.cs and click on the View Code button to open the code window. On the opened code window, scroll down to find the FacultyForm_Load method. Enter the codes shown in Figure 5.52 into this method.

Let's see how this piece of code works.

- First, we need to use the Add method to add all faculty names into the Faculty Name ComboBox control.

The screenshot shows a code editor window with the title 'SelectWizard.FacultyForm' and 'FacultyForm_Load'. The code is as follows:

```
private void FacultyForm_Load(object sender, EventArgs e)
{
    A      ComboName.Items.Add("Ying Bai");
           ComboName.Items.Add("Satish Bhalla");
           ComboName.Items.Add("Black Anderson");
           ComboName.Items.Add("Steve Johnson");
           ComboName.Items.Add("Jenney King");
           ComboName.Items.Add("Alice Brown");
           ComboName.Items.Add("Debby Angles");
           ComboName.Items.Add("Jeff Henry");
    B      ComboName.SelectedIndex = 0;
    C      this.cmdSelect_Click(this.cmdSelect, null);
    D      ComboMethod.Items.Add("TableAdapter Method");
           ComboMethod.Items.Add("LINQ & DataSet Method");
           this.ComboMethod.SelectedIndex = 0;
}
}
```

Figure 5.52 Coding for the FacultyForm_Load method.

- B.** Then we set the SelectedIndex value to 0, which means that the first faculty name that has an index value 0 has been selected as a default name as the project runs.
- C.** This instruction is very important, and the purpose of this coding is to call the Select button's Click method to perform an SQL SELECT command, which is equivalent to clicking on the Select button by the user as the project runs. The reason we add this code is: The FillByFacultyName() method will not be called and the related faculty information cannot be reflected in five labels in the FacultyForm window when this Select button is clicked on the first time as the FacultyForm is loaded and opened without this instruction. In other words, in the first time the FacultyForm runs, you have to run this instruction to trigger the Select button to perform an SQL SELECT command to retrieve back the information and display it for the default selected faculty in this form.
- D.** Two methods, TableAdapter and LINQ, are added into the ComboMethod Combobox to allow users to select either one to perform the data query. Similarly, the first method, TableAdapter, is selected as the default method by setting up the SelectedIndex property of the ComboMethod to zero.

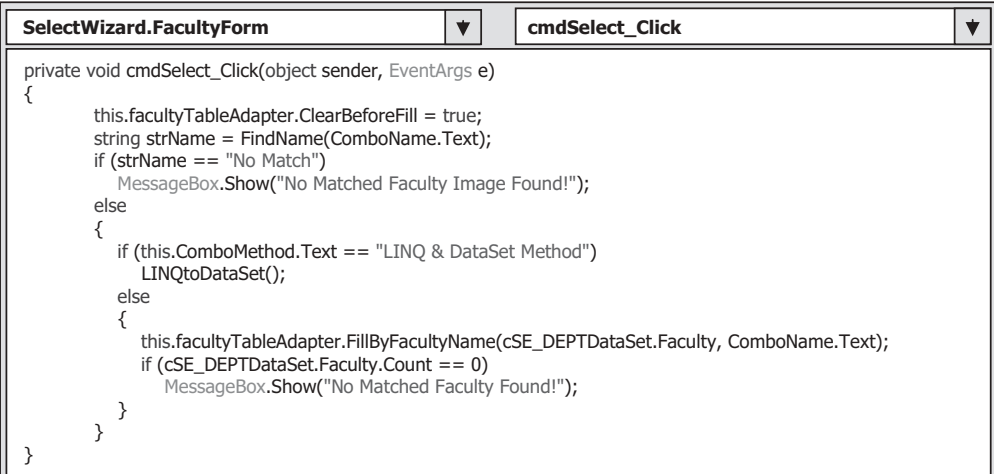
Another point to note is that we do not need to load and fill the Faculty table by using the default coding that is created and added by the system as this method is added into this project; therefore we have already deleted that code.

Now we need to do the coding for the Select button Click method to perform the data query using the SQL SELECT method.

Click on the View Designer button to open the Faculty graphical user interface. On the opened Faculty form, double-click on the Select button to open this method, then enter the codes shown in Figure 5.53 into this method.

Let's see how this piece of code works.

- A.** First, we need to clean up the Faculty table in the DataSet before it can be filled by setting the ClearBeforeFill property to True.
- B.** Next, we need to call a user-defined method FindName(), which will be developed in Section 5.13, to identify and display the selected faculty image in the faculty PictureBox control.



```

SelectWizard.FacultyForm  cmdSelect_Click
private void cmdSelect_Click(object sender, EventArgs e)
{
    A      this.facultyTableAdapter.ClearBeforeFill = true;
    B      string strName = FindName(ComboName.Text);
    C      if (strName == "No Match")
           MessageBox.Show("No Matched Faculty Image Found!");
    D      else
           {
    E          if (this.ComboMethod.Text == "LINQ & DataSet Method")
               LINQtoDataSet();
           else
           {
    F              this.facultyTableAdapter.FillByFacultyName(cSE_DEPTDataSet.Faculty, ComboName.Text);
    G              if (cSE_DEPTDataSet.Faculty.Count == 0)
                   MessageBox.Show("No Matched Faculty Found!");
           }
           }
}

```

Figure 5.53 Coding for the Select button Click method.

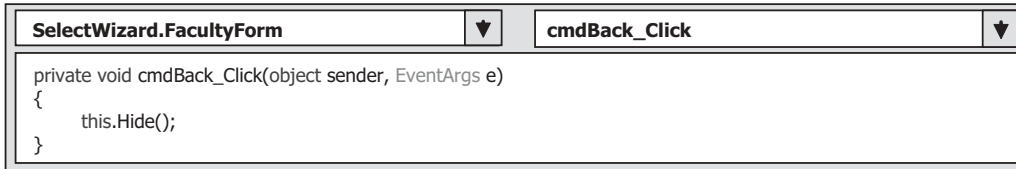


Figure 5.54 Coding for the Back button.

- C. A warning message will be displayed if no matched faculty image can be found.
- D. Otherwise, the FindName() method is executed successfully, and a matched faculty image is displayed in the faculty PhotoBox.
- E. Before we can continue to perform this data query, we need to check which method has been selected. If the LINQ method is selected, a user-defined method LINQtoDataSet() that will be developed below is called to retrieve back the faculty information in LINQ method.
- F. Otherwise, the TableAdapter method is selected and the method FillByFacultyName() we built in Section 5.11 is called to fill the Faculty table with a dynamic parameter, which is selected by the user from the Faculty Name ComboBox control as the project runs.
- G. By checking the Count property of the Faculty table that is in our DataSet, we will know whether this fill is successful or not. If this property is equal to 0, which means that no matched record has been found from the Faculty table in the database, and therefore no record or data has been filled into the Faculty table in our DataSet, a warning message is given for this situation to require users to handle this problem. The user can either continue to select correct faculty name or exit the project. If this property is nonzero, which indicates that this fill is successful and a matched faculty name is found, the Faculty table in our DataSet has been filled. All information related to the matched faculty will be displayed in the five labels and a picturebox.

The coding for the Back button Click method is very simple. The Faculty form will be hidden when this button is clicked. A Hide() method is used for this purpose, which is shown in Figure 5.54.

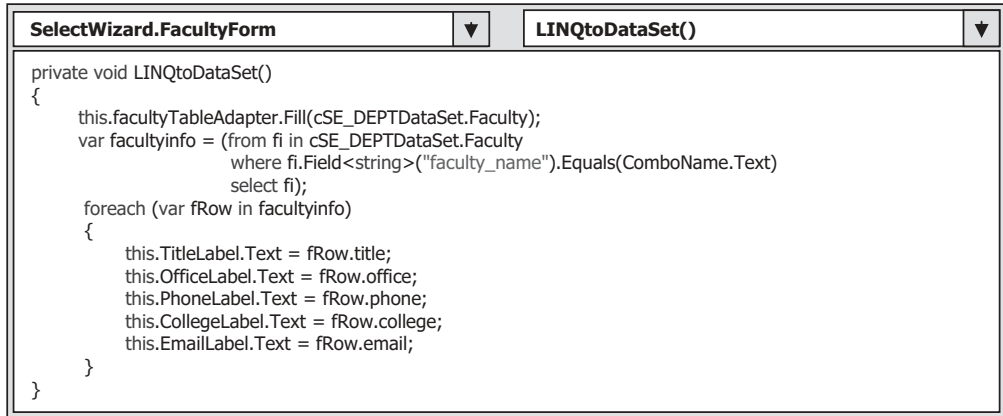
5.12.2 Develop Codes to Query Data Using LINQ Method

The LINQ query technique is new in Visual Studio 2008. The query process can be significantly integrated and improved by using this technology. We have already provided a very detailed discussion about this technology in Chapter 4. Refer to Chapter 4 to get a clear picture of this issue. In this chapter, we will concentrate on the coding for this method.

Open the Code Window of the FacultyForm if it is not opened, create a user-defined method, and enter the code shown in Figure 5.55 into this method.

Let's see how this piece of code works.

- A. First, the default Fill() method of the facultyTableAdapter is executed to load data from the Faculty table in the database into the Faculty table in our DataSet. This step is necessary since the LINQ technique is applied with the DataSet, and the DataSet must contain the valid data in all tables before this technique can be implemented.



```

SelectWizard.FacultyForm LINQtoDataSet()
private void LINQtoDataSet()
{
    this.facultyTableAdapter.Fill(cSE_DEPTDataSet.Faculty);
    var facultyinfo = (from fi in cSE_DEPTDataSet.Faculty
                      where fi.Field<string>("faculty_name").Equals(ComboName.Text)
                      select fi);
    foreach (var fRow in facultyinfo)
    {
        this.TitleLabel.Text = fRow.title;
        this.OfficeLabel.Text = fRow.office;
        this.PhoneLabel.Text = fRow.phone;
        this.CollegeLabel.Text = fRow.college;
        this.EmailLabel.Text = fRow.email;
    }
}

```

Figure 5.55 Coding for the LINQ method.

- B.** A typical LINQ query structure is created and executed to retrieve back all related information for the selected faculty member. The `facultyinfo` is a C# 2008 implicitly typed local variable with a data type `var`. The C# 2008 will be able to automatically convert this `var` to any suitable data type; in this case, it is a `DataSet`, when it sees it. An iteration variable `fi` is used to iterate over the result of this query from the `Faculty` table. Then a similar SQL `SELECT` statement is executed with the `WHERE` clause.
- C.** The `foreach` loop is utilized to pick up each column from the selected data row `fRow`, which is obtained from the `facultyinfo` we get from the LINQ query.
- D.** Assign each column to the associated label to display them in the `FacultyForm` window.

At this point, we have almost completed the coding for this form. Before we can test our project, we need one more step.

5.13 DISPLAY PICTURES FOR FACULTY FORM

To store images in the database is not an easy job. In this section, to simplify this process; we just save the faculty images in a special folder in our computer. We can load a picture into our project to show it as our project runs.

To display the correct faculty photo from the correct location, we need to perform the following steps to configure this operation:

- In order to make this project portable, which means that the project can be executed as an integrated body without any other additional configurations, the best place to save these faculty images is a folder in which your Visual C# 2008 executable file is stored. The exact folder is dependent on your output file type. The folder should be `your_project_folder\bin\Debug` if your output file is a debug file, otherwise you should save those faculty images in the folder `your_project_folder\bin\Release` if your output file is a release file. In this application, our output file is a debug file; therefore, save those faculty images into the folder `SelectWizard\bin\Debug`. You do not need to specify the full path for those images' location if you save images in this way when you load them as the project runs.

- Go to the Images folder at the accompanying ftp site (See Chapter 1) to get all faculty and student images used for this project. You can copy all of those images and paste them into your local folder, such as SelectWizard\bin\Debug.
- In order to select the correct faculty image based on the faculty selected by the user, a function should be developed to complete this function.
- To display the image, a system method, `System.Drawing.Image.FromFile()`, is used.

Now let's develop a method to select the matched image for the faculty selected by the user and display it. The input parameter should be a faculty name and the output should be a name of the matched faculty image.

Keep the FacultyForm window selected, click on the View Code button from the Solution Explorer window to open its code window. Create a new method `FindName()` by entering the code shown in Figure 5.56 into this method.

```

private string FindName(string fName)
{
    string strName;
    switch (fName)
    {
        case "Black Anderson":
            strName = "Anderson.jpg";
            break;
        case "Ying Bai":
            strName = "Bai.jpg";
            break;
        case "Satish Bhalla":
            strName = "Satish.jpg";
            break;
        case "Steve Johnson":
            strName = "Johnson.jpg";
            break;
        case "Jenney King":
            strName = "King.jpg";
            break;
        case "Alice Brown":
            strName = "Brown.jpg";
            break;
        case "Debby Angles":
            strName = "Angles.jpg";
            break;
        case "Jeff Henry":
            strName = "Henry.jpg";
            break;
        default:
            strName = "No Match";
            break;
    }
    if (strName != "No Match")
    {
        PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage;
        PhotoBox.Image = System.Drawing.Image.FromFile(strName);
    }
    return strName;
}

```

Figure 5.56 Codes for the method `FindName`.

Let's see how this piece of code works.

- A. A local String variable `strName` is created to hold the selected image file name.
- B. The Switch Case structure is used to choose the matched faculty image file. A string "No Match" is returned if no matched faculty image is found.
- C. If the content of the variable `strName` is not equal to "No Match", which means that a valid faculty image is found, then the system drawing method, `FromFill()`, is executed to display that faculty image in the Faculty picturebox control.
- D. Finally, the `strName` variable is returned to the calling function.

Now we are ready to test our project. Click the Build/Build Solution menu item to build and link our project, and click on the Start button to run the project. Enter `ybai` as the username and `reback` as the password on the LogIn form. Click on the LogIn button to open the Selection Form window, select the Faculty Information item, and then click on the OK button to open the Faculty form. Select **Ying Bai** from the Faculty Name ComboBox, and click on the Select button. All information related to this faculty with a faculty picture will be displayed, as shown in Figure 5.57.

Remember that you must save all faculty image files into the folder in which your project's executable file is located in order to make your project work properly. In this application, this folder is `C:\SelectWizard\SelectWizard\bin\Debug`.

At this point, we complete the designing and building of our Faculty form. Next we will take care of our Course form.



In this example, we saved our faculty image file in the folder in which the project's executable file is stored. If you do not want to save your image file in this folder, you must provide the full name for your image file, including the full path for the folder in which you saved your image file and the image file name. For instance, one image file `Bai.jpg` is saved in the folder `C:\FacultyImage`. You must give the full name as the returned string as `C:\FacultyImage\Bai.jpg`.

Figure 5.57 Running status of the Faculty form window.

5.14 BINDING DATA TO ASSOCIATED CONTROLS IN COURSE FORM

The functions of this form are illustrated in the following steps:

1. This form allows users to find the course taught by the selected faculty from the Faculty Name ComboBox control when users click on the Select button. The courses (basically all course_id) are displayed in the Course ListBox.
2. The detailed information for each course such as the course title, course schedule, classroom, credits, and enrollment can be obtained by clicking the desired course_id from the Course ListBox, and displayed in five TextBox controls.
3. The Back button allows users to return to the Selection form to make other selections to obtain desired information related to that selection.

In this section, we only take care of two buttons, the Select and the Back buttons, and the coding for the Insert button will be discussed in the following chapters.

For step 1, in order to find the courses taught by the selected faculty from the Course table, we need first to obtain the selected faculty ID associated with the selected faculty from the Faculty Name Combobox control when users click on the Select button because no faculty name is available from the Course table. The only available information in the Course table is the faculty_id. So we need first to create a query that returns a single value (faculty_id) from the Faculty table, and then we will create another query in the Course table to find the courses taught by the selected faculty based on the faculty_id we obtained from the Faculty table.

Now let's do the first job, to create a query to obtain the associated faculty_id from the Faculty table based on the selected faculty from the Faculty Name Combobox in the Course form.

Open the DataSet Designer Wizard and right-click on the last line of the Faculty table and select AddQuery to open the TableAdapter Query Configuration Wizard window, keep the default selection Use SQL statements, and click on the Next button to go to the next window. Check the radio button in front of SELECT, which returns a single value to choose this query type, and click on the Next button to go the next dialog. Click the Query Builder to build our query.

Perform the following operations to complete this query building:

- Click on the first row from the second pane to select it.
- Then right-click on this row and select Delete from the pop-up menu to delete this row.
- Go to the top pane and select the faculty_id and faculty_name columns from the Faculty table by checking two checkboxes related to those two columns.
- Go to the second pane and uncheck the checkbox for the faculty_name column from the Output column since we do not want to use it as the output, but instead we need to use it as a criterion to filter this query.
- Still in the second pane, right-click on the Group By column and select Delete from the pop-up menu to remove this Group By choice.
- Type a question mark on the Filter field in the faculty_name column, and press the Enter key on your keyboard. Your finished Query Builder should match the one shown in Figure 5.58.

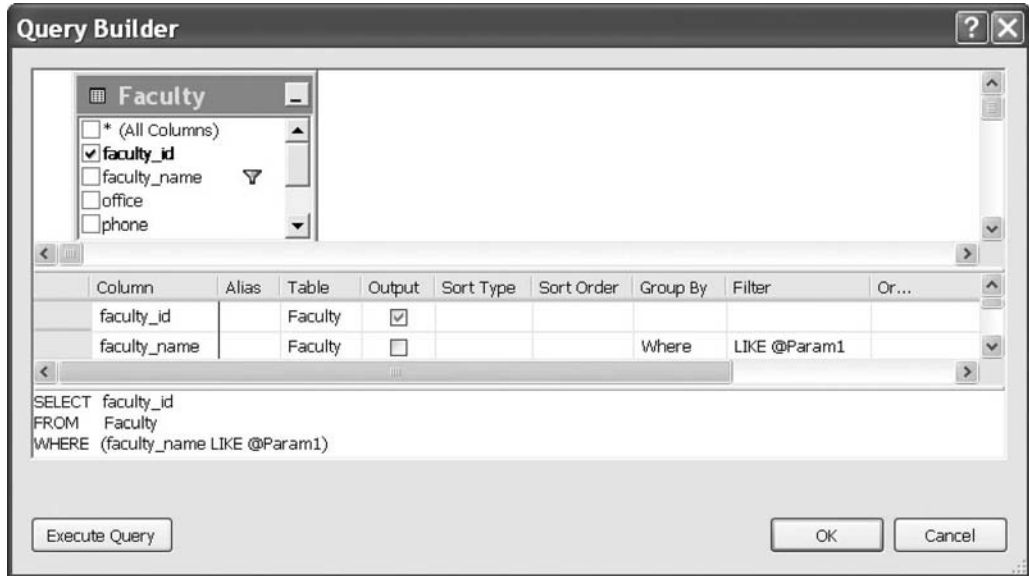


Figure 5.58 Finished query for the faculty_id.

The SQL statement shown in the text pane or the third pane is:

```
SELECT faculty_id FROM Faculty WHERE (faculty_name LIKE @
Param1)
```

Click on the OK and the Next buttons to continue to the next window. Enter the FindFacultyIDByName into the box as our function name and then click on the Next and the Finish buttons to complete this query building.

Now let's continue to build our query to find the courses taught by the selected faculty from the Course table. Open the DataSet Designer to create our desired query and modify the Fill() method for the CourseTableAdapter.

Open the Data Source window by clicking the Data>Show Data Sources menu item from the menu bar. Then right-click on any place inside this window and select the Edit DataSet with Designer item to open the DataSet Designer Wizard. Right-click on the last line of the Course table and choose the Add Query item to open the TableAdapter Query Configuration Wizard. Then click the Query Builder to open the Query Builder window, which is shown in Figure 5.59.

Keep the default selections for the top graphical pane even we only need the course_id column, and we will show you why we need to keep this default item later. Go to the Filter column along the faculty_id row, and type a question mark and press the Enter key on your keyboard. This is equivalent to setting a dynamic parameter for this SQL SELECT statement. The completed SQL statement is displayed in the text pane and the content of this statement is:

```
SELECT course_id, course, credit, classroom, schedule,
enrollment, faculty_id
FROM Course
WHERE (faculty_id = @Param1)
```

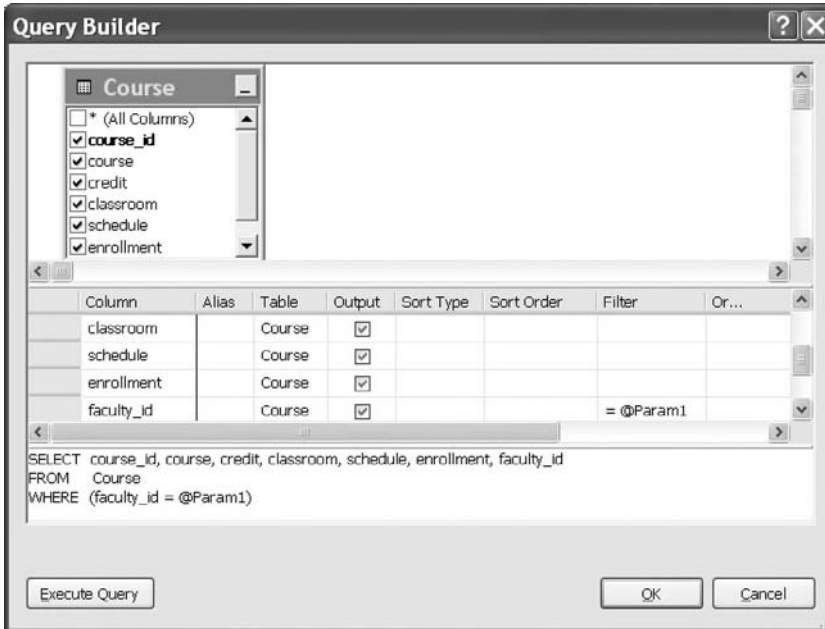


Figure 5.59 Query Builder.

The dynamic parameter @Param1 is a temporary parameter, and it will be replaced by the real parameter, faculty_id, as the project runs.

Click on OK and then the Next button to return to the TableAdapter Query Configuration Wizard to modify the Fill() method. Attach ByFacultyID to the end of the Fill() method to get a modified method: FillByFacultyID(). Uncheck the Return a DataTable checkbox since we do not need this method for this query. Then click on the Next and the Finish buttons to complete this configuration.

The next step is to bind five textbox controls in the CourseForm to the associated data column in the Course table in the DataSet. Select the CourseForm.cs from the Solution Explorer window and click on the View Designer button to open the CourseForm's graphical user interface.

First, we need to bind the CourseList to the course_id column in the Course table in the DataSet. Recall that there are multiple course_id with the same faculty_id in this Course table when we built our sample database in Chapter 2. Those multiple records with the same faculty_id are distinguished by the different courses taught by that faculty. To bind a ListBox to those multiple records with the same faculty_id, we cannot continue to use the binding method we used for label or textbox controls in the previous sections. This is the specialty of binding a ListBox control. The special point is that the relationship between the ListBox and the data columns in a table is one-to-many, which means that a ListBox can contain multiple items; in this case, the CourseList can contain many course_ids. So the binding of a ListBox control is exactly to bind a ListBox to a table in the DataSet, exactly to the Course table in this application.

To do this binding, click the CourseList control from the Course form, go to the DataSource property, and click on the drop-down arrow to expand the data source until

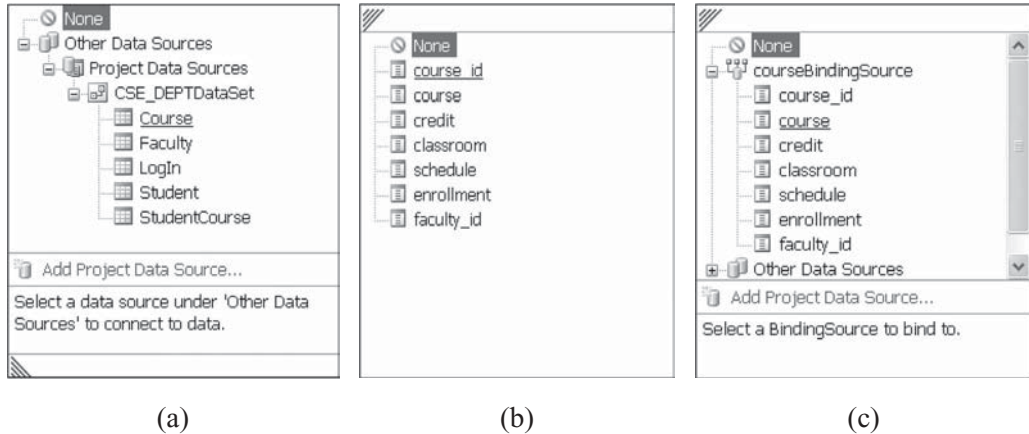


Figure 5.60 Expansion of the data source.

the Course table is found. Select this table by clicking it. Figure 5.60a shows this expansion situation.

Continue this binding by going to the DisplayMember property and expand the Course table to find the course_id column, select it by clicking on this item (Figure 5.60b). In this way, we set up a binding relationship between the Course ListBox in the Course form and the Course data table in the DataSet.

To execute step 2, we need to bind five textbox controls in the Course form to five columns in the Course data table in the DataSet. To do this binding, still keep the Course form open, and then select the Course textbox from the Course Information GroupBox control. Go to the DataBindings property and expand to the Text item, click on the drop-down arrow, and you will find that a CourseBindingSource object is already created there for this project. Expand this CourseBindingSource until you find the course column, which is shown in Figure 5.60c, and then choose it by clicking on the course column. In this way, a binding is set up between the Course textbox in the Course form and the course column in the Course table in the DataSet.

In a similar way, finish the binding for the other four textbox controls: Schedule, Classroom, Credits, and Enrollment. Note the order in which to perform these two bindings. You must first perform the binding for the CourseList control, and then perform the binding for the five textboxes.

Now we can answer the question of why we need to keep the default selections at the top graphical pane when we build our query in the Query Builder (refer to Figure 5.59). The reason for this is that we need those columns to perform data binding for our five textbox controls here. In this way, each textbox control in the Course form is bound with the associated data column in the Course table in the DataSet. After this kind of binding relationship is set up, all data columns in the Course table in the DataSet will be updated by the data columns in the Course table in the real database each time when a FillByFacultyID() method is executed. At the same time, the content of all five textboxes will also be updated since those textbox controls have been bound to those data columns in the Course table in the DataSet. Now, it is time for us to write the code for this form.

5.15 DEVELOP CODES TO QUERY DATA FOR COURSE FORM

Based on the analysis of the function of the Course form we did above, when the user selected a faculty name and clicked on the Select button, all courses, basically all course_id, taught by that faculty should be listed in the Course ListBox. To check the details for each course, click on the course_id from the CourseList control and all detailed information related to the selected course_id will be displayed in five textbox controls. The coding is divided into two parts. The first part is to query data using the TableAdapter method, and the second part is to perform the data query using the LINQ method.

5.15.1 Query Data from the Course Table Using TableAdapter Method

Open the CourseForm window and click on the View Code button from the Solution Explorer window to open the code window. Remove all default codes and enter the codes into the CourseForm_Load() method shown in Figure 5.61.

This coding is straightforward with no tricks. The Add method is used to add all faculty names into the faculty name combobox and add two methods into the method combobox. To reset the SelectedIndex property to 0 is to select the first faculty and the first method as the default from the combobox as the project runs.

Open the CourseForm window by clicking on the View Designer button from the Solution Explorer window, and then double-click on the Select button to open its Click method. Enter the codes into this method shown in Figure 5.62.

Let's see how this piece of code works.

- A. A new faculty table adapter object is created based on the FacultyTableAdapter class located at the namespace CSE_DEPTDataSetTableAdapters. We need this object to access the Faculty table to retrieve back the faculty_id based on the selected faculty name.
- B. A local string variable strFacultyID is declared, and it is used to hold the returned faculty_id when our built query FindFacultyIDByName() is executed.

```

private void CourseForm_Load(object sender, EventArgs e)
{
    this.ComboName.Items.Add("Ying Bai");
    ComboName.Items.Add("Satish Bhalla");
    ComboName.Items.Add("Black Anderson");
    ComboName.Items.Add("Steve Johnson");
    ComboName.Items.Add("Jenney King");
    ComboName.Items.Add("Alice Brown");
    ComboName.Items.Add("Debby Angles");
    ComboName.Items.Add("Jeff Henry");
    ComboName.SelectedIndex = 0;
    ComboMethod.Items.Add("TableAdapter Method");
    ComboMethod.Items.Add("LINQ & DataSet Method");
    this.ComboMethod.SelectedIndex = 0;
}

```

Figure 5.61 Coding for the CourseForm_Load method.

```

SelectWizard.CourseForm cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
  A   CSE_DEPTDataSetTableAdapters.FacultyTableAdapter FacultyTableApt =
      new CSE_DEPTDataSetTableAdapters.FacultyTableAdapter();

  B   string strFacultyID = FacultyTableApt.FindFacultyIDByName(ComboName.Text);
  C   if (strFacultyID != string.Empty)
  {
  D       if (this.ComboMethod.Text == "LINQ & DataSet Method")
          LINQtoDataSet(strFacultyID);
  E       else
          {
  F           this.courseTableAdapter.FillByFacultyID(cSE_DEPTDataSet.Course, strFacultyID);
              if (cSE_DEPTDataSet.Course.Count == 0)
                  MessageBox.Show("No Matched Courses Found!");
          }
  G       }
          else
              MessageBox.Show("No matched faculty_id found!");
  }
}

```

Figure 5.62 Coding for the Select button Click method.

- C.** If the returned faculty_id is not an empty string, which means that a valid faculty_id has been obtained, then we can continue to perform the following faculty courses query based on this faculty_id.
- D.** If the user selected the LINQ to DataSet method to perform this query, a user-defined method LINQtoDataSet() is called to perform this data action. The argument is the faculty_id obtained from the first query we did above.
- E.** Otherwise the TableAdapter method is chosen by the user, and the query we built in the DataSet Designer, FillByFacultyID(), is called to fill the Course table in our DataSet using the dynamic parameter @Param1, which is replaced by our real parameter strFacultyID now.
- F.** To check whether this fill is successful, the Count property of the Course table is detected. If this property is reset to 0, which means that no any data is filled into our Course table in our DataSet, the fill fails and a warning message will be displayed to require users to handle this situation. Otherwise, the fill is successful and all courses, basically all course_id, taught by the selected faculty will be filled into the Course table and loaded into the Course ListBox control in the Course form. Furthermore, the detailed course information including the course title, course schedule, classroom, credits, and enrollment for the selected course_id in the Course ListBox will be displayed in the five textbox controls since these five textbox controls have been bound to those related columns in the Course table.
- G.** If the returned faculty_id is an empty string, which means that no valid faculty_id can be found from the Faculty table, a warning message is displayed to indicate this situation.

Return to the Course form window by clicking on the View Designer button from the Solution Explorer window, then double-click on the Back button to open its Click method and enter the code this.Hide() into this method.

That's it! The coding for the data query using the TableAdapter is done. Next let's take care of the coding for the data query using the LINQ to dataset method.

5.15.2 Query Data from the Course Table Using LINQ Method

In the coding we did in the last section (refer to Figure 5.62), the program will be directed to the LINQtoDataSet() method if the user selected the LINQ method from the Method combobox. Refer to Chapter 4 for a detailed discussion about the data query between LINQ to DataSet. In this section, we will develop the code to use this method to perform the data query from the Course table in our DataSet.

Open the Code Window of the CourseForm if it is not opened, create a new method named LINQtoDataSet(), and enter the code shown in Figure 5.63 into this method.

Let's see how this piece of code works.

- A. First, we need to call the query method FillByFacultyID() we built in Section 5.14 to load course data from the Course table in the database to the Course table in our DataSet. This operation is important and necessary for our next action, retrieving data from the Course table in the DataSet and display them in the CourseList box.
- B. A typical LINQ query structure is created and executed to retrieve back all related course information for the selected faculty member. The courseinfo is a C# 2008 implicitly typed local variable with a data type var. The C# 2008 will be able to automatically convert this var to any suitable data type, in this case, it is a DataSet, when it sees it. An iteration variable ci is used to iterate over the result of this query from the Course table. Then a similar SQL SELECT statement is executed with the WHERE clause.
- C. The foreach loop is utilized to pick up each column from the selected data row cRow, which is obtained from the courseinfo we get from the LINQ query. Assign each column to the associated textbox to display them in the CourseForm window.

A trick exists in this assignment, which is the last running status of the foreach loop. When the foreach loop is done, the course title column stored in the iteration variable cRow (cRow.course) will contain the last course title even if we want to assign the first course title to this variable. To avoid this error, the first assignment must not be contained in this loop. Therefore, we should comment out the first assignment. But how can we assign the course title column to the Course textbox in the CourseForm window? This job can be handled by the BindingSource as you click and select the desired course_id from the CourseList box automatically.

```

SelectWizard.CourseForm LINQtoDataSet()
private void LINQtoDataSet(string facultyID)
{
    this.courseTableAdapter.FillByFacultyID(cSE_DEPTDataSet.Course, facultyID);
    var courseinfo = (from ci in cSE_DEPTDataSet.Course.AsEnumerable()
                     where ci.Field<string>("faculty_id") == facultyID
                     select ci);
    foreach (var cRow in courseinfo)
    {
        //this.txtName.Text = cRow.course;
        this.txtSchedule.Text = cRow.schedule;
        this.txtClassRoom.Text = cRow.classroom;
        this.txtCredits.Text = cRow.credit.ToString();
        this.txtEnroll.Text = cRow.enrollment.ToString();
    }
}

```

Figure 5.63 Coding for the LINQ to DataSet method.

Figure 5.64 Running status of the Course form.

At this point, we have completed all the coding for the CourseForm. Now let's test our project by running it. Click on the Start button to run our project. Enter `ybai` and `reback` as the username and password for the LogIn form, and then select the Course Information from the Selection ComboBox, click on OK to open our Course form, which is shown in Figure 5.64.

On the opened CourseForm window, select the default faculty name Ying Bai and click on the Select button to load and fill all courses (`course_id`) taught by this faculty into the Course table in our DataSet as well as the Course ListBox in this form. The filled `course_id` is displayed in the Course ListBox, as shown in Figure 5.64.

Now let's go one more step forward by just clicking on `course_id` from the Course ListBox. Immediately the detailed information about that selected course, including the course, schedule, classroom, credits, and enrollment, will be displayed in the five textbox controls. This makes sense since those five textbox controls have been bound to those five associated columns in the Course table in our DataSet. As you click on one `course_id` from the Course ListBox, effectively you selected and picked up one course record from the Course table. Recall that the Course ListBox is bound to the Course table in our DataSet by using the CourseBindingSource when we perform this data binding in Section 5.14. For the selected course record, five columns of that record have been bound to the five textbox controls in the form, so the data related to those five columns will also be reflected on these five textbox controls. These relationships can be represented and illustrated by connections in Figure 5.65.

You can try to select the LINQ to DataSet method to perform this data query, and the same querying result will be obtained without any problems. This is very interesting, is not it?

Yes! This is the power provided by Visual Studio 2008. By using these Design Tools and Wizards, it is very easy to develop a professional database programming in the Visual C# 2008 environment, and it is fun to develop a database programming in the template of Windows applications.

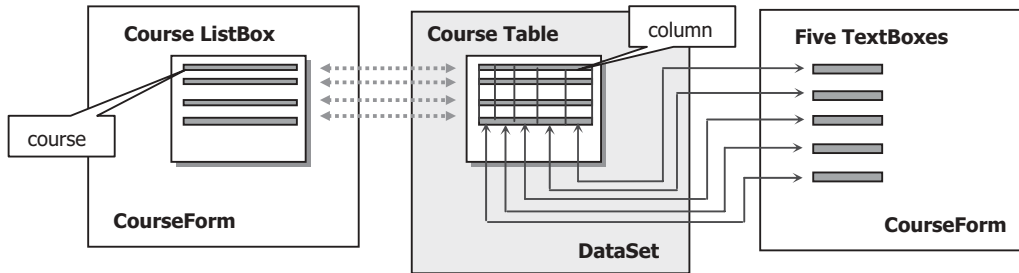


Figure 5.65 Relationships between Course ListBox, Course table, and TextBox.

We have the last form, which is the StudentForm, and we want to leave this as homework for students to allow them to finish developing the data connection and operation between the Student form and the Student table as well as the StudentCourse table. For your reference, a completed project named SampleWizards Project that contains the coding for the Student form has been developed, and you can get it from the folder Chapter 5\SampleWizards Solution, which is located at the folder DBProjects at the accompanying ftp site (See Chapter 1). The database used for that project is Microsoft Access 2007.

The completed coding for this SelectWizard project, including the source codes, graphical user interface designs, Data Source, and Query Builders, can be found at the accompanying ftp site (See Chapter 1) at the folder DBProjects\Chapter 5.

5.16 BUILD A SAMPLE DATABASE PROJECT—SELECTWIZARDORACLE WITH ORACLE DATABASE

Basically, there is no significant difference between building a C# project with SQL Server database and another C# project with Oracle database using the Design Tools and Wizards. There is a small difference when creating and connecting to the different data sources used in the applications. All other elements, including the coding, graphic user interface, and query building, are identical and can be used interchangeably. For this reason, we only concentrate on the data source selection and connection for a new sample project, SelectWizardOracle, with the Oracle Database 10g XE as the data source.

5.16.1 Create a New Visual C# Project—SelectWizardOracle

Now let's create a new Visual C# project and connect it with the Oracle database we built in Chapter 2. Open Visual Studio 2008 and go to the File\New\Project menu item to open the New Project dialog. Select Visual C#\Windows from the Project types pane and Windows Forms Application item from the Templates pane. Enter SelectWizardOracle into the Name box as the name for this project. Select any desired location or folder for the Location box to save this project. In our case, it is C:\Book6\Chapter 5. Keep all other default setting values unchanged, and click on the OK button to create this new project.

Refer to Sections 5.3.1.1 to 5.3.1.5 to build the five graphic user interfaces, LogIn, Selection, Faculty, Course, and Student. To save time, you can copy controls from each

form in the project SelectWizard we built in Section 5.3 and paste them to the new associated form created for this project. The operational sequence is:

1. Create new forms such as LogIn, Selection, Faculty, Course, and Student in the current project SelectWizardOracle.
2. Open the project SelectWizard we built in Section 5.3 and a desired form window, such as the LogIn.
3. Go to Edit>Select All menu item to select all controls from the LogIn form.
4. Go to Edit|Copy menu item to copy all controls in the LogIn form window.
5. Return or reopen the current project SelectWizardOracle and the LogIn form window.
6. Go to Edit|Paste menu item to paste all copied controls to this form.

Similarly, copy all controls for all other forms in this sequence.

Note that when you copy and paste these controls, you also copy and paste the BindingSource for each form. We need to delete these BindingSources since we do not need those old bindings and we need to create our new bindings in this new project. Therefore, just right-click on those BindingSources and click on the Delete item from the pop-up menu to delete them.

Another point is that you may change the TabIndex values for those pasted controls since the TabIndex values will be modified after you paste them into a new form. Go to View|Tab Order menu item to perform this modification.

5.16.2 Select and Add Oracle Database 10g XE as Data Source

In the opened SelectWizardOracle project, open the Data Source window by selecting Data>Show Data Sources menu item. Click on Add New Data Source link to open the Data Source Configuration Wizard, keep the default Database selection unchanged, and click on the Next button to open the Add Connection dialog. Click on the New Connection button since we need to create a new connection between our project and the Oracle database.

Click on the Change button for the Data Source box to open the Change Data Source dialog, and then select Oracle Database as our new data source. Click on the OK button to return to the Add Connection dialog. Enter the following items into the associated boxes for our Oracle database:

- Server name XE
- User name CSE_DEPT
- Password reback

Recall that in Chapter 2, we built an Oracle database named CSE_DEPT with the password reback using Oracle Database 10g Release 2. Refer to Section 2.11 in Chapter 2 to get more detailed information for the definitions of these items. Your finished Add Connection dialog should match one that is shown in Figure 5.66a.

Click on the Test Connection button to confirm this connection. A Connection succeeded message should be displayed if the connection is fine. Click on the OK button to go to the next dialog.

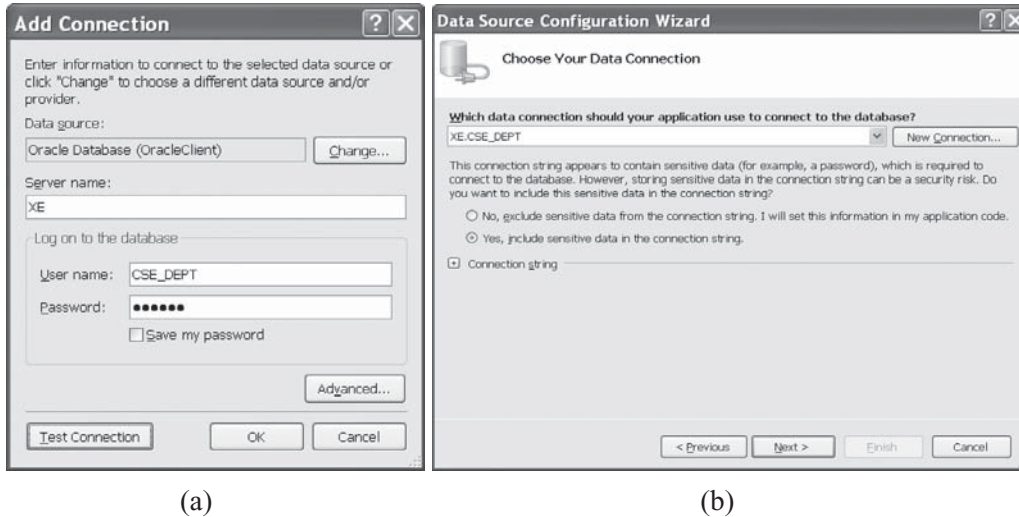


Figure 5.66 Add Connection and Data Source Configuration Wizard.

The next dialog allows you to check and confirm the connection string. Click on the Yes button, which will allow us to add the username and password into this connection string to make this connection as an integrated body (Figure 5.66b). Click on the Next button to open the next dialog. Change the name of this connection string to ConnStringOracle and click on the Next button.

The Next dialog allows us to select the database objects. Generally, we always use data tables to save our data. So click on the small plus icon before the Table object to expand it. By default, quite a few data tables can be selected as our data table objects and most of them are built by the database vendors. For our application, we only need five of the tables we created in Section 2.11 in Chapter 2, such as the LogIn, Faculty, Course, Student, and StudentCourse. Select all of these five tables by checking each of checkboxes one by one. Another issue is the name of the DataSet. In order to match and use codes we developed in the project SelectWizard, change this DataSet name to CSE_DEPTDataSet by modifying the content of the DataSet name box. Your finished dialog should match the one shown in Figure 5.67.

Click on the Finish button to complete this database connection. Immediately you can find that five tables have been added into the Data Source window, which is shown in Figure 5.68.

After this data source connection is complete, you can use all the codes we developed in the project SelectWizard in this project. The following issues must be addressed in order to successfully develop this project by using codes from the project SelectWizard:

- When you perform the copy-and-paste operations for those codes located inside different Click methods, you need first to open those methods by double-clicking on the associated buttons from each form window, and then you can paste codes into that method. In other words, you cannot copy and paste the whole body of those methods including the method's header and ender, but you can copy and paste only the content of each method.
- You need to build each query method using the Query Builder one by one in order to perform the data query. These methods include the PassWordQuery() and FillBy

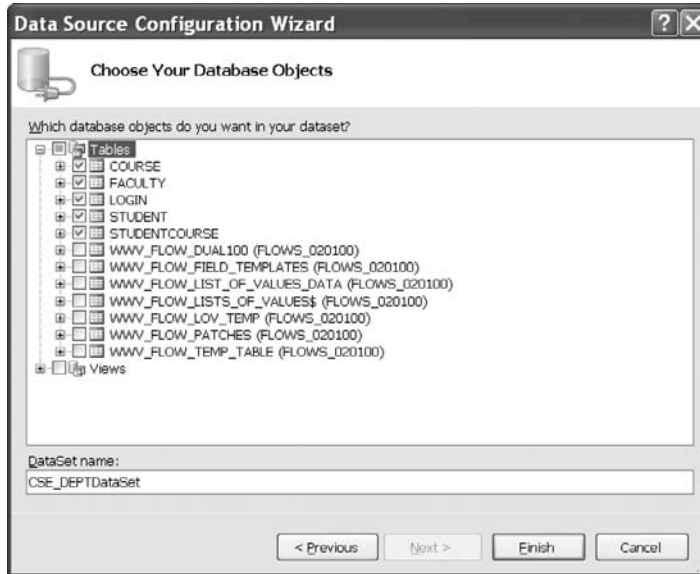


Figure 5.67 Database object selection dialog.

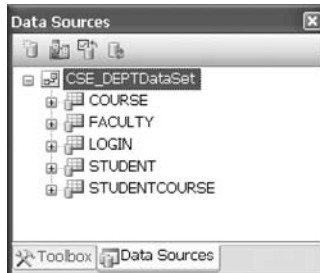


Figure 5.68 Five added data tables.

UserNamePassWord() for LogIn form, FillByFacultyName(), and FindFacultyIDByName() for the Faculty form and FillByFacultyID() for the Course form.

- You need to change all data table names and all data column names in the five tables (LogIn, Faculty, Course, Student, and StudentCourse) to uppercase in your program codes since the Oracle database engine changed all of those names to uppercase when this new database was generated. Otherwise you may encounter some debug errors when you build your project.
- You may also need to modify the BindingSource's name and the TableAdapter's name for each form (except the Selection form). For example, when you finished the data binding process for the Faculty form, the name of the BindingSource may have looked like **fACULTYBindingSource**. That looks a little ugly. Modify this name to **facultyBindingSource** by clicking on it and changing it in the Name property. Perform the same modification to the FACULTYTableAdapter.
- Copy all faculty and student image files and paste them into your desired folder in order to display each faculty and student picture. All image files are located at the folder Images that

is located at the accompanying ftp site (See Chapter 1). The general folder used to save those image files is the folder in which your executable C# project file is located, such as C:\Chapter 5\SelectWizardOracle\bin\Debug.

In addition to the points listed above, you also need to perform the following operations to make your project work properly:

For the LogIn form:

1. Bind two textbox controls, txtUserName and txtPassWord, to two columns, user_name and pass_word, in the LogIn table in the DataSet using the logInBindingSource.
2. Build the query methods FillByUserNamePassWord() and PassWordQuery() using the Query Builder.
3. Remove the LogInForm_Load() method and its content.

For the Faculty form:

1. Bind five label controls, TitleLabel, OfficeLabel, PhoneLabel, CollegeLabel, and EmailLabel, to the corresponding five columns in the Faculty table using the faculty BindingSource.
2. Build the query methods FillByFacultyName() and FindFacultyIDByName() using the Query Builder.

For the Course form:

1. Bind five textbox controls, txtName, txtSchedule, txtClassRoom, txtCredits, and txtEnroll, to the corresponding five columns in the Course table using the courseBindingSource.
2. Bind the listbox control CourseList to the Course table in the DataSet using the courseBindingSource.
3. Build the query method FillByFacultyID() using the Query Builder.

If you want to perform any query for the Student form, you need to build the associated query methods using the Query Builder and bind the target controls with the corresponding columns in the Student and StudentCourse tables.

A complete project that uses Oracle database, SelectWizardOracle, can be found in the folder Chapter 5 located at the accompanying ftp site (See Chapter 1). This project contains query methods, binding objects, and full code development for the StudentForm as well as the connections to the Student and StudentCourse data tables.

PART II DATA QUERY WITH RUNTIME OBJECTS

Unlike the sample data-driven application program we developed in Part I, in which quite few of the design tools and wizards provided by Visual Studio, such as the DataSet, BindingSource, BindingNavigator, and TableAdapter, are utilized to help us to develop professional data-driven applications easily and conveniently, the sample projects developed in this part have nothing to do with these tools and wizards. This means that we create these ADO.NET objects by directly writing Visual C# code without the aid of Visual Studio design-time tools and wizards as well as the autogenerated codes. All data-driven objects are created and implemented during the period the project runs. In other words, all these objects are created dynamically.

The shortcoming of using these Visual Studio design tools and wizards to create data connections is that the autogenerated connection codes related to tools and wizards are embedded in the programs, and those connection codes are machine dependent. Once that connection information in the programs is compiled, it cannot be modified. In other words, your programs cannot be distributed to and run in other platforms.

Compared with design tools and wizards, the advantages of using the runtime objects to make the data operations are: (1) They provide programmers more flexibility in creating and implementing connection objects and data operation objects related to ADO.NET and (2) they allow you to use different methods to access and manipulate data from the data source and the database. But everything has good and bad elements, and it is also true here. The flexibility also brings some complex stuff. For example, you have to create and use different data providers and different commands to access the different databases by using the different codes. Unlike the sample project we developed in Part I, in which you can use tools and wizards to select any data source you want and produce the same coding for the different data sources, in this part you must specify the data provider and command type based on your real data source to access the data in your project. The good news is that the LINQ technique released by Microsoft in Visual Studio 2008 provides an easier way to access different data sources from Visual C# projects. Before we can start to develop our projects, a detailed understanding of the connection and data operational classes is very important, and these classes are directly related to ADO.NET. Although already somewhat discussed in Chapter 3, here we will have a more detailed discussion of this topic in order to give readers a clearer picture of this issue.

5.17 INTRODUCTION TO RUNTIME OBJECTS

Runtime objects can be described as objects or instances used for data connections and operations in a data-driven application that are created and implemented during the period the project runs. In other words, these objects are created and utilized dynamically. To understand what kind of objects are most popularly used in a data-driven application, let's first have a detailed discussion about the most useful classes provided by ADO.NET.

According to Chapter 3, the ADO.NET architecture can be divided into three components: Data Provider, DataSet, and a DataTable. These three components are directly related to different associated classes, which are shown in Figure 5.69.

The Data Provider contains four components

1. **Data Connection**
2. **Data Command**
3. **DataReader**
4. **TableAdapter (DataAdapter)**

All components inside the Data Provider are Data Provider-dependent components, which means that all components including the Connection, Command, TableAdapter (DataAdapter), and DataReader are identified and named based on the real data provider or database the user used. For example, the Data Provider used for SQL Server database must be identified and named by a prefix such as the Sql, such as

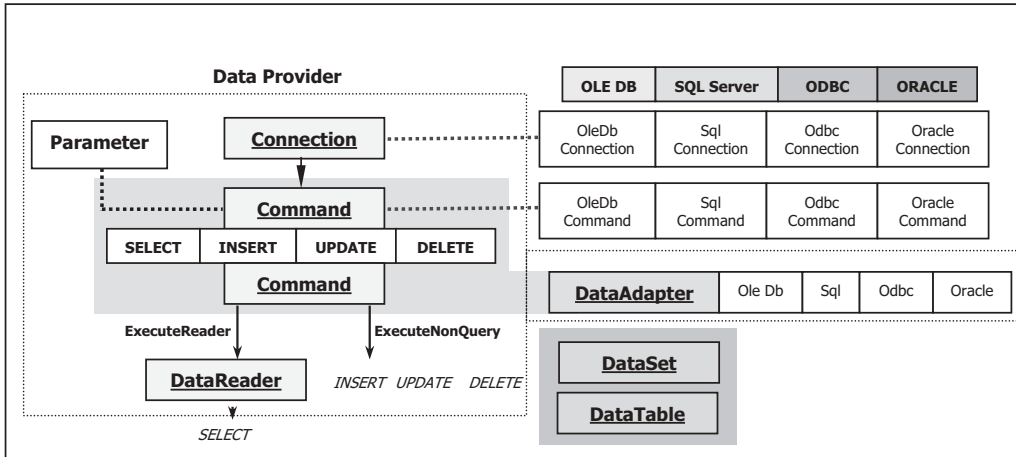


Figure 5.69 Classes provided by ADO.NET.

- Data Connection component SqlConnection
- Data Command component SqlCommand
- DataAdapter (TableAdapter) SqlDataAdapter (SqlTableAdapter)
- DataReader component SqlDataReader

and the same definition works for all three Data Providers. All classes, methods, properties, and constants of these four types of Data Provider are located at four different namespaces: `System.Data.OleDb`, `System.Data.SqlClient`, `System.Data.Odbc`, and `System.Data.OracleClient`.

As shown in Figure 5.69, four kinds of data providers are popularly used in database programming in Visual C# 2008. You must create the correct connection object based on your real database by using the specific prefix.

However, two components in the ADO.NET are Data Provider independent, that is, `DataSet` and `DataTable`. These two components are located at `System.Data` namespace. You do not need to use any prefix when you use these two components in your applications. Both `DataSet` and the `DataTable` can be filled by using the `DataAdapter` or the `TableAdapter` components.

ADO.NET provides different classes to allow users to develop a professional data-driven application by using the different methods. Among these methods, two popular methods will be discussed in this part in detail.

The first method is to use the so-called `DataSet–DataAdapter` method to build a data-driven application. `DataSet` and `DataTable` classes can have different roles when they are implemented in a real application. Multiple `DataTables` can be embedded into a `DataSet`, and each table can be filled, inserted, updated, and deleted by using the different query method provided by the `DataAdapter` such as the `SelectCommand`, `InsertCommand`, `UpdateCommand`, or `DeleteCommand` when one develops a data-driven application using this method. As shown in Figure 5.69, when you use this method, the `Command` and `Parameter` objects are embedded or attached to the `DataAdapter` object (represented by a shaded block), and the `DataTable` object is embedded into the

DataSet object (represented by another shaded block). This method is relatively simple since you do not need to call some specific objects such as the DataReader with a specific method such as the ExecuteReader or ExecuteNonQuery to complete this data query. You just need to call the associated command of the TableAdapter to finish this data operation. But this simplicity brings some limitations for your applications. For instance, you cannot access different data tables separately to perform multiple specific data operations by using this single specified DataAdapter.

The second method allows you to use each object individually, which means that you do not have to use the DataAdapter to access the Command object or embed the DataTable into the DataSet. This provides more flexibility. In this method, no DataAdapter or DataSet is needed, and you can only create a new Command object with a new Connection object, and then build a query statement and attach some useful parameter into that query for the new created Command object. You can fill any DataTable by calling the ExecuteReader() method to a DataReader object. You can also perform any data manipulation by calling the ExecuteNonQuery() method to the desired DataTable.

In addition to these two traditional data query methods, the LINQ method, which is a new method released by Microsoft in Visual Studio 2008, is also discussed later in this chapter.

In this section, we provide three sample projects to cover these two popular methods and the LINQ method: AccessSelectRTOject, SQLSelectRTOject, and OracleSelectRTOject, which are associated with three kinds of databases: Microsoft Access 2007, Microsoft SQL Server 2005, and Oracle Database 10g XE.

To have a better understanding about the LINQ method, refer to Chapter 4. To better understand these two popular methods, we need to have a clearer picture of how to develop a data-driven application using the related classes and methods provided by ADO.NET.

5.17.1 Procedure of Building a Data-Driven Application Using Runtime Objects

First, we will concentrate on two traditional query methods, and the LINQ method will be discussed later in this chapter. Recall that we discussed the architecture and important classes of the ADO.NET in Chapter 3. To connect and implement a database with your Visual C# project, you need to follow the operational sequences listed next:

1. Create a new Connection String with correct parameters.
2. Create a new Connection object by using the suitable Connection String built in step 1.
3. Call the Open() method to open the database connection using the Try...Catch block.
4. Create a new TableAdapter (DataAdapter) object.
5. Create a new DataTable object that can be filled with data.
6. Call the suitable command/object such as the SelectCommand [or the Fill()] or the DataReader to make data query.
7. Fill the data to the bound controls on the Visual C# form window.
8. Release the TableAdapter, Command, DataTable, and the DataReader used.
9. Close the database Connection object if no more database operation is needed.

Now let's first develop a sample project to access the data using the runtime objects for Microsoft Access 2007 database.

5.18 QUERY DATA USING RUNTIME OBJECTS TO MICROSOFT ACCESS 2007 DATABASE

The Microsoft Access 2007 database file used in this sample project is CSE_DEPT.accdb that we developed in Chapter 2, and it is located in the folder Database\Access at the accompanying ftp site (see Chapter 1).

First, we need to create a new Visual C# 2008 project named AccessSelectRuntimeObject with five form windows: LogIn, Selection, Faculty, Course, and Student. Refer to Sections 5.3.1.1 to 5.3.1.5 to build these form windows if you like. But in order to save time, a sample project named AccessSelectRuntimeObject with all completed forms is provided at the accompanying ftp site at the folder DBProjects\Chapter 5 (see Chapter 1). Open this project and let's begin to develop a data-driven application starting from the LogIn form.

5.18.1 Query Data Using Runtime Objects for LogIn Form

Open the LogIn form from the Solution Explorer window by clicking on the View Designer button, which is shown in Figure 5.70.

Two LogIn buttons are used in this form. The TabLogIn button is used to trigger the TabLogIn button click method to execute the first data query method, the DataSet-DataAdapter method. The ReadLogIn button is used to trigger the second method to run the data query, DataReader method, respectively. Change the original Login button's Name property to cmdTabLogIn and the Text property to TabLogIn. Add one more button and sets its Name property to cmdReadLogIn and the Text property to ReadLogIn. Now click on the View Code button to open its Code Window to begin our coding.

5.18.1.1 Declare Runtime Objects

As we mentioned in the last section, all components related to the OLE DB Data Provider supplied by ADO.NET are located at the namespace System.Data.OleDb. To access the Microsoft Access 2007 database file, you need to use this Data Provider.



Figure 5.70 LogIn form window.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace AccessSelectRTOject
{
    public partial class LogInForm : Form
    {
        public OleDbConnection accConnection = new OleDbConnection();
        public LogInForm()
        {
            InitializeComponent();
        }
    }
}

```

Figure 5.71 Declaration of the namespace for the OleDb Data Provider.

Therefore you must first declare this namespace inside the current project namespace using the **using** keyword to allow Visual C# 2008 to know that you want to use this specified Data Provider. Enter this declaration coding line just below the namespace System.Data, which appears in boldface and is shown in Figure 5.71.

This declaration provides a reference to the namespace that will be used in this project. As we discussed in the last section and in Chapter 3, both the DataSet and the DataTable components related to OleDb are also located at this namespace, so the Visual C# 2008 can use this reference to access these components.

Enter the following code into this project namespace to declare a new instance of the OleDbConnection classes:

```
public OleDbConnection accConnection = new OleDbConnection();
```

This global connection object accConnection is created based on the class OleDbConnection, and this connection will be used for the whole application. Although it is declared as a class-level object, but the accessing mode **Public** makes it be accessed by all other methods and objects defined in all forms in the project.

The first job we need to do is to connect our project with the database we selected after a new instance of the data connection object is declared.

5.18.1.2 Connect to Data Source with Runtime Objects

Because the connection job is the first thing you need to do before you can make any data query, you need to do the connection job in the constructor of the LogIn class to allow the connection to be made first as your project runs.

In the code window, enter the codes shown in Figure 5.72 into the constructor. Let's see how this piece of code works.

```

AccessSelectRTOject.LogInForm   LogInForm()
public LogInForm()
{
  InitializeComponent();
  A  string strConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;" +
                                "Data Source=C:\\database\\Access\\CSE_DEPT.accdb;";
  B  OleDbConnection conn = new OleDbConnection(strConnectionString);
  C  try
  {
    conn.Open();
  }
  D  catch (OleDbException e)
  {
    MessageBox.Show("Access Error");
    MessageBox.Show("Error Code = " + e.ErrorCode);
    MessageBox.Show("Error Message = " + e.Message);
  }
  E  catch (InvalidOperationException e)
  {
    MessageBox.Show("Invalid Message = " + e.Message);
  }
  F  if (conn.State != ConnectionState.Open)
  {
    MessageBox.Show("Database connection is Failed");
    Application.Exit();
  }
}

```

Figure 5.72 Coding for the constructor of the LogInForm class.



Starting from Microsoft Access 2007, the database driver is upgraded to Microsoft ACE.OLEDB 12.0. In the early versions of Microsoft Access, such as Access 2000, 2003, and XP, an old database driver Microsoft Jet.OLEDB 4.0 was used. The good news is that Visual Studio 2008 still allows users to use these old versions of Access with old database drivers.

Notice that a double backslash \\ is used in the connection string. The first backslash is used to indicate that the second backslash is to be treated literally, which means that the \\ is treated as \ in the connection string.

- A.** A connection string should be created first based on the procedure listed in Section 5.17.1. The connection string is used to indicate the detailed connection information including the name of the data provider, the location, and the name of the database, username, and password used to access the database. In this case, no username and password are utilized for our database; thus, those two items are missed from this connection string. You can add these two pieces of information if your database did utilize them. An important point to note is that the database driver for Microsoft Access 2007 is Microsoft ACE OLEDB 12.0, not Microsoft Jet OLEDB 4.0, which is used for the Access 2003 or earlier. Our sample database file is named CSE_DEPT.accdb and located at C:\database\Access directory.
- B.** By using the keyword **new**, we create a new instance of the connection class OleDbConnection with the connection string we built in step A.
- C.** A Try ... Catch block is utilized here to try to catch up any mistakes caused by opening a connection between our project and the Access database file, and furthermore connect-

ing our project to the database we selected. The advantage of using this kind of strategy is to avoid unnecessary system debug processes and simplify this debug procedure.

- D. A sequence of OleDbExceptionError messages will be displayed if an error related to the OleDb connection occurred.
- E. An InvalidOperationExceptionError message will also be displayed if an invalid operation error were encountered.
- F. This step is used to confirm that our database connection is successful. If not, an error message is displayed and the project is exited.

After a database connection is successfully made, we need to use this connection to access the database to perform our data query job.

5.18.1.3 Coding for Method 1: Using DataSet–DataAdapter to Query Data

In this section, we show how to create and use the runtime objects to query the data by using the DataSet–DataAdapter method.

Open the LogIn form window by clicking on the View Designer button, and then double-click on the TabLogIn button to open its Click method. Enter the codes shown in Figure 5.73 into this method.

Let's see how this piece of code works.

- A. Since the query string applied in this application is relatively long, we break it into two substrings. Then we combine these two substrings together using the += operator to form

AccessSelectRTOject.LogInForm	cmdTabLogIn_Click()
<pre> private void cmdTabLogIn_Click(object sender, EventArgs e) { A string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "; cmdString += "WHERE (user_name=@Param1) AND (pass_word=@Param2)"; B OleDbDataAdapter LogInDataAdapter = new OleDbDataAdapter(); DataTable accDataTable = new DataTable(); OleDbCommand accCommand = new OleDbCommand(); SelectionForm selForm = new SelectionForm(); C accCommand.Connection = accConnection; accCommand.CommandType = CommandType.Text; accCommand.CommandText = cmdString; D accCommand.Parameters.Add("@Param1", OleDbType.Char).Value = txtUserName.Text; accCommand.Parameters.Add("@Param2", OleDbType.Char, 8).Value = txtPassWord.Text; E LogInDataAdapter.SelectCommand = accCommand; F LogInDataAdapter.Fill(accDataTable); G if (accDataTable.Rows.Count > 0) { MessageBox.Show("LogIn is successful!"); selForm.Show(); this.Hide(); } H else { MessageBox.Show("No matched username/password found!"); } I accDataTable.Dispose(); accCommand.Dispose(); LogInDataAdapter.Dispose(); } </pre>	

Figure 5.73 Coding for the TabLogIn button.

a complete query string. A trick when you combine multiple strings into a final string is that you must leave a space either at the end of the first subquery string or at the beginning of the second subquery string since a space is required between the ... FROM LogIn and the WHERE clause. Otherwise, a running error will be encountered and this bug is not easy to find and correct. Two dynamic parameters, Param1 and Param2, are included in the WHERE clause, and these two parameters will be replaced by the real values entered by the user as the project runs.

- B.** Some data components used for this query are generated here, such as the DataAdapter, DataTable, Command, and the next form window.
- C.** The Command object accCommand is initialized by using three properties: Connection object, Command type, and Command string.
- D.** Note that there are two dynamic parameters, @Param1 and @Param2, in the query string, and these two parameters need to be replaced by two real parameters that will be entered by the user via two textboxes, txtUserName and txtPassWord, respectively, when the project runs. To add these two parameters, the Add() method in the Parameters collection is called. The Add() method can be overloaded and it has four different constructors. In this code fragment, two of them are used. The first constructor contains two arguments: the parameter's name and the parameter's type. The second one includes one more argument, the parameter's length in bytes. One can also assign the value to the parameter by using the Value property. In this application, two values come from the users' input: txtUserName.Text and txtPassWord.Text.



An SQL statement should be represented by a string in Visual C#. If an SQL statement is too long, it can be broken into multiple substrings, but the original format of the SQL statement cannot be modified, which means that a space is required between each clause in the SQL statement.

- E.** After two parameters are added into the Parameters collection that is the property of the Command object, the command object is ready to be used. It is then assigned to the method SelectCommand() of the DataAdapter.
- F.** The Fill() method of the DataAdapter is called to fill the LogIn table. Basically, when the Fill() is called, the SelectCommand() is executed to perform the query we built in the previous steps.
- G.** By checking the Count property, we can inspect whether this fill is successful or not. A successful message is displayed if this property's value is greater than 0, which means that some data have been added to the LogIn table. The next form window, Selection, will be displayed and the current form window, LogIn, will be closed. Later in our normal programming codes, this message will be commented out since it is only used for the testing purpose.
- H.** An error message will be issued if this property is 0, which means that no row or record has been added to the LogIn table.
- I.** A cleaning job is executed to release all objects we used for this data query. This cleaning includes the DataTable, DataAdapter, and Command objects. A Dispose() method is also used to finish this cleaning job.

Now let's take a look at the codes for the second method.

5.18.1.4 Coding for Method 2: Using DataReader to Query Data

Open the LogIn form window by clicking on the View Designer button from the Solution Explorer window, and then double-click on the ReadLogIn button to open its Click method. Enter the codes shown in Figure 5.74 into this method.

Most codes in the top section are identical with those codes in the TabLogIn button's method with two exceptions. First, a DataReader object is created to replace the DataAdapter to perform the data query, and, second, the DataTable is removed from this method since we do not need it for our data query. Let's see how this piece of code works.

- A. Instead of calling the Fill() method, here we call the ExecuteReader() method to run the Command object with two dynamic parameters to perform a query. The acquired data would be assigned to the DataReader object.
- B. By checking the HasRows property of the DataReader object, we can inspect whether the DataReader has received data or not. A successful message will be displayed if this property is True, which means that the DataReader has received the data from the LogIn table. The next form window, Selection, will be displayed and the current form window, LogIn, will be closed. Later in our normal programming codes, this successful message will be commented out since it is only used for the testing purpose at this moment.
- C. An error message will be displayed to require the user to handle this situation if the HasRows is False, which means that no data has been received by the DataReader and the login has failed.
- D. A cleaning job is performed to release all objects we used for this data query.

```

private void cmdReadLogIn_Click(object sender, EventArgs e)
{
    string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
    cmdString += "WHERE (user_name=@Param1 ) AND (pass_word=@Param2)";
    OleDbCommand accCommand = new OleDbCommand();
    SelectionForm selfForm = new SelectionForm();
    OleDbDataReader accDataReader;
    accCommand.Connection = accConnection;
    accCommand.CommandType = CommandType.Text;
    accCommand.CommandText = cmdString;
    accCommand.Parameters.Add("@Param1", OleDbType.Char).Value = txtUserName.Text;
    accCommand.Parameters.Add("@Param2", OleDbType.Char, 8).Value = txtPassWord.Text;
    accDataReader = accCommand.ExecuteReader();
    if (accDataReader.HasRows == true)
    {
        A
        B
        C
        D
        MessageBox.Show("LogIn is successful!");
        selfForm.Show();
        this.Hide();
    }
    else
        MessageBox.Show("No matched username/password found!");
    accCommand.Dispose();
    accDataReader.Close();
}

```

Figure 5.74 Coding for the ReadLogIn button Click method.

AccessSelectRTOject.LogInForm	▼	cmdCancel_Click()	▼
<pre>private void cmdCancel_Click(object sender, EventArgs e) { accConnection.Close(); accConnection.Dispose(); Application.Exit(); }</pre>			

Figure 5.75 Coding for the Cancel button Click method.

AccessSelectRTOject.LogInForm	▼	getLogInForm()	▼
<pre>public LogInForm getLogInForm() { return this; }</pre>			

Figure 5.76 Coding for the getLogInForm method.

Before we can finish the coding for this part, we need to do two more coding jobs; the first one is for our last button, Cancel. The purpose for this coding is to clean up the objects used in this form. Return to the LogIn form window by clicking the View Designer button from the Solution Explorer window, and then double-click on the Cancel button to open its Click method. Enter the codes shown in Figure 5.75 into this method.

To release the Connection object, a Close() method is called first. Then the Dispose() method is used to finish this release. Finally the system method Application.Exit() is called to close the whole project. Please note that the Connection instance would not be released if this Cancel button were not clicked. In the normal case, we still need to use this Connection object for the following data query if the login process is successful.

The second coding is to reserve a reference of the current form LogInForm. The purpose to keep this reference is that we can access this LogInForm object later from other objects or classes when we need to use it, for example, we need to access it from the SelectionForm object to close this form window as the Exit button is clicked by the user. Add the codes shown in Figure 5.76 to the end of the code window of the LogInForm.

It is the time for us to test our project. Click the Start button to begin our project. Enter ybai and reback as the username and the password into the two textboxes on the LogIn form window, as shown in Figure 5.77a; then click on the TabLogIn button. A successful message is displayed and it means that our data query is successful, which is shown in Figure 5.77b.

Click on OK to the MessageBox dialog, and then click on the ReadLogIn button to test our data query by using the DataReader method. Similarly, a successful message is displayed. You can try to enter other correct usernames and passwords to test the project, and even to enter some wrong usernames or passwords to see what will happen. Click on the Cancel button to terminate the project.

Our project is successful! But before we can move on, do not forget to comment out two successful MessageBoxes in the two Click methods we just developed.

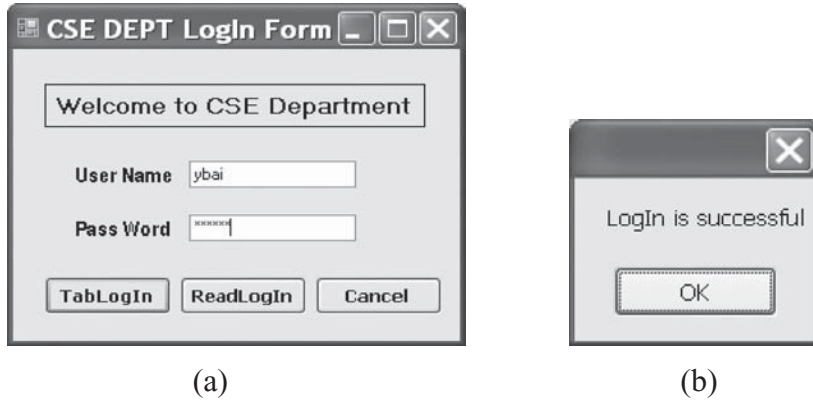


Figure 5.77 Running status of the project.

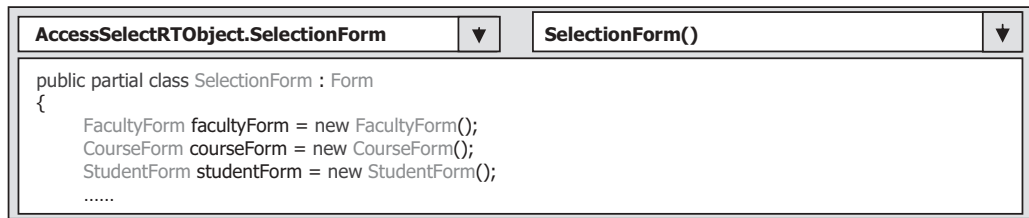


Figure 5.78 Coding of the fields variables.

5.18.2 Coding for Selection Form

The coding is very similar to those we did for the SelectionForm in the project SelectWizard. The coding can be divided into three parts:

1. Coding for the constructor of the SelectionForm class and the coding to create some fields variables
2. Coding for the OK button's Click method
3. Coding for the Exit button's Click method

Let's start with the first coding. Open the code window of the SelectionForm, and enter the codes shown in Figure 5.78 into the field section to create some fields variables.

Then enter the codes shown in Figure 5.79 into the constructor of the SelectionForm class as the initialization codes. The coding in this part is straightforward. First, we create some class-level variables, or fields variables, for the three form window classes. Second, we add three pieces of information related to the CSE_DEPT using the Add() method to the ComboBox in the SelectionForm window to allow users to select one of them to perform the data query.

The coding for the second step is for the OK button's Click method. Open this method by double-clicking on the OK button from the SelectionForm window, and enter the codes shown in Figure 5.80 into this method.

AccessSelectRTOject.SelectionForm	▼	SelectionForm()	▼
<pre> public SelectionForm() { InitializeComponent(); this.ComboSelection.Items.Add("Faculty Information"); this.ComboSelection.Items.Add("Course Information"); this.ComboSelection.Items.Add("Student Information"); this.ComboSelection.SelectedIndex = 0; } </pre>			

Figure 5.79 Coding for the constructor of the SelectionForm class.

AccessSelectRTOject.SelectionForm	▼	cmdOK_Click()	▼
<pre> private void cmdOK_Click(object sender, EventArgs e) { if (this.ComboSelection.Text == "Faculty Information") facultyForm.Show(); else if (this.ComboSelection.Text == "Course Information") courseForm.Show(); else if (this.ComboSelection.Text == "Student Information") studentForm.Show(); else MessageBox.Show("Invalid Selection!"); } </pre>			

Figure 5.80 Coding for the constructor of the SelectionForm class.

AccessSelectRTOject.SelectionForm	▼	cmdExit_Click()	▼
<pre> private void cmdExit_Click(object sender, EventArgs e) { A LogInForm logForm = new LogInForm(); logForm = logForm.getLogInForm(); B if (logForm.accConnection.State == ConnectionState.Open) logForm.accConnection.Close(); C logForm.Close(); courseForm.Close(); facultyForm.Close(); studentForm.Close(); D Application.Exit(); } </pre>			

Figure 5.81 Coding for the cmdExit button's Click method.

Finally, we come to the coding for the third step, coding for the Exit button's Click method. Open the Exit button's Click method and enter the codes into this method shown in Figure 5.81.

Let's see how this piece of code works.

- A. Since the Connection instance `accConnection` is created in the LogIn form window with the Public accessing mode, we need first to create a new instance of the LogInForm class. Then we can call the `getLogInForm()` method to obtain the original LogInForm instance and assign it to the new created LogInForm instance. In this way, we get the original LogInForm instance and can access any component located in that instance.

- B. Next we need to check whether a valid database is still connected to our project, in other words, a connection between our project and our database is still open or active. If it is, the Close() method is called to close this connection.
- C. The Close() method is executed to close all opened form windows.
- D. Finally, the project is exited by calling a system method Application.Exit().

Now we can move on to the next form, the Faculty form.

5.18.3 Query Data Using Runtime Objects for Faculty Form

Three data query methods will be used for the data query in this form: DataAdapter, DataReader, and LINQ method.

Because we need to use the OleDb data provider to perform the data query in this form, first we need to add one more namespace, System.Data.OleDb, into the namespace section on the code window of this form. Open the code window and add

```
using System.Data.OleDb;
```

to the namespace part of this form.

Next we need to create a class-level label array, FacultyLabel(7), to save seven columns in the Faculty data table. We need this array for the retrieving and assigning columns to the associated label controls on the FacultyForm window as the project runs.

Now we need to do the coding for the constructor of the FacultyForm as it performs some initialization tasks. Enter the codes into the constructor, and your finished coding should match that shown in Figure 5.82.

Let's see how this piece of code works.

```

AccessSelectRTOject.FacultyForm | FacultyForm()
public partial class FacultyForm : Form
{
A   private Label[] FacultyLabel = new Label[7];
    public FacultyForm()
    {
B       InitializeComponent();
        ComboName.Items.Add("Ying Bai");
        ComboName.Items.Add("Satish Bhalla");
        ComboName.Items.Add("Black Anderson");
        ComboName.Items.Add("Steve Johnson");
        ComboName.Items.Add("Jenney King");
        ComboName.Items.Add("Alice Brown");
        ComboName.Items.Add("Debby Angles");
        ComboName.Items.Add("Jeff Henry");
        ComboName.SelectedIndex = 0;
C       this.cmdSelect_Click(this.cmdSelect, null);
D       ComboMethod.Items.Add("DataAdapter Method");
        ComboMethod.Items.Add("DataReader Method");
        ComboMethod.Items.Add("LINQ & Object Method");
        this.ComboMethod.SelectedIndex = 0;
    }
}

```

Figure 5.82 Coding for the constructor of the FacultyForm.

- A.** In order to hold the retrieved columns from the Faculty table temporarily, an object array is created as a class-level label array. Since the array index is zero based and there are seven columns in our faculty data table, the array size is selected as seven. A little trick for the size of this array is that the first index of this array is 0, not 1. So we have seven label objects defined with an index starting from 0 to 6.
- B.** Eight faculty names are added into the Faculty Name combobox to allow the user to make selections as the project runs. Resetting the SelectedIndex property to 0 means that the first faculty name from the combobox has been chosen as the default one.
- C.** This line is used to call and execute the Click method of the Select button. When you run the project and open the Faculty form the first time, no faculty should have been selected, and therefore no any faculty information is retrieved and displayed in the five labels. In order to avoid this, we try to call the Click method to select the default faculty and retrieve the related information from the database and display it in the Faculty form. This call is equivalent to clicking on the Select button from the Faculty form. It sounds good, but you may encounter an error when you add this line into this constructor. Do not worry about this and this error will be corrected when we perform the coding for the Select button's Click method later.
- D.** Three query methods are added into the ComboMethod combobox to enable the user to choose one of them to perform the data query. The DataAdapter method is selected as the default one.

Now let's do the coding for the Select button's Click method. As the project runs, the user can choose one method from the ComboMethod combobox to perform the data query job. Then the user needs to click on the Select button to begin the data query.

Open the form window of the FacultyForm and double-click on the Select button to open the cmdSelect click method. Enter the codes shown in Figure 5.83 into this method.

Let's see how this piece of code works.

- A.** The query string is created first, and this string is composed of two substrings.
- B.** The necessary data objects to be used in this method are declared here. The objects `accDataTable` and `FacultyDataAdapter` are used for the first, and the third method—`DataAdapter` and `LINQ`—and `accDataReader` is used for the second method. The object `accCommand` will be used for all methods. The `DataSet` is used to query data using the `LINQ to DataSet` method since there is no direct way to connect `LINQ` to the Microsoft Access database. The new instance of the `LogInForm` is used to access and get the original `LogInForm` object we created in the `LogInForm` window since we declared the `Connection` object in that form and we need to use that connection in this form.
- C.** The `Command` instance is initialized with the `Connection` object that is created in the `LogInForm`, the `Command` type, and the query string. The `Add()` method is used to add the content of the Faculty Name combobox control, which is a dynamic parameter and entered by the user as the project runs, to the `Parameters` collection that is a property of the `Command` object. In this way, we completed the building of the `Command` object with our desired data query statement. One point one needs to note is that the first argument of this `Add()` method must be identical to the dynamic parameter we defined in the `WHERE` clause in the query string `cmdString`.
- D.** A user-defined method `ShowFaculty()` is executed to display the selected faculty photo in the `PhotoBox` control on the Faculty form window. The argument is the faculty name. An error message will be displayed if this method has failed.

```

AccessSelectRTOject.FacultyForm cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
A   string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty ";
B   cmdString += "WHERE faculty_name = @Param1";
   OleDbDataAdapter FacultyDataAdapter= new OleDbDataAdapter();
   OleDbCommand accCommand = new OleDbCommand();
   OleDbDataReader accDataReader;
   DataTable accDataTable = new DataTable();
   DataSet ds = new DataSet();
   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();
C   accCommand.Connection = logForm.accConnection;
   accCommand.CommandType = CommandType.Text;
   accCommand.CommandText = cmdString;
   accCommand.Parameters.Add("@Param1", OleDbType.Char).Value = ComboName.Text;
D   string strName = ShowFaculty(ComboName.Text);
   if (strName == "No Match")
       MessageBox.Show("No Matched Faculty Image found!");
E   if (ComboMethod.Text == "DataAdapter Method")
   {
       FacultyDataAdapter.SelectCommand = accCommand;
       FacultyDataAdapter.Fill(accDataTable);
F       if (accDataTable.Rows.Count > 0)
           FillFacultyTable(ref accDataTable);
G       else
           MessageBox.Show("No matched faculty found!");
H       accDataTable.Dispose();
       FacultyDataAdapter.Dispose();
   }
I   else if (ComboMethod.Text == "DataReader Method")
   {
       accDataReader = accCommand.ExecuteReader();
J       if (accDataReader.HasRows == true)
           FillFacultyReader(accDataReader);
K       else
           MessageBox.Show("No matched faculty found!");
       accDataReader.Close();
   }
L   else //LINQ to Object Method is selected
   {
       FacultyDataAdapter.SelectCommand = accCommand;
       FacultyDataAdapter.Fill(ds, "Faculty");
M       var facultyinfo = (from fi in ds.Tables["Faculty"].AsEnumerable()
                           where fi.Field<string>("faculty_name").Equals(ComboName.Text)
                           select fi);
N       foreach (var fRow in facultyinfo)
       {
           this.TitleLabel.Text = fRow.Field<string>("title");
           this.OfficeLabel.Text = fRow.Field<string>("office");
           this.PhoneLabel.Text = fRow.Field<string>("phone");
           this.CollegeLabel.Text = fRow.Field<string>("college");
           this.EmailLabel.Text = fRow.Field<string>("email");
       }
   }
}
}

```

Figure 5.83 Coding for the cmdSelect click method.

- E.** Next we need to check which data query method the user has selected. If the `DataAdapter` method is chosen, which means that the user wants to perform the data query with the `DataAdapter` to fill the `Faculty` table, then we need to assign the completed `Command` object to the `SelectCommand` property of the `DataAdapter`, and call the `Fill()` method to execute this `Command` to fill the `Faculty` table.
- F.** By checking the `Rows.Count` property of the `Faculty` table, we can determine whether the `Faculty` table is filled successfully or not. If this table filling is fine, a user-defined method `FillFacultyTable()` is executed with the filled `Faculty` table as the argument to fill the label controls on the `Faculty` form window. A `ref` is prefixed before this argument to indicate that this argument is passed by reference, and any modification to this `Faculty` table will be reflected to the original table.
- G.** Otherwise an error message is displayed to indicate that this fill has failed.
- H.** A cleaning job is performed to release the `FacultyDataAdapter` and the `DataTable` objects.
- I.** If the user selected the second method or the `DataReader` method to perform this data query, the `ExecuteReader()` method is called to run the completed `Command` object with our desired query statement. The returned data is assigned to the `DataReader` object.
- J.** By checking the `HasRows` property, we can determine whether this query is successful or not. If this property is `True`, which means that the `DataReader` did receive the data from this query, a user-defined method `FillFacultyReader()` is called with the `DataReader` as an argument to fill the label controls on the `Faculty` form window.
- K.** Otherwise an error message is displayed to show the user that no matched faculty has been found from this query, and the `DataReader` object is released.
- L.** If the user selected the `LINQ to Object` method, we must first build a `DataSet` and fill it with our `Faculty` data table. The reason for that is because there is no direct relationship between the `LINQ` and the Microsoft Access database, but we can set up an indirect relationship between them using the `LINQ to ADO.NET` since `ADO.NET` covers any kinds of database including the Microsoft Access 2007. In order to set up this relationship, a `DataSet` must be built by filling it with the desired data table such as `Faculty` in this case. One point to note is that this `DataSet` is a nontyped `DataSet` if it is built in this way, and you must use the field location, not field name, to identify each column in the data table.
- M.** A typical `LINQ` query structure is created and executed to retrieve all related information for the selected faculty member. The `facultyinfo` is a C# 2008 implicitly typed local variable with a data type `var`. The C# 2008 will be able to automatically convert this `var` to any suitable data type. In this case, it is a data table, when it sees it. An iteration variable `fi` is used to iterate over the result of this query from the `Faculty` table. Then a similar SQL `SELECT` statement is executed with the `WHERE` clause. The key point for this structure is the operator `AsEnumerable()`. Since different database systems use different collections and query operators, these collections must be converted to the type of `IEnumerable<T>` in order to use the `LINQ` technique because all data operations in `LINQ` use a `Standard Query Operator` method that can perform complex data queries on an `IEnumerable<T>` sequence. A compiling error would be encountered without this operator.
- N.** The `foreach` loop is utilized to pick up each column from the selected data row `fRow`, which is obtained from the `facultyinfo` we get from the `LINQ` query. Then, assign each column to the associated label in the `FacultyForm` window to display them. Since we are using a nontyped `DataSet`, we must indicate each column clearly with the `field<string>` and the column's name as the position for each of them.

```

AccessSelectRTOject.FacultyForm FillFacultyTable()
private void FillFacultyTable(ref DataTable FacultyTable)
{
A   int pos1 = 0;
B   for (int pos2 = 0; pos2 <= 6; pos2++) //Initialize the object array
    FacultyLabel[pos2] = new Label();
C   MapFacultyTable(FacultyLabel);
D   foreach (DataRow row in FacultyTable.Rows)
    {
        foreach (DataColumn column in FacultyTable.Columns)
        {
            FacultyLabel[pos1].Text = row[column].ToString();
            pos1++;
        }
    }
}

```

Figure 5.84 Coding for the method FillFacultyTable().

Now let's take a look at the coding for our two user-defined methods, FillFacultyTable() and FillFacultyReader(). The first method is FillFacultyTable() and its coding is shown in Figure 5.84.

Let's see how this piece of code works.

- A.** The integer variable pos1 is used as a loop counter later to retrieve the data from the DataTable and assign them to the associated bound label control on the Faculty form.
- B.** Next we need to initialize the class-level object array FacultyLabel by creating seven instances of the Label control. Recall that we have seven columns in our Faculty table, so this label array size is 7 (from 0 to 6).
- C.** Then another user-defined method MapFacultyTable() is called to set the correct mapping relationship between each label object in the label array and the data retrieved from the DataTable.
- D.** A double foreach loop is utilized to assign each data column read out from the DataTable to the mapped label control on the Faculty form window. Basically, we have only one row (one record) selected from the Faculty table based on the faculty name, so the outer loop is only executed by one time and the inner loop is executed by seven times. Because the distribution order of the label controls on the Faculty form and the column order in the query string (cmdString) is not identical, we need this MapFacultyTable() method to align them. A ToString() method is used to convert the returned data column whose type is Object to the string and assign it to the Text property of the associated label control in the FacultyForm.

The detailed coding for the method MapFacultyTable() is shown in Figure 5.85. This coding is straightforward and easy to understand. Let's take a closer look at this piece of code.

The order of labels on the right-hand side of the equals symbol is the order of the queried columns in the query string—cmdString (columns 0 and 1 are faculty_id and faculty_name), but the distribution order of five label controls on the Faculty form window is different. By performing this assignment, the five label controls on the Faculty form window have a one-to-one relation with the queried columns in the Faculty table.

AccessSelectRTOBJECT.FacultyForm	▼	MapFacultyTable()	▼
<pre> private void MapFacultyTable(Object[] fLabel) { fLabel[2] = OfficeLabel; //The order must be identical fLabel[3] = PhoneLabel; //with the real order in the query string fLabel[4] = CollegeLabel; fLabel[5] = TitleLabel; fLabel[6] = EmailLabel; } </pre>			

Figure 5.85 Coding for the MapFacultyTable subroutine.

AccessSelectRTOBJECT.FacultyForm	▼	FillFacultyReader()	▼
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p>	<pre> private void FillFacultyReader(OleDbDataReader FacultyReader) { int intIndex = 0; for (intIndex = 0; intIndex <= 6; intIndex++) //Initialize the object array FacultyLabel[intIndex] = new Label(); MapFacultyTable(FacultyLabel); while (FacultyReader.Read()) { for (intIndex = 0; intIndex <= FacultyReader.FieldCount - 1; intIndex++) FacultyLabel[intIndex].Text = FacultyReader.GetString(intIndex); } } </pre>		

Figure 5.86 Coding for the method FillFacultyReader().

Now let's continue to do the coding for another method FillFacultyReader(). This method is used to retrieve the queried data from the DataReader and distribute them to five label controls on the Faculty form window. The detailed codes for this method are shown in Figure 5.86.

Let's see how this piece of code works.

- A.** A loop counter intIndex is declared.
- B.** Seven instances of the label object array are created and initialized. These seven objects are mapped to 7 columns in the Faculty table in the database.
- C.** The user-defined method MapFacultyTable() is called to set up the correct mapping between the 5 label controls on the Faculty form window and the 5 columns in the Faculty table in the database.
- D.** A while loop is executed as long as the loop condition Read() method is True, which means that a valid data is read out from the DataReader. This method will return a False if no valid data can be read out from the DataReader, which means that all data has been read out. In this application, in fact, this while loop is only executed one time since we have only one row (one record) read out from the DataReader.
- E.** A for loop is utilized to pick up each data read out from the DataReader object, and assign each of them to the associated label control on the Faculty form window. Please note that the first two columns (faculty_id and faculty_name in the Faculty data table) are not used for this application. The GetString() method with the index is used here to identify each column from the DataReader.

The final method we need to make the coding is the ShowFaculty(). This method is used to identify the faculty image file based on the faculty name, and display the selected faculty image in the PhotoBox control on the Faculty form window. The coding for this method is very similar to one we made in Section 5.13.

A switch structure is utilized to select the correct faculty image file, and the system drawing method System.Drawing.Image.FromFile() is called to display the faculty image based on the selected faculty image file. One point you need to note is the location where the faculty images should be located. Generally, you can store those faculty image files in any folder on your computer or in any server that is connected to a network with your computer. The point is that you must provide the full name of the faculty image file, including the drive and path as well as the name of the faculty image file, to the system drawing method to perform this displaying. A convenient way to do this is to save the faculty image files in the folder in which your Visual C# project executable file is located. In this way, you do not need to give the full name of the faculty image files but only the name of the faculty image file itself. For example, in this application, our Visual C# project executable file is in the folder Chapter 5\AccessSelectRTOBJECT Solution\AccessSelectRTOBJECT Project\bin\Debug. So you should save all faculty image files to this folder.

The detailed coding for this method is shown in Figure 5.87. Let's see how this piece of code works.

- A. A local string variable strName is declared, and it is used to hold the name of the faculty image file.
- B. A switch structure is used to choose the correct faculty image file, and save it into the local string variable strName. Since these faculty image files are saved in the folder in which our Visual C# project executable file is located, only the name of the faculty image file is needed for the system drawing method to display the faculty image.
- C. If no matched faculty image file is found, a "No Match" string is assigned to the strName variable.
- D. The system drawing method is executed to draw the faculty image based on the name of the selected faculty image file if the content of the local variable strName is not equal to "No Match".
- E. Finally, the strName is returned to the calling program as a feedback of the execution of this method.

The last thing we need to do is to make coding for the Back button Click method. This coding is very easy. The current form window, the FacultyForm, should be made to disappear from the screen if this button is clicked. The method Hide() is suitable for this coding, which is shown in Figure 5.88.

At this point, we have finished the coding for the Faculty form. Let's test our project now. Click on the Start button to run our project. Enter ybai and reback as the username and password for the LogIn form window, and then click on either the TabLogIn or the ReadLogIn button to open the Selection form window. Make the default selection DataAdapter Method from the ComboMethod combobox, and click on the OK button to open the Faculty form window, which is shown in Figure 5.89.

Select a desired faculty member from the Faculty Name combobox, such as Jenney King, and then click on the Select button to make the information query for this selected

```

private string ShowFaculty(string fName)
{
  string strName;
  switch (fName)
  {
    case "Black Anderson":
      strName = "Anderson.jpg";
      break;
    case "Ying Bai":
      strName = "Bai.jpg";
      break;
    case "Satish Bhalla":
      strName = "Satish.jpg";
      break;
    case "Steve Johnson":
      strName = "Johnson.jpg";
      break;
    case "Jenney King":
      strName = "King.jpg";
      break;
    case "Alice Brown":
      strName = "Brown.jpg";
      break;
    case "Debby Angles":
      strName = "Angles.jpg";
      break;
    case "Jeff Henry":
      strName = "Henry.jpg";
      break;
    default:
      strName = "No Match";
      break;
  }
  if (strName != "No Match")
  {
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage;
    PhotoBox.Image = System.Drawing.Image.FromFile(strName);
  }
  return strName;
}

```

Figure 5.87 Coding for the ShowFaculty method.

```

private void cmdBack_Click(object sender, EventArgs e)
{
  this.Hide();
}

```

Figure 5.88 Coding for the cmdBack button Click method.

faculty. Five label controls bound to the associated columns in the Faculty table are updated with the queried information, and the selected faculty image is also displayed in the PhotoBox control, which is shown in Figure 5.89. You can try to select the different method by clicking on the drop-down arrow from the Method combobox. Yes, the project works fine with all three methods without any problem at all!



Figure 5.89 Running status of the Faculty form.

Our next job is to do the coding for the Course form.

5.18.4 Query Data Using Runtime Objects for Course Form

Three data query methods will be used for the data query on this form: `DataAdapter`, `DataReader`, and LINQ method. As we did for the `FacultyForm`, we also need to use the `OleDb` data provider to perform the data query in this `CourseForm`. Thus, first we need to add one more namespace, `System.Data.OleDb`, into the namespace section on the code window of this form. Open the code window and add

```
using System.Data.OleDb;
```

to the namespace part of this form.

Next we need to create a class-level textbox array, `CourseTextBox[6]`, and a class level `DataSet` object `ds`. The textbox array is used to temporarily save five columns in the Course data table, and we need this array when we retrieve and assign columns to the associated textbox controls on the `CourseForm` window as the project runs. The `ds` `DataSet` is used for the LINQ method since there is no direct relationship between the LINQ and Access database, and we need this `DataSet` to perform a LINQ to `DataSet` operation to do the data query (refer to Figure 5.90).

Now we need to do the coding for the constructor of the `CourseForm` to do some initialization tasks. Enter the codes into the constructor, and your finished coding should match that shown in Figure 5.90.

Let's see how this piece of code works.

- A.** The namespace `System.Data.OleDb` is added here since we need to use this `OleDb` Data Provider to perform the data query in this form.
- B.** This coding fragment is very similar to the one we did for the `Faculty` form. The only difference is that the `Label` array has been replaced by a `TextBox` array since we used 5 textbox controls to display the detailed course information that is related to the selected faculty from the `Faculty Name` combobox. The `Course` table has 7 columns, but we only

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
A using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace AccessSelectRTOject
{
    public partial class CourseForm : Form
    {
B         private TextBox[] CourseTextBox = new TextBox[6];
        DataSet ds = new DataSet();
        public CourseForm()
        {
C             InitializeComponent();
            ComboName.Items.Add("Ying Bai");
            ComboName.Items.Add("Satish Bhalla");
            ComboName.Items.Add("Black Anderson");
            ComboName.Items.Add("Steve Johnson");
            ComboName.Items.Add("Jenney King");
            ComboName.Items.Add("Alice Brown");
            ComboName.Items.Add("Debby Angles");
            ComboName.Items.Add("Jeff Henry");
            ComboName.SelectedIndex = 0;
D             ComboMethod.Items.Add("DataAdapter Method");
            ComboMethod.Items.Add("DataReader Method");
            ComboMethod.Items.Add("LINQ & DataSet Method");
            ComboMethod.SelectedIndex = 0;
        }
    }
}

```

Figure 5.90 Coding for the constructor of the CourseForm.

need six of them, so the size of this TextBox array is 6 and each element or each TextBox control in this array is indexed from 0 to 5. The ds DataSet is used for the LINQ to DataSet method.

- C.** All faculty names in the CSE_DEPT are added into the ComboName combobox control and the first faculty is selected as the default one.
- D.** Three query methods, DataAdapter, DataReader, and LINQ to DataSet, are added into the ComboMethod combobox control, and the first method is selected as the default method.

The next coding job is for the Select button. After the user selected the desired data query method from the Method combobox and the faculty member from the Faculty Name combobox, the Select button is used to trigger its Click method to retrieve all courses (basically all course_id) taught by the selected faculty.

Before we can go ahead and do this coding, you need to note that we need two queries to perform this data action in this method because there is no faculty_name column available in the Course table, and the only available column in the Course table is faculty_id. Therefore, we must first make a query to the Faculty table to find the faculty_id that is related to the faculty name selected by the user from the Faculty Name combobox in the Course form, and then we can make the second query to the Course table to pickup all course_id based on the faculty_id we obtained from the first query.

The queried `course_id` is displayed in the `CourseList` box, and the detailed course information for each course can be displayed in five textboxes when the user clicks on the associated `course_id` from the `CourseList` box.

Now return to the `CourseForm` window by clicking on the `View Designer` button, and double-click on the `Select` button to open its `Click` method and enter the codes shown in Figure 5.91 into this method.

The coding of this part is very similar to the one we did for the `Select` button `Click` method in the `Faculty` form. Let's see how this piece of code works.

- A. Two query strings are used for this data query. The first is used to find the `faculty_id` based on the `faculty` name from the `Faculty` table. The second is used to retrieve all `course_id` from the `Course` table. The `course` table has seven columns but we only need six of them. There are six query items related to six columns: `course_id`, `course`, `credit`, `classroom`, `schedule`, and the `enrollment`. The `course_id` column contains the `course_id` that will be displayed in the `CourseList` box, and the other five items will be displayed in five textboxes as the detailed information for selected `course_id`. The `faculty_id` is used as the criterion to query the desired course information for the selected `faculty`.
- B. The necessary instances and data components are also created at this part to aid with the data query task. Two sets of objects, which include `DataAdapters`, `Commands`, and `DataTables`, are declared, and one set is for the `Faculty` table and the other set is for the `Course` table. The `DataReader` object is used for the data querying using the `DataReader` method, and the `DataRow` object is used to reserve the returned row from the `Faculty` table. The string variable `strFacultyID` is used to hold the queried `faculty_id` from the `Faculty` table.
- C. Then the first `Command` object, `accCmdFaculty`, is initialized by assigning it with the `Connection` instance, `Command` type, and the query string. The dynamic parameter `Param1` is obtained from the `Faculty` Name combobox in which the `faculty` name will be selected by the user as the project runs.
- D. The completed `Command` object `accCmdFaculty` is assigned to the `SelectCommand` property of the `FacultyDataAdapter`, which is ready to make query by using the `Fill()` method.
- E. The `Fill()` method is executed to fill the `faculty` data table named `accFacultyTable`, and it is used to find the `faculty_id` that is matched to the selected `faculty` name from the `ComboName` in the `CourseForm` from the `Faculty` table.
- F. By checking the `Count` property, we can confirm whether this `Fill` is successful or not. If the value of this property is greater than 0, which means that at least one row has been selected and filled into the `Faculty` table, the returned first row or the only row, `accFacultyTable.Rows[0]`, is assigned to the `DataRow` object `rowFaculty`. Then the first column in that row, `rowFaculty[0]`, which is the matched `faculty_id`, is converted to string and saved to the `strFacultyID` variable that will be used later.
- G. An error message will be displayed if this `Fill()` has failed.
- H. Next the `course` `Command` object `accCmdCourse` is initialized and the dynamic parameter `@Param2` is replaced by the real parameter `faculty_id` obtained from the first query we did above.
- I. As we did for the `Faculty` form, the user can make a choice among three query methods: `DataAdapter`, `DataReader`, and `LINQ to DataSet`. If the user selects the `DataAdapter` method, the built command object in step **H** will be assigned to the `SelectCommand` property of the `CourseDataAdapter` and the `Fill()` method of the `DataAdapter` will be executed to fill the `Course` table, `accCourseTable`.

AccessSelectRTOject.CourseForm		cmdSelect_Click()	
		private void cmdSelect_Click(object sender, EventArgs e)	
		{	
A		string strFaculty = "SELECT faculty_id FROM Faculty WHERE faculty_name = @Param1";	
		string strCourse = "SELECT course_id, course, credit, classroom, schedule, enrollment, faculty_id FROM Course ";	
B		strCourse += "WHERE faculty_id = @Param2";	
		OleDbDataAdapter CourseDataAdapter = new OleDbDataAdapter();	
		OleDbDataAdapter FacultyDataAdapter = new OleDbDataAdapter();	
		OleDbCommand accCmdFaculty = new OleDbCommand();	
		OleDbCommand accCmdCourse = new OleDbCommand();	
		DataTable accCourseTable = new DataTable();	
		DataTable accFacultyTable = new DataTable();	
		LogInForm logForm = new LogInForm();	
		logForm = logForm.getLogInForm();	
		OleDbDataReader accDataReader;	
		string strFacultyID = string.Empty;	
		DataRow rowFaculty;	
C		accCmdFaculty.Connection = logForm.accConnection;	
		accCmdFaculty.CommandType = CommandType.Text;	
		accCmdFaculty.CommandText = strFaculty;	
D		accCmdFaculty.Parameters.Add("@Param1", OleDbType.Char).Value = ComboName.Text;	
E		FacultyDataAdapter.SelectCommand = accCmdFaculty;	
F		FacultyDataAdapter.Fill(accFacultyTable);	
		if (accFacultyTable.Rows.Count > 0)	
		{ rowFaculty = accFacultyTable.Rows[0]; strFacultyID = rowFaculty[0].ToString(); }	
G		Else { MessageBox.Show("No matched faculty_id found!"); }	
H		accCmdCourse.Connection = logForm.accConnection;	
		accCmdCourse.CommandType = CommandType.Text;	
		accCmdCourse.CommandText = strCourse;	
		accCmdCourse.Parameters.Add("@Param2", OleDbType.Char).Value = strFacultyID;	
I		if (ComboMethod.Text == "DataAdapter Method")	
		{	
		CourseDataAdapter.SelectCommand = accCmdCourse;	
		CourseDataAdapter.Fill(accCourseTable);	
J		if (accCourseTable.Rows.Count > 0) { FillCourseTable(accCourseTable); }	
K		else { MessageBox.Show("No matched course found!"); }	
L		accFacultyTable.Dispose(); accCourseTable.Dispose(); CourseDataAdapter.Dispose();	
		}	
M		else if (ComboMethod.Text == "DataReader Method")	
		{	
		accDataReader = accCmdCourse.ExecuteReader();	
N		if (accDataReader.HasRows == true) { FillCourseReader(accDataReader); }	
O		else { MessageBox.Show("No matched course found!"); }	
P		accDataReader.Close(); accDataReader.Dispose();	
		}	
Q		else //LINQ to DataSet Method is selected	
		{	
		CourseList.Items.Clear();	
R		CourseDataAdapter.SelectCommand = accCmdCourse;	
		CourseDataAdapter.Fill(ds, "Course");	
S		var courseinfo = (from ci in ds.Tables["Course"].AsEnumerable()	
		where ci.Field<string>("faculty_id") == (string)strFacultyID	
		select ci);	
T		foreach (var cRow in courseinfo)	
		CourseList.Items.Add(cRow.Field<string>("course_id"));	
U		ds.Clear();	
		}	
V		accCmdFaculty.Dispose();	
		accCmdCourse.Dispose();	
W		CourseList.SelectedIndex = 0;	
		}	

Figure 5.91 Coding for the cmdSelect button Click method.

- J.** If this fill is successful, the Count property of the Course table should be greater than 0, which means that the table is filled by at least one row. The user-defined method FillCourseTable() will be called with the filled table as the argument to fill the CourseList box control with the course_id on the Course form.
- K.** Otherwise, if this Count is equal to 0, which means that no row or record has been filled into the Course table. An error message will be displayed for this situation.
- L.** Some necessary cleaning jobs are performed to release all objects we used for this data query.
- M.** If the user selected the DataReader method, the ExecuteReader() method is called to perform a reading-only operation to the Course table.
- N.** If the HasRows property of the DataReader is True, which means that the DataReader did receive some data, the user-defined method FillCourseReader() is executed with the DataReader as the argument to fill the CourseList box control with the course_id on the Course form window.
- O.** An error message will be displayed if the HasRows property is false.
- P.** The DataReader object is released after it is used up.
- Q.** If the user selects the LINQ to DataSet method, first we need to clean up the CourseList control to make it ready to be filled with course_id.
- R.** Then we need to build a new DataSet object by executing the Fill() method to fill the Course table in that new DataSet object since we need this DataSet to perform a LINQ to DataSet query.
- S.** A typical LINQ query structure is created and executed to retrieve back all related course_id for the selected faculty_id. The courseinfo is a Visual C# 2008 implicitly typed local variable with a data type var. The Visual C# 2008 will be able to automatically convert this var to any suitable data type; in this case, it is a collection when it sees it. An iteration variable ci is used to iterate over the result of this query from the Course table. Then a similar SQL SELECT statement is executed with the WHERE clause. The first key point for this structure is the operator AsEnumerable(). Since different database systems use different collections and query operators, those collections must be converted to a type of IEnumerable<T> in order to use the LINQ technique because all data operations in LINQ use a Standard Query Operator method that can perform complex data queries on an IEnumerable<T> sequence. A compiling error would be encountered without this operator. The second key point is that you have to use the explicit cast (string) to convert the strFacultyID to the string object and then assign it to the field of faculty_id as the criterion for this query.
- T.** The foreach loop is utilized to pick up each column from the selected data row cRow, which is obtained from the courseinfo we get from the LINQ query. Then, add each column to the CourseList in the CourseForm window to display them. Since we are using a nontyped DataSet, we must indicate each column clearly with the field<string> and the column's name as the position for each of them.
- U.** This DataSet's cleaning is very important, and a lot of sequences of duplicated course_id would be added into the CourseList without this cleaning. The reason is that the new course_id columns will be attached to the DataSet each time when the Fill() method is executed if the DataSet were not cleaned.
- V.** Two Command objects are released before we can exit this method.
- W.** This coding is very important, and it is used to select the first course_id as the default one from the CourseList box, and this coding can be used to trigger the CourseList_


```

AccessSelectRObject.CourseForm
FillCourseTable()

private void FillCourseTable(DataTable CourseTable)
{
    CourseList.Items.Clear();
    foreach (DataRow row in CourseTable.Rows)
    {
        CourseList.Items.Add(row[0]);           //the 1st column is course_id - strCourse
    }
}
private void FillCourseReader(OleDbDataReader CourseReader)
{
    string strCourse = string.Empty;
    CourseList.Items.Clear();
    while (CourseReader.Read())
    {
        strCourse = CourseReader.GetString(0);   //the 1st column is course_id
        CourseList.Items.Add(strCourse);
    }
}

```

Figure 5.92 Coding for two user-defined methods.

SelectedIndexChanged() method to display detailed information for the selected course_id in five textboxes. Without this default course_id selected, no detailed course information can be displayed as the Course List_SelectedIndexChanged() method is executed at the first time.

Now let's take a look at the codes of two user-defined methods, FillCourseTable() and FillCourseReader(). These two methods are used to fill the CourseList box control on the Course form by using the queried data obtained either from the DataAdapter or from the DataReader. The detailed codes for these two methods are shown in Figure 5.92.

Let's see how this piece of code works.

- A.** Before we can fill the CourseList box, a cleaning job is needed. This cleaning is very important and necessary, otherwise multiple duplicated course_id would be displayed in this listbox if you forget to clean it up first.
- B.** A foreach loop is used to scan all rows of the filled Course table. Recall that we queried six columns from the Course table in the database and filled them to this Course table in the DataSet starting with the first column that is course_id (refer to query string strCourse defined in the cmdSelect button Click method; see Figure 5.91). Now we need to pick up the first column—course_id (column index = 0)—for each returned row from the Course table. Then add each course_id into the CourseList control to display them by using the Add() method.
- C.** For the FillCourseReader() method, a local string variable strCourse is created, and this variable can be considered as an intermediate variable that is used to temporarily hold the queried column from the Course table.
- D.** We also need to clean up the CourseList box before it can be filled.
- E.** A while loop is utilized to retrieve each first column's data [GetString(0)] whose column index is 0 and the data value is the course_id. The queried data is first assigned to the intermediate variable strCourse, and then it is added into the CourseList box by using the Add() method.

Next we need to take care of the coding for the `CourseList_SelectedIndexChanged()` method. The functionality of this method is to display the detailed course information related to the selected `course_id` from the `CourseList` box, which includes the course name, classroom, schedule, credit, and enrollment in 5 textbox controls on the `Course` form. This method can be triggered as the user clicked on a `course_id` from the `CourseList` box.

Open the `Course` form window by clicking on the `View Designer` button from the `Solution Explorer` window, and then double-click on the `CourseList` box to open its `CourseList_SelectedIndexChanged()` method. Enter the codes shown in Figure 5.93 into this method. The code segment in this part is very similar to the one we did for the `cmdSelect` button `Click` method.

Let's see how this piece of code works.

- A. The query string is defined with 6 data columns that contain the detailed course information. Note that the first column is the course name with a column index of 0, and the criterion for the `WHERE` clause is `course_id`. This is because we want to retrieve all course information related to the selected `course_id` and display those pieces of information in the 5 textbox controls.
- B. The data components and objects used in this method are declared and created here, which include `CourseDataAdapter`, `Command`, `DataTable`, `DataReader`, and an instance of the `LogInForm` class. The purpose of creating this new instance is to get the `Connection` object from the `LogInForm` object since we created our database connection object in that class.
- C. The `Command` object is initialized with the `Connection` object, `CommandType`, and `CommandText` properties.
- D. The dynamic parameter `@Param1` is replaced by the real parameter, `CourseList.SelectedItem`, which will be selected by the user from the `CourseList` box as the project runs.
- E. If the user selects the `DataAdapter` method, the built command object is assigned to the `SelectCommand` property of the `CourseDataAdapter`, and the `Fill()` method is executed with the `Course` table as the argument to fill the `Course` table.
- F. If this fill is successful, which can be detected by checking the `Count` property of the `DataTable`, the queried data has been stored in the `Course` table. Next the `FillCourseTextBox()` method is executed with the `DataTable` as the argument to fill five textbox controls in the `Course` form window.
- G. Otherwise an error message will be displayed if this fill has failed.
- H. A cleaning job is performed to release objects used in this part, which include the `DataTable` and the `CourseDataAdapter`.
- I. If the user selects the `DataReader` method, the `ExecuteReader()` method is executed to perform a read-only operation to retrieve the information related to the selected `course_id` from the `CourseList` box.
- J. If this read-only operation is successful, the `HasRows` property of the `DataReader` will be `True`, the method `FillCourseReaderTextBox()` is called to fill five textbox controls on the `Course` form window.
- K. An error message will be displayed if this read-only operation has failed.
- L. A cleaning job is performed to release the `DataReader` object used in this data query.

```

AccessSelectRTObject.CourseForm CourseList_SelectedIndexChanged()
private void CourseList_SelectedIndexChanged(object sender, EventArgs e)
{
A   string cmdString = "SELECT course, credit, classroom, schedule, enrollment, course_id FROM Course ";
B   cmdString += "WHERE course_id = @Param1";
   OleDbDataAdapter CourseDataAdapter = new OleDbDataAdapter();
   OleDbCommand accCommand = new OleDbCommand();
   DataTable accDataTable = new DataTable();
   OleDbDataReader accDataReader;
   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();
C   accCommand.Connection = logForm.accConnection;
   accCommand.CommandType = CommandType.Text;
   accCommand.CommandText = cmdString;
D   accCommand.Parameters.Add("@Param1", OleDbType.Char).Value = CourseList.SelectedItem;
E   if (ComboMethod.Text == "DataAdapter Method")
   {
       CourseDataAdapter.SelectCommand = accCommand;
       CourseDataAdapter.Fill(accDataTable);
F       if (accDataTable.Rows.Count > 0)
           FillCourseTextBox(accDataTable);
G       else
           MessageBox.Show("No matched course information found!");
H       accDataTable.Dispose();
       CourseDataAdapter.Dispose();
   }
I   else if (ComboMethod.Text == "DataReader Method")
   {
       accDataReader = accCommand.ExecuteReader();
J       if (accDataReader.HasRows == true)
           FillCourseReaderTextBox(accDataReader);
K       else
           MessageBox.Show("No matched course information found!");
L       accDataReader.Close();
       accDataReader.Dispose();
   }
   else //LINQ & DataSet Method is selected
   {
M       DataSet dc = new DataSet();
       CourseDataAdapter.SelectCommand = accCommand;
       CourseDataAdapter.Fill(dc, "Course");
N       var cinfo = (from c in dc.Tables["Course"].AsEnumerable()
                    where c.Field<string>("course_id") == (string)CourseList.SelectedItem
                    select c);
O       foreach (var crow in cinfo)
       {
           txtName.Text = crow.Field<string>("course");
           txtSchedule.Text = crow.Field<string>("schedule");
           txtClassRoom.Text = crow.Field<string>("classroom");
           txtCredits.Text = crow.Field<int>("credit").ToString();
           txtEnroll.Text = crow.Field<int>("enrollment").ToString();
       }
P       dc.Clear();
   }
Q   accCommand.Dispose();
}

```

Figure 5.93 Coding for the CourseList SelectedIndexChanged method.

- M.** If the user selects the LINQ to DataSet method, first we need to create a new DataSet object `dc` and build it by executing the `Fill()` method. The `SelectCommand` property of the `CourseDataAdapter` should have been initialized with the `accCommand` object we built in step **C**.
- N.** A typical LINQ query structure is created and executed to retrieve back the detailed course information related to `course_id`. The `cinfo` is a Visual C# 2008 implicitly typed local variable with a data type `var`. The Visual C# 2008 will be able to automatically convert this `var` to any suitable data type, in this case, it is a collection, when it sees it. An iteration variable `c` is used to iterate over the result of this query from the `Course` table. Then a similar SQL `SELECT` statement is executed with the `WHERE` clause. The first key point for this structure is the operator `AsEnumerable()`. Since different database systems use different collections and query operators, those collections must be converted to a type of `IEnumerable<T>` in order to use the LINQ technique because all data operations in LINQ use a Standard Query Operator method that can perform complex data queries on an `IEnumerable<T>` sequence. A compiling error would be encountered without this operator. The second key point is that you have to use the explicit cast (string) to convert the `CourseList.SelectedItem` to the string object and then assign it to the field of `course_id` as the criterion for this query.
- O.** The `foreach` loop is utilized to pick up each column from the selected data row `crow`, which is obtained from the `cinfo` we get from the LINQ query. Then, assign each column to the associated textbox control in the `CourseForm` window to display them. Since we are using a nontyped `DataSet`, we must indicate each column clearly with the `field<string>` and the column's name as the position for each of them.
- P.** After this data query is done, we need to clean up the `DataSet` object. This cleaning job is very important and necessary. Otherwise the multiple duplicated `course_id` will be attached to the `DataSet` and displayed in the `CourseList` box each time when you click the `Select` button to query the `course_id` using this method if you forget this cleaning code.
- Q.** Finally the `Command` object is disposed of.

The codes for three user-defined methods, `FillCourseTextBox()`, `MapCourseTable()`, and `FillCourseReaderTextBox()`, are shown in Figure 5.94.

Let's see how this piece of code works.

- A.** A local integer variable `pos1` is created, and this variable will work as the loop counter for the `foreach` loop to retrieve each column from the `Course` table and assign each of them to the associated textbox control in the `Course` form to display them.
- B.** The class-level object array and textbox array are initialized here. Here six textbox objects are initialized, and they can be mapped to five textbox controls in the `Course` form window (note that the `course_id` does not have a matched textbox control in the `CourseForm` since we do not need it). We use these five textbox objects to display the detailed course information for the selected `course_id` from the `CourseList` box later.
- C.** The `MapCourseTable()` method is executed to set up a one-to-one mapping relationship between each textbox control on the `Course` form window and each queried column in the query. This step is necessary since the distribution order of five textbox controls on the `CourseForm` is different with the column order in the query string `cmdString` defined in the `CourseList_SelectedIndexChabnged()` method.
- D.** A double `foreach` loop is utilized to retrieve all columns and all rows from the `Course DataTable`; that is, the outer loop is only executed one time since we only query one record (one row) based on the selected `course_id` from the `Course` data table. The inner loop is

```

AccessSelectRTObject.CourseForm
FillCourseTextBox()

private void FillCourseTextBox(DataTable CourseTable)
{
    A   int pos1 = 0;
    B   for (int pos2 = 0; pos2 <= 5; pos2++)           //Initialize the object array
        CourseTextBox[pos2] = new TextBox();
    C   MapCourseTable(CourseTextBox);
    D   foreach (DataRow row in CourseTable.Rows)
        {
            foreach (DataColumn column in CourseTable.Columns)
            {
                CourseTextBox[pos1].Text = row[column].ToString();
                pos1++;
            }
        }
}

private void MapCourseTable(Object[] fCourse)
{
    E   fCourse[0] = txtName;           //The order must be identical with
        fCourse[1] = txtCredits;       //the real order in the query string -
        fCourse[2] = txtClassRoom;     //cmdString
        fCourse[3] = txtSchedule;
        fCourse[4] = txtEnroll;
}

private void FillCourseReaderTextBox(OleDbDataReader CourseReader)
{
    F   int intIndex = 0;
    G   for (intIndex = 0; intIndex <= 5; intIndex++) //Initialize the object array
        CourseTextBox[intIndex] = new TextBox();
    H   MapCourseTable(CourseTextBox);
    I   while (CourseReader.Read())
        {
            for (intIndex = 0; intIndex <= CourseReader.FieldCount - 1; intIndex++)
                CourseTextBox[intIndex].Text = CourseReader.GetValue(intIndex).ToString();
        }
}

```

Figure 5.94 Coding for three methods.

executed six times to pick up six pieces of course-related information that contains the course name, classroom, credit, schedule, enrollment, and course_id. Then the retrieved information is assigned to each textbox control in the textbox array, which will be displayed in that textbox control in the CourseForm later. The course_id is an exception since we do not need to display it in this application.

- E.** For the method MapCourseTable(), it assigns each textbox control to the matched partner in the textbox array to set up a correct relationship between them.
- F.** In the FillCourseReaderTextBox() method, a loop counter intIndex is first created, and it is used for the loop of initialization of the textbox object array and the loop of retrieving data from the DataReader later.
- G.** This loop is used to initialize the textbox object array.
- H.** The MapCourseTable() method is executed to set up a correct relationship between the textbox controls and the textbox array.
- I.** A while and a for loop are used to pick up all five piece of course-related information from the DataReader one by one. The Read() method is used as the while loop condition. A

```

AccessSelectRTOBJECT.CourseForm  cmdBack_Click()
private void cmdBack_Click(object sender, EventArgs e)
{
    this.Hide();
}

```

Figure 5.95 Coding for the cmdBack button Click method.

Figure 5.96 Running status of the Course form window.

returned True means that a valid data is read out from the DataReader, and a returned False means that no valid data has been read out from the DataReader; in other words, no more data is available and all data has been read out. The for loop uses the FieldCount-1 as the termination condition since the index of the first data field is 0, not 1, in the DataReader object. Each read-out data is converted to a string and assigned to the associated textbox control in the textbox object array.

The last coding is for the cmdBack button Click method. This coding is very simple and it is shown in Figure 5.95. The CourseForm window will disappear from the screen when this button is clicked on by the user.

Now let's test our project by clicking on the Start button. Enter the username and password as we did before, and select the Course Information from the Selection form window to open the Course form window, which is shown in Figure 5.96.

Select any method you want by clicking on the drop-down arrow from the Method combobox, and then select your desired faculty from the Faculty Name combobox. Click on the Select button, and all courses represented by the related course_id taught by that faculty will be displayed in the CourseList box, which is shown in Figure 5.96. Then select and click a desired course_id from the CourseList box. The detailed course information related to that selected course_id will be displayed in five textbox controls. (Isn't it so funny!)

5.18.5 Query Data Using Runtime Objects for Student Form

Basically, the coding for this Student form is similar to the coding we did for the Course form in the last section. The functionality of this Student form is to allow users to review the detailed information for each student in the CSE DEPT, which includes the student ID, major, GPA, school year, total credits the student earned, and courses the student took. The courses taken by the student are displayed in a CourseList box, and all other information is displayed in five textboxes as one clicks on the Select button.

The coding for this form is a little special since two data tables are utilized for this form: Student and StudentCourse. The first table contains the student's general information and the second one contains all courses taken by the student. Therefore two DataAdapters are needed for this application. Also two different data queries are needed to query data from two tables. The first one is used to retrieve the student's general information from the Student table, and the second is to pick up all course information (course_id) for the courses taken by the student from the StudentCourse table.

In order to save space, only two query methods—DataAdapter and LINQ to DataSet methods—are provided in this section. For the DataReader query method, we like to leave it as homework to the students.

The coding job is divided into two parts with two major methods: the constructor of the Student class and the cmdSelect_Click method. The first one is used to initialize the Student form and display all students' names on the combobox control, which can be selected by the user to review the related information for the selected student. The second one is to execute the data queries to pick up the selected student's general and course information and display them in the associated textbox controls and the ListBox control.

5.18.5.1 Coding for Constructor of Student Form

As we did before for the LogIn, Faculty, and Course forms, add the OleDb namespace, System.Data.OleDb, into this code window since we need to use it in this data query.

The coding for this constructor is shown in Figure 5.97. Let's take a look at this piece of code to see how it works.

- A. A new DataSet instance and a TextBox array StudentTextBox[] is created here. The DataSet instance is used for the LINQ to DataSet method, and the textbox object array is used to set up a mapping relationship between the 6 textboxes in the Student form and 6 query columns in the query string strStudent, student_id, gpa, credits, major, schoolYear, and email. The reason we defined the size of this array as 7 is that we need the seventh column, student_name, when we query data using the LINQ to DataSet method later.
- B. Five students' names are added into the ComboName combobox. As the project runs, the user can select any student by clicking the associated name to review the detailed information for the selected student in 6 textboxes and all courses taken by that student in the CourseList box.
- C. Two querying methods, DataAdapter and LINQ to DataSet, are added into the ComboMethod combobox to allow users to select either one to perform the data query as the project runs. The first item in both comboboxes is selected as the default item by setting the SelectedIndex property to 0.

```

public partial class StudentForm : Form
{
    DataSet ds = new DataSet();
    private TextBox[] StudentTextBox = new TextBox[7];    //We query 7 columns from the Student table
    public StudentForm()
    {
        InitializeComponent();
        ComboName.Items.Add("Erica Johnson");
        ComboName.Items.Add("Ashly Jade");
        ComboName.Items.Add("Holes Smith");
        ComboName.Items.Add("Andrew Woods");
        ComboName.Items.Add("Blue Valley");
        ComboName.SelectedIndex = 0;
        ComboMethod.Items.Add("DataAdapter Method");
        ComboMethod.Items.Add("LINQ & DataSet Method");
        ComboMethod.SelectedIndex = 0;
    }
    .....
}

```

Figure 5.97 Coding for the constructor of the Student Form.

5.18.5.2 Coding for Student Select Button Click Method

As the project runs, the user can select a student's name from the student name combobox and click on the Select button to acquire the detailed information for the selected student. The detailed information for the selected student is retrieved from the Student table in the CSE_DEPT database and displayed in six textboxes. Also all courses that are represented by the course_id and taken by the selected student are retrieved from the StudentCourse table and displayed in the CourseList listbox. So this method needs to perform two queries from two different tables.

The coding for this method is shown in Figure 5.98. Let's take a close look at this piece of code.

- A.** The query string for the Student table is declared first and the string contains seven columns in the Student data table, which are student_id, gpa, credits, major, schoolYear, email, and student_name. The criterion for this query is the student name stored in the student combobox. It looks like a contradiction exists in this string: Why do we need to query student_name that is the last column with a known student_name as the criterion? The answer is that the LINQ to DataSet method needs this column to call the Standard Query Operators to perform the data query, and we will show this situation later. The second query string is for the StudentCourse table. The queried columns are course_id and student_id, and the query criterion is the student_id. Similarly, the repeated query for the column student_id is for the requirement of the LINQ to DataSet method.
- B.** All data objects are created here, which include the Student and the StudentCourse DataAdapters, Commands, and DataTables. The string variable strName is used to hold the returned name of the student's image file from the calling of the function FindName().
- C.** The FindName() function is called to get and display the appropriate student's image based on the student name. If no matched image can be found, an error message is displayed.
- D.** The method BuildCommand() is called to build the Student Command object with the Connection object and the student query string as the arguments. You will find that the data type of the first argument, accCmdStudent, is a reference type (ref), which is equiva-

AccessSelectRTOject.StudentForm		cmdSelect_Click()
	private void cmdSelect_Click(object sender, EventArgs e)	
	{	
A	string strStudent = "SELECT student_id, gpa, credits, major, schoolYear, email, student_name FROM Student ";	
	strStudent += "WHERE student_name = @Param1";	
B	string strStudentCourse = "SELECT course_id, student_id FROM StudentCourse WHERE student_id = @Param2";	
	OleDbDataAdapter StudentDataAdapter = new OleDbDataAdapter();	
	OleDbDataAdapter StudentCourseDataAdapter = new OleDbDataAdapter();	
	OleDbCommand accCmdStudent = new OleDbCommand();	
	OleDbCommand accCmdStudentCourse = new OleDbCommand();	
	DataTable accStudentTable = new DataTable();	
	DataTable accStudentCourseTable = new DataTable();	
	string strName = string.Empty;	
C	strName = FindName(ComboName.Text);	
	if (strName == "No Match")	
	MessageBox.Show("No Matched Student's Image Found!");	
D	BuildCommand(ref accCmdStudent, strStudent);	
E	accCmdStudent.Parameters.Add("@Param1", OleDbType.Char).Value = ComboName.Text;	
	StudentDataAdapter.SelectCommand = accCmdStudent;	
F	if (ComboMethod.Text == "DataAdapter Method")	
	{	
	StudentDataAdapter.Fill(accStudentTable);	
G	if (accStudentTable.Rows.Count > 0)	
	FillStudentTextBox(accStudentTable);	
H	else	
	MessageBox.Show("No matched student found!");	
I	BuildCommand(ref accCmdStudentCourse, strStudentCourse);	
	accCmdStudentCourse.Parameters.Add("@Param2", OleDbType.Char).Value = txtID.Text;	
	StudentCourseDataAdapter.SelectCommand = accCmdStudentCourse;	
J	StudentCourseDataAdapter.Fill(accStudentCourseTable);	
K	if (accStudentCourseTable.Rows.Count > 0)	
	FillCourseList(accStudentCourseTable);	
L	else	
	MessageBox.Show("No matched course_id found!");	
	}	
	else //LINQ to DataSet Method	
	{	
M	StudentDataAdapter.Fill(ds, "Student");	
	LINQStudent(ds);	
N	BuildCommand(ref accCmdStudentCourse, strStudentCourse);	
	accCmdStudentCourse.Parameters.Add("@Param2", OleDbType.Char).Value = txtID.Text;	
	StudentCourseDataAdapter.SelectCommand = accCmdStudentCourse;	
	StudentCourseDataAdapter.Fill(ds, "StudentCourse");	
O	LINQStudentCourse(ds);	
P	ds.Clear();	
	}	
Q	accStudentTable.Dispose();	
	accStudentCourseTable.Dispose();	
	StudentDataAdapter.Dispose();	
	StudentCourseDataAdapter.Dispose();	
	accCmdStudent.Dispose();	
	accCmdStudentCourse.Dispose();	
	}	

Figure 5.98 Coding for the Student Select button Click method.

lent to a memory address or a constant pointer variable in C++. The reason is that when this method is done, the built command object will still be stored in that memory address and we can use it directly. This is very similar to using a returned object from the calling of this method.

- E. The dynamic parameter @Param1 in the first query string is replaced by the real student name obtained from the student name combobox, and the completed Command

object `accCmdStudent` is assigned to the `SelectCommand` property of the `StudentDataAdapter`.

- F.** If the `DataAdapter` method is selected, the `Fill()` method is called to fill the `Student` table.
- G.** By checking the `Count` property, we can inspect whether this fill is successful or not. If this property is greater than 0, which means that at least one row has been filled into the `Student` data table and the fill is successful, the user-defined method `FillStudentTextBox()` is called with the filled `Student` table as the argument to fill six textboxes in the `Student` form with the detailed student's information such as `student_id`, `gpa`, `credits`, `major`, `schoolYear`, and `email`.
- H.** Otherwise, an error message is displayed.
- I.** To make the second query to the `StudentCourse` table to find all courses taken by the selected student, the `BuildCommand()` is called again to initialize and build the `StudentCourse` Command object. The dynamic parameter `@Param2` is replaced by the real `student_id`, which was obtained from the first query and stored in the textbox `txtID`. The completed `StudentCourse` Command object, `accCmdStudentCourse`, is assigned to the `SelectCommand` property of the `StudentCourseDataAdapter`.
- J.** The `Fill()` method is called to fill the `StudentCourse` data table.
- K.** If the `Count` property of the `DataRow` object in the `StudentCourse` table is greater than 0, which means that the fill is successful, another user-defined method `FillCourseList()` is executed to fill all courses (that is, `course_id`) stored in the filled `StudentCourse` table into the `CourseList` box in the `Student` form.
- L.** Otherwise if the `Count` property is 0, which means that this fill has failed, an error message is displayed.
- M.** If the `LINQ to DataSet` method is selected, a new instance of the `DataSet` `ds` is created by executing the `Fill()` method since we need this `DataSet` to perform the data query using this method. Then a user-defined method, `LINQStudent()`, is called to perform this data query and retrieve back all queried information for the selected student, and display it in six textboxes in the `Student` form.
- N.** Next we need to perform the second query to the `StudentCourse` table to retrieve back all courses taken by the selected student. First we need to build the Command object, and then we need to create another instance of the `DataSet` class using the `Fill()` method.
- O.** Another user-defined method `LINQStudentCourse()` is called to retrieve back all courses (that is, `course_id`) from the `StudentCourse` table and add them into the `CourseList` listbox in the `Student` form.
- P.** The new created instance of the `DataSet` is cleaned up. This step is very important and necessary for this data query. Otherwise a lot of duplicated courses will be added into the `DataSet` object and displayed in the `CourseList` listbox each time you click on the `Select` button if this cleaning is not done. Those duplications can be effectively avoided by cleaning the `DataSet` object each time when this method is executed.
- Q.** Finally another cleaning job is performed to release all objects used in this method.

Now let's continue to finish the coding for all user-defined methods, and these methods are:

- `FindName()`
- `BuildCommand()`

```

AccessSelectRTOObject.StudentForm FindName()
private string FindName(string sName)
{
    string strName;
    switch (sName)
    {
        case "Erica Johnson":
            strName = "Erica.jpg";
            break;
        case "Ashly Jade":
            strName = "Ashly.jpg";
            break;
        case "Holes Smith":
            strName = "Holes.jpg";
            break;
        case "Andrew Woods":
            strName = "Andrew.jpg";
            break;
        case "Blue Valley":
            strName = "Blue.jpg";
            break;
        default:
            strName = "No Match";
            break;
    }
    if (strName != "No Match")
    {
        PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage;
        PhotoBox.Image = System.Drawing.Image.FromFile(strName);
    }
    return strName;
}

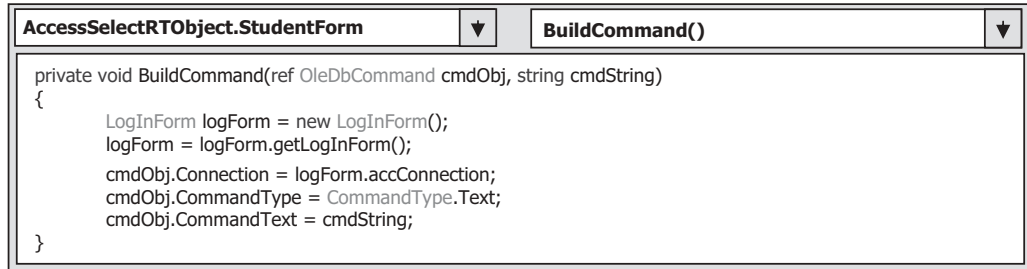
```

Figure 5.99 Coding for the FindName method.

- FillStudentTextBox()
- MapStudentTextBox()
- FillCourseList()
- LINQStudent()
- LINQStudentCourse()

First let's handle the coding for the FindName() method. This method is similar to the one we developed in the Faculty form, and the coding for this method is shown in Figure 5.99.

A switch case structure is used to select the desired student's image file based on the input student's name, and the selected student's image is displayed in a PictureBox in the Student form using the FromFile() system method. Note the location in which the student image files are located. You can save those image files in any folder on your computer or a server, but you must provide the full name for the selected image and assign it to the strName variable to be returned. The so-called full name includes the machine name, driver name, and folder name as well as the image file name. An easy way to save these image files is to save them in the folder in which your Visual C# project executable file is located. For instance, in this application our Visual C# project executable file is located in the folder C:\Chapter 5\MSAccessSelectRTOObject\bin\Debug. When save all student's

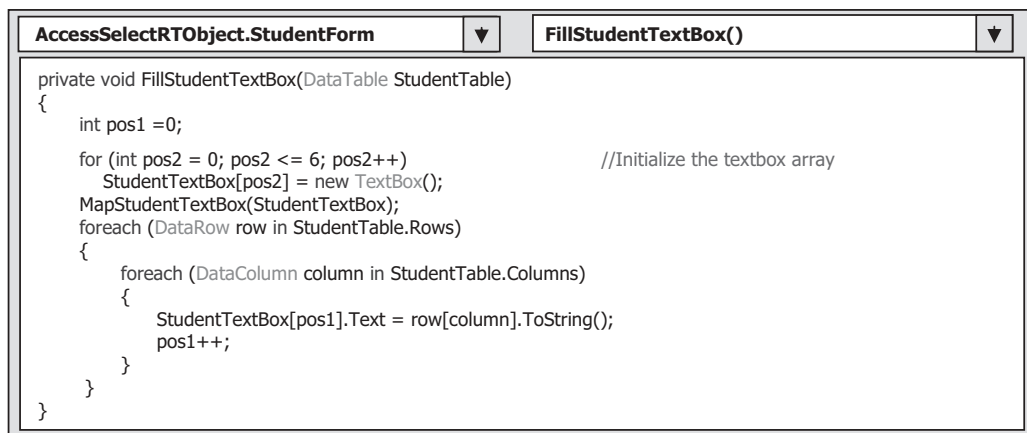


```

AccessSelectRTOject.StudentForm BuildCommand()
private void BuildCommand(ref OleDbCommand cmdObj, string cmdString)
{
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    cmdObj.Connection = logForm.accConnection;
    cmdObj.CommandType = CommandType.Text;
    cmdObj.CommandText = cmdString;
}

```

Figure 5.100 The BuildCommand method.



```

AccessSelectRTOject.StudentForm FillStudentTextBox()
private void FillStudentTextBox(DataTable StudentTable)
{
    int pos1 = 0;
    for (int pos2 = 0; pos2 <= 6; pos2++) //Initialize the textbox array
        StudentTextBox[pos2] = new TextBox();
    MapStudentTextBox(StudentTextBox);
    foreach (DataRow row in StudentTable.Rows)
    {
        foreach (DataColumn column in StudentTable.Columns)
        {
            StudentTextBox[pos1].Text = row[column].ToString();
            pos1++;
        }
    }
}

```

Figure 5.101 Coding for the FillStudentTextBox method.

image files in this folder, you don't need to provide the so-called full name for those images, and you only need to provide the image file's name and assign it to the variable strName. That is much easier! The codes for the BuildCommand() method are shown in Figure 5.100.

The coding is straightforward with no tricks. The different properties of the Command class such as the Connection string, Command type, and Command text are assigned to the Command object. Note that the data type of the first argument—cmdObj—is a reference (ref), as we mentioned above in step **D**. A reference in Visual C# 2008 is equivalent to a memory address or a pointer in C++, and the argument cmdObj is called passing by reference. When an argument is passing in this mode, the object cmdObj will work as both an input and an output argument, and they will be stored at the same address when this method is completed. We can use this built cmdObj as a returned object even if it is an argument without needing to return this cmdObj object from this method.

For some other user-defined methods used in this form, such as FillStudentTextBox(), FillCourseList(), and MapStudentTextBox(), the coding for them is similar to that we developed in the Course form. For your convenience, we list them here again with some simple explanations. The coding for the FillStudentTextBox() method is shown in Figure 5.101.

AccessSelectRObject.StudentForm	▼	MapStudentTextBox()	▼
<pre>private void MapStudentTextBox(Object[] sTextBox) { sTextBox[0] = txtID; //The order must be identical with the sTextBox[1] = txtGPA; //order in the query string - strStudent sTextBox[2] = txtCredits; sTextBox[3] = txtMajor; sTextBox[4] = txtSchoolYear; sTextBox[5] = txtEmail; }</pre>			

Figure 5.102 Coding for the MapStudentTextBox method.

AccessSelectRObject.StudentForm	▼	FillCourseList()	▼
<pre>private void FillCourseList(DataTable StudentCourseTable) { CourseList.Items.Clear(); foreach (DataRow row in StudentCourseTable.Rows) { CourseList.Items.Add(row[0]); //the 1st column is course_id - strStudentCourse } }</pre>			

Figure 5.103 Coding for the FillCourseList method.

The function of this piece of code is to fill six textboxes in the Student form with six column's data obtained from the Student table, such as student_id, gpa, credits, major, schoolYear, and email, which is the first query we discussed above. The StudentTextBox array is initialized first, and then the method MapStudentTextBox() is called to set up a correct relationship between the StudentTextBox array and six textboxes in the Student form. A nested foreach loop is executed to pick up each column from the queried rows. Exactly only one row data that matches the selected student name is obtained from the Student table; therefore, the outer loop is only executed once. The reason for using a double loop is that both the DataRow and the DataColumn are classes, and in order to pick up data from any DataTable, one must use the row and column objects as the index to access each row or column of DataTable instead of using an integer. The local integer variable pos1 works as an index for the StudentTextBox array.

The coding for the MapStudentTextBox() method is shown in Figure 5.102. The purpose of this coding is to set up a correct relationship between each textbox control in the StudentTextBox array and each column data in our first query string—strStudent. Each textbox control in the StudentTextBox array is related to an associated textbox control in the Student form such as student_id, gpa, credits, major, schoolYear, and email. Since the distribution order of those textboxes in the StudentTextBox array may be different with the order of those column data in our first query, a correct order relationship can be set up by executing this method.

The coding for the FillCourseList() method is shown in Figure 5.103. The function of this method is to fill the CourseList box with all courses taken by the selected student, and those queried courses are stored in the StudentCourse table, which are obtained by executing the second query to the StudentCourse table based on the student_id. In order

```

AccessSelectRTOject.StudentForm LINQStudent()
private void LINQStudent(DataSet dSet)
{
A   var studentinfo = (from si in dSet.Tables["Student"].AsEnumerable()
                        where si.Field<string>("student_name") == (string)ComboName.Text
                        select si);
B   foreach (var sRow in studentinfo)
    {
        this.txtID.Text = sRow.Field<string>("student_id");
        this.txtSchoolYear.Text = sRow.Field<string>("schoolYear");
        this.txtGPA.Text = sRow.Field<string>("gpa");
        this.txtCredits.Text = sRow.Field<int>("credits").ToString();
        this.txtMajor.Text = sRow.Field<string>("major");
        this.txtEmail.Text = sRow.Field<string>("email");
    }
}

```

Figure 5.104 Coding for the LINQStudent method.

to pick up each `course_id` from the `StudentCourse` table, a `DataRow` object is created first, and it can be used to hold each row or record queried from the `StudentCourse` table. After the `CourseList` box is cleared, a `foreach` loop is executed to pick up each row from the `StudentCourse` table, and the first column, which is `row(0)`, is `course_id` added into the `CourseList` box by executing the `Add` method.

Now let's handle the coding for two methods related to the LINQ to DataSet method. First let's take care of the method `LINQStudent()`. The coding for this method is shown in Figure 5.104.

Let's take a closer look at this piece of code to see how it works.

- A.** A typical LINQ query structure is created and executed to retrieve back the detailed student information related to the `student_name`. The `studentinfo` is a Visual C# 2008 implicitly typed local variable with a data type `var`. The Visual C# 2008 will automatically convert this `var` to any suitable data type; in this case, it is a collection. An iteration variable `si` is used to iterate over the result of this query from the `Student` table. Then a similar SQL `SELECT` statement is executed with the `WHERE` clause. The first key point for this structure is the operator `AsEnumerable()`. Since different database systems use different collections and query operators, those collections must be converted to a type of `IEnumerable<T>` in order to use the LINQ technique because all data operations in LINQ use a Standard Query Operator method that can perform complex data queries on an `IEnumerable<T>` sequence. A compiling error would be encountered without this operator. The second key point is that you have to use the explicit cast `(string)` to convert the `ComboName.Text` to the string object and then assign it to the field of `student_name` as the criterion for this query.
- B.** The `foreach` loop is utilized to pick up each column from the selected data row `sRow`, which is obtained from the `studentinfo` we get from the LINQ query. Then, assign each column to the associated textbox control in the `StudentForm` window to display them. Since we are using a nontyped `DataSet`, we must indicate each column clearly with the `field<string>` and the column's name as the position for each of them.

The coding for the `LINQStudentCourse()` method is shown in Figure 5.105. Let's see how this piece of code works.

AccessSelectRTOject.StudentForm	LINQStudentCourse()
A B C	<pre>private void LINQStudentCourse(DataSet dt) { CourseList.Items.Clear(); var scinfo = (from sc in dt.Tables["StudentCourse"].AsEnumerable() where sc.Field<string>("student_id") == (string)txtID.Text select sc); foreach (var sRow in scinfo) { CourseList.Items.Add(sRow.Field<string>("course_id")); } }</pre>

Figure 5.105 Coding for the LINQStudentCourse method.

- A.** As we did before, first we need to clean up the CourseList box by calling the Clear() method to make it ready to be filled with new courses (course_id). This step is necessary and important. Without this step, multiple duplicate course_ids will be added and displayed in this CourseList listbox control as the users click on the Select button and run this method to perform the student's information query.
- B.** A typical LINQ query structure is created and executed to retrieve back the course information related to the student_id. The scinfo is a Visual C# 2008 implicitly typed local variable with a data type var, and the Visual C# 2008 can automatically convert this var to any suitable data type; in this case, it is a collection. An iteration variable sc is used to iterate over the result of this query from the StudentCourse table. Then a similar SQL SELECT statement is executed with the WHERE clause.
- C.** The foreach loop is utilized to pick up each column from the selected data row sRow, which is obtained from the scinfo we get from the LINQ query. Then, add each column to the CourseList listbox control in the StudentForm window to display them. Since we are using a nontyped DataSet, we must indicate each column clearly with the field<string> and the column's name as the position for each of them.

The last coding is for the button Back. Open the cmdBack_Click method and enter the code: this.Hide() into this method.

Now it is time for us to run and test our project for this Student form. One thing you need to do before you can run this project is to make sure that all students' image files have been stored in the same folder as our Visual C# project executable file is located. Click on the Start Debugging button to run the project. Enter suitable username and password such as jhenry and test for the LogIn form, and click on the Students Information item from the Selection form to open the Student form window, which is shown in Figure 5.106.

Select a student name such as Ashly Jade from the Student Name combobox, and then click on the Select button. All courses taken by this student are shown in the CourseList box, and the detailed information about this student is displayed in the six textboxes.

A completed Visual C# 2008 project, MSAccessSelectRTOject, which includes the five form windows and related codes, can be found in the folder DBProjects\Chapter 5 located at the accompanying ftp site (see Chapter 1).

Next we will show readers how to develop a professional data-driven project using the runtime object for the SQL Server database.

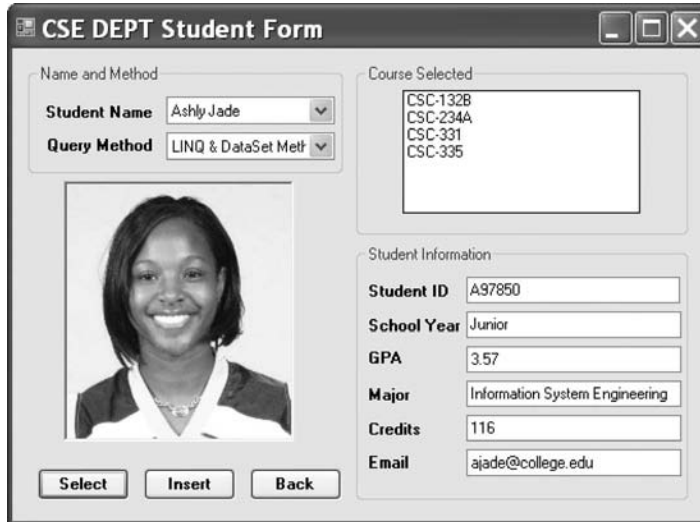


Figure 5.106 Running status of the Student form.

5.19 QUERY DATA USING RUNTIME OBJECTS TO SQL SERVER DATABASE

In the previous section you learned how to build a data-driven application using the runtime objects for the Microsoft Access 2007 database. Microsoft Access is a very good candidate when a small group of users with small amounts of data are concerned. However, when you need to work with a large group of users and large amounts of data, you need to use an enterprise relational database such as SQL Server or Oracle.

As we discussed in Chapter 3, one needs to use the different data providers to access the different databases, and the ADO.NET provides different namespaces for three different data providers: `System.Data.OleDb` for OLEDB, `System.Data.SqlClient` for SQL Server, and `System.Data.OracleClient` for Oracle database.

We divide this discussion into two parts: (1) query data using the runtime objects with general data query methods and (2) data query using runtime objects with the LINQ to SQL method. LINQ to SQL is a new technique available with Visual Studio 2008, and it is basically an API interface for working with SQL Server databases. We will first provide a detailed discussion about the general query methods for the SQL Server databases, and then we will concentrate on the LINQ to SQL with another project. We divide this part into two separate projects because of the contradiction of sharing the same SQL Server 2005 Express database between the project using general runtime objects and the project using the LINQ to SQL. As you know, SQL Server 2005 Express database allows only single database instance to be created and applied for a data-driven application, which means that it does not allow two or more users to access and use the same SQL Server 2005 Express database simultaneously. The first project we will develop is `SQLSelectRTOObject`, which uses the general runtime objects and SQL command object to connect to our sample SQL Server database file `CSE_DEPT.mdf` to perform the data query. The second project we will develop is `SQLSelectRTOObjectLINQ`, which uses the

LINQ to SQL technique and DataContext to connect to the same sample database file CSE_DEPT.mdf to perform the similar data operations. As we discussed in Chapter 4, the DataContext is a special class that provides a connection to the database and translates Standard Query Operators to the standard SQL statements to access our database. In order to avoid access and use the same database simultaneously, we have to separate this discussion into two parts with two different projects.

Before we can start this section, we need a clear picture about how to migrate from the Microsoft Access database to the SQL Server and Oracle databases.

5.19.1 Migrating from Access to SQL Server and Oracle Databases

Basically, the similar runtime objects and structures are utilized to develop a data-driven project that can access the different databases. For example, all three kinds of data providers need to use the Connection, Command, DataAdapter, and DataReader objects to perform data queries to either a DataSet or a DataTable. The DataSet and the DataTable components are data provider independent, but the first four objects are data provider dependent. This means that one must use the different prefix to specify what kind of data provider is utilized for certain databases. A prefix Sql would be used if an SQL Server data provider is utilized, such as SqlConnection, SqlCommand, SqlDataAdapter, and SqlDataReader. The same rule applies to the Oracle data provider.

So the differences between the data-driven applications that can access the different databases are the data provider-dependent components. Among them, the Connection String is a big issue. Different data providers need to use different connection strings to make the correct connection to the associated database.

Regularly, a Connection String is composed of five parts:

- Provider
- Data Source
- Database
- User ID
- Password

A typical data connection instance with a general connection string can be expressed by the following codes:

```
Connection = new xxxConnection("Provider=MyProvider;" +
                                "Data Source=MyServer;" +
                                "Database=MyDatabase;" +
                                "User ID=MyUserID;" +
                                "Password=MyPassWord;");
```

where **xxx** should be replaced by the selected data provider in a real application, such as OleDb, Sql, or Oracle. You need to use the real parameter values implemented in your applications to replace those nominal values such as MyServer, MyDatabase, MyUserID, and MyPassWord in a real application.

The Provider parameter indicates the database driver you selected. If you installed a local SQL server and client such as the SQL Server 2005 Express on your computer,

the provider should be *localhost*. If you are using a remote SQL Server instance, you need to use that *remote server's network name*. If you are using the default named instance of SQLX on your computer, you need to use *.SQLEXPRESS* as the value for your provider parameter. Similar values can be used for the Oracle server database.

The Data Source parameter indicates the name of the network computer on which your SQL server or Oracle server is installed and running. The Database parameter indicates your database name. The User ID and Password parameters are used for security issues for your database. In most cases, the default Windows NT Security Authentication is utilized.

You can also use the OLEDB as the SQL Server database provider. A sample connection string to be connected to an SQL Server database using the OLEDB data provider can be expressed as follows:

```
Connection = new OleDbConnection("Provider=SQLOLEDB;" +
    "Data Source=MyServer;" +
    "Database=CSE_DEPT;" +
    "User ID=MyUserID;" +
    "Password=MyPassWord;");
```

You need to use the real parameter values implemented in your applications to replace those nominal values such as MyServer, MyUserID, and MyPassWord for this connection object.

When you want to connect the SQL Server database using SqlClient, the connection string is a little different from those strings shown above. The Provider parameter should be replaced by the Server parameter, and the User ID and the Password parameters should be replaced by the Integrated Security parameter. A sample connection string to be used to connect to an SQL Server database using the SqlClient is:

```
Connection = new SqlConnection("Server=losthost;" +
    "Data Source=Susan\\SQLEXPRESS;" +
    "Database=CSE_DEPT;" +
    "Integrated Security=SSPI");
```

where the value for the Data Source parameter is *Computer Name\SQL Server 2005 Express name* since we installed the Express version of the SQL 2005 Server in our local computer. Also we installed the SQL 2005 Client on the same computer to make it work as both a server and a client.

When you build a connection string to be used by an Oracle database using the OLEDB provider, you can use the same parameters as those shown in the typical connection string with three exceptions: The Provider, Database, and Data Source parameters. First, to connect to an Oracle database, an MSDAORA driver should be used for the Provider parameter. Second, the Database parameter is not needed when connecting to an Oracle database because the tnsnames.ora file contains this piece of information, and this tnsnames.ora file is created as you installed and configured the Oracle client on your computer. Third, the Data Source will not be used to indicate the computer name on which the Oracle is installed and running. This information is included in the tnsnames.ora file, too.

A sample connection string to be connected to an Oracle database using the OLEDB provider can be expressed as follows:

```

Connection = new OleDbConnection("Provider=MSDAORA;" +
                                   "Data Source=MySID;" +
                                   "User ID=MyUserID;" +
                                   "Password=MyPassWord;");

```

You need to use the real parameter values implemented in your applications to replace those nominal values such as MySID, MyUserID, and MyPassWord for this connection object.

To build a connection string to be used by the Oracle database using the OracleClient, you should understand that most parameters are included in the tnsnames.ora file, and an Oracle connection string is inseparable from Oracle names resolution. Suppose we had a database alias of OraDb defined in a tnsnames.ora file as follows:

```

OraDb =
  (DESCRIPTION=
    (ADDRESS_LIST=
      (ADDRESS=(PROTOCOL=TCP)(HOST=OTNSRVR)(PORT=1521))
    )
    (CONNECT_DATA=
      (SERVER=DEDICATED)
      (SERVICE_NAME=ORCL)
    )
  )

```

To use the OraDb alias defined in the tnsnames.ora file shown above, you can create a very simple connection string. A sample connection string that is built using the OracleClient and will be used by Oracle database:

```

OraDb = new OracleConnection("Data Source=OraDb;" +
                               "User ID=MyUserID;" +
                               "Password=MyPassWord;");

```

We have discussed the development of the data-driven application using the OLEDB data provider in the last section. In the following sections, we will show users how to develop the professional data-driven applications connecting to the SQL Server or Oracle databases using the different data providers. First, we discuss the data query for the SQL Server database and then the Oracle database.

In this section, we use an SQL Server 2005 Express database and connect it with our example project using the SQL Server data provider. The SQL Server database file used in this sample project is CSE_DEPT.mdf, which was developed in Chapter 2. You can get it from the folder Database\SQLServer located at the accompanying ftp site (see Chapter 1). The advantages of using the Express version of SQL Server 2005 SP2 include, but are not limited to:

- The SQL Server 2005 Express is fully compatible with SQL Server 2005 database and has full functionalities of the latter.
- The SQL Server 2005 Express can be easily download from the Microsoft site free of charge.
- The SQL Server Management Studio Express 2005 can also be downloaded and installed on your local computer free of charge. You can use this tool to build your database easily and conveniently.

- The SQL Client can be downloaded and installed on your local computer free of charge. You can install both SQL Server and Client on your local computer to develop professional data-driven applications to connect to your SQL Server database easily.

5.19.2 Query Data Using General Runtime Objects

Now we need to create a Visual C# 2008 project named SQLSelectRuntimeObject with five form windows: LogIn, Selection, Faculty, Course, and Student. Because this project and the project developed in the last section are very similar, AccessSelectRuntimeObject, you do not need to redevelop all coding for this one. You do need to create a new project named SQLSelectRuntimeObject and add all four forms (except the LogIn) from the last project to this one by using Project/Add Existing Item menu. The reason we did not include the LogIn form for this form-adding operation is that we want to modify the default Form1 in this new project to make it our LogIn form. Otherwise it would be tough and complicated if you delete the default Form1 and add the LogIn form into this new project since the default Form1 works as a start form and some default codes related to it have been created by the system. You need to do some modifications for those codes if you did that deletion. The important differences between these two projects that are connected to the different databases are:

- The namespace for each form object
- The Data Provider–dependent objects, including the connection string

To save time, in this section we only emphasize the different codes that exist between this project and those for in the last one. Now let's create a new Windows-based Visual C# 2008 project and name it SQLSelectRuntimeObject.

Open the default Form1 window from the Solution Explorer window and perform the following operations to this form:

1. Change the name of the Form File object from Form1.cs to LogIn Form.cs.
2. Click on **Yes** on the MessageBox to allow the Visual C# to rename all references of this form object.
3. Change the name of the Form window object from Form1 to LogInForm.

Now we need to modify the LogInForm by copying all controls from the LogInForm in the project AccessSelectRuntimeObject and pasting them into the LogInForm in our new project SQLSelectRuntimeObject. To do that, open the LogIn form window from the project AccessSelectRuntimeObject we developed in the last section, go to Edit/Select All to select all controls on that form, and then go to Edit/Copy to copy all of them. Now return to the LogInForm window in our new project SQLSelectRuntimeObject. Go to Edit/Paste to paste all copied controls to this form window.

Next we need to add the other four forms into our new project. To do that, click on the Project menu item and then the Add Existing Item; browse to the AccessSelectRuntimeObject project folder. Press and hold the Ctrl key on your keyboard and click on the other four forms one by one: SelectionForm.cs, FacultyForm.cs, CourseForm.cs, and StudentForm.cs. Click on the Add button to add these forms into our current project.

Next we need to change the following Form File object's names as shown below:

- Change the Faculty.cs to Faculty Form.cs.
- Change the Course.cs to Course Form.cs.
- Change Student.cs to Student Form.cs.

The reason we need to do this modification is that there are duplications between these Form File objects' names and the data tables' names in our database. For example, we have a LogIn Form File object named LogIn, but we also have a data table named LogIn in our CSE_DEPT database. These duplications will cause some compiling errors when we create the entity classes used for the LINQ to SQL method later since the extension classes may have the same names as those of Form File objects. In order to avoid these errors, we need to do these modifications.

Besides changing the Form File objects' names, you also need to perform the modification to change the namespace for all of these four added forms. The reason for that is the namespace in the code window of each added form is still the old one, AccessSelectRTOBJECT, since we just add them into our new project without touching any code part. In order to use these codes on each form, we must change those namespaces to our current project or namespace SQLSelectRTOBJECT.

As an example, to change the namespace for the codes on the Faculty form, perform the following operations:

1. Open the code window of the Faculty Form object.
2. Change the namespace (project name), which is located at the first code line under the system namespace declaration section, from AccessSelectRTOBJECT to SQLSelectRTOBJECT.
3. Open the Faculty Form Design file, Faculty Form.Designer.cs, which is a subfolder of the Faculty Form.cs, by double-clicking on this item.
4. Change the namespace, which is the first code line on this window, from AccessSelectRTOBJECT to SQLSelectRTOBJECT.

Perform the same operations for the three other forms, Selection Form, Course Form, and Student Form, respectively, to complete the namespace modifications. You may encounter some compiling errors [unsuitable override method Dispose()] if you did not modify these namespaces on these added forms.

5.19.2.1 Query Data Using the General Runtime Objects for LogIn Form

Open the LogInForm code window from the Solution Explorer window by clicking on the View Code button to begin our coding. The first thing we need to do is to add the namespace SqlClient, which contains all data components related to the SQL Server Data Provider since we need it to access the SQL Server databases in this project. Type the following code line in the namespace declaration section, which is in the top part, of this code window:

```
using System.Data.SqlClient;
```

An easy way to develop the codes for this section is to modify the codes we created for the LogInForm in the project AccessSelectRTOBJECT project we did in the last section. Open that project and the code window of the LogInForm, copy all codes, and paste

them into the LogInForm in this new project. Then replace the prefix `acc`, which precedes all Data Provider–dependent objects in the codes in this form, such as `accConnection`, `accCommand`, `accDataReader`, and `accDataAdapter`, with the prefix `sql`.

Note that when you copy the codes from the LogInForm in the AccessSelectRTOBJECT project and paste them into the same form in our current project SQLSelectRTOBJECT, you cannot copy and paste the whole body of the codes for each method that is created by the system. For example, the methods `cmdTabLogIn_Click()`, `cmdReadLogIn_Click()`, and `cmdCancel_Click()` are created by the Visual Studio as you double-click on the associated buttons from the design view of the LogIn form. To copy and paste the codes for these three methods, you need to perform the following operations:

1. Open the code window of the LogIn form from the AccessSelectRTOBJECT project and copy the codes inside the `cmdTabLogIn_Click()` method. This copy does not include the header and the ender of this method, but only the content of this method.
2. Open the design view of the LogIn form (LogIn form window) from our current project SQLSelectRTOBJECT, double-click on the TabLogIn button to open this method, and then paste the codes we just copied into this method.

Perform the similar operations for the other two methods to finish this copy-and-paste operation.

One of the major differences between the LogInForm in this project and the same form in the project AccessSelectRTOBJECT is the connection string; therefore, as long as the connection string is modified and matched to the selected database or the data provider, most other codes can be used for this project without any modification.

5.19.2.1.1 Connect to Data Source with Runtime Objects Since the connection job is the first thing we need to do before we can make any data query, we need to do the connection job in the constructor of this form object to allow the connection to be made first as the project runs. Open the Code window and find the constructor of this form. Enter the codes shown in Figure 5.107 into this part.

Let's see how this piece of code works.

- A. The namespace for the SQL Server Data Provider `System.Data.SqlClient` is added into this form to allow us to use all data components contained in that namespace.
- B. A global SQL Connection object `SqlConnection` is declared here. The accessing mode for this object is `Public`, which means that any object or method defined in this project can use this connection object to access the sample SQL Server database `CSE_DEPT`.
- C. An `SqlConnection String` is created here, and it is used to connect our project with the SQL Server database selected. Please note that this connection string is different from the one we created in the last project AccessSelectRTOBJECT. The `Server` parameter is assigned with a value `localhost`, which means that the SQL Server is installed in our local computer. The `Data Source` parameter is used to indicate the server name. In this case, since we install the server in our local computer, we can use the local computer's name (YBAI) or the default wild name. `\` followed by the server name (`SQLEXPRESS`) as the server name. To identify your computer's name, right-click on the My Computer icon on your computer screen, select and open the System Properties window by clicking on the View system information item from the System Tasks pane, then click on the Network Identification or Computer Name tab, and you can find your computer's full name. The Database we used for this project is an SQL sample database we developed in Chapter 2, `CSE_DEPT`.

```

SQLSelectRTOject.LogInForm | LogInForm()
.....
using System.Data;
A using System.Data.SqlClient;
using System.Data.Linq;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace SQLSelectRTOject
{
    public partial class LogInForm : Form
    {
B         public SqlConnection sqlConnection;
        public LogInForm()
        {
C             InitializeComponent();
            string sqlString = "Server=localhost;Data Source=.\SQLEXPRESS;" +
D                 "Database=C:\database\SQLServer\CSE_DEPT.mdf;Integrated Security=SSPI";
E             sqlConnection = new SqlConnection(sqlString);
            try
            {
                sqlConnection.Open();
            }
            catch (SqlException e)
            {
                MessageBox.Show("SQL Server Error");
                MessageBox.Show("Error Code = " + e.ErrorCode);
                MessageBox.Show("Error Message = " + e.Message);
            }
            catch (InvalidOperationException e)
            {
                MessageBox.Show("Invalid Message = " + e.Message);
            }
F             if (sqlConnection.State != ConnectionState.Open)
            {
                MessageBox.Show("Database connection is Failed");
                Application.Exit();
            }
        }
    }

```

Figure 5.107 Coding for the database connection.

In this application, no usernames and passwords are utilized for our database. Instead, the standard Integrated Security is used here. You can add those two pieces of information if your database did utilize those two items.

- D.** The new instance of the SqlConnection class is created and initialized with the connection string as the argument.
- E.** A try ... catch block is utilized to try to catch any mistakes caused by opening this connection. The advantage of using this kind of strategy is to avoid unnecessary system debug processes and simplify this debug procedure. Both SqlException and InvalidOperationException are checked and caught.
- F.** This step is used to confirm whether our database connection is successful. If not, an error message is displayed and the project is exited.

After a database connection is successfully made, we need to use this connection to access the SQL Server database to perform our data query job.

5.19.2.1.2 Coding Method 1: Using DataAdapter to Query Data In this section, we show the readers how to create and use the runtime objects to query the data from the SQL Server database by using the DataAdapter method. Open the LogInForm window by clicking on the View Designer button, and then double-click on the TabLogIn button to open its Click method. Enter the codes shown in Figure 5.108 into this method.

Let's see how this piece of code works.

- A. The query string for the SQL Server database, which is different from the one we used in the Microsoft Access database, is declared first. This query string is identical to the one we used for the AccessSelectRTOject project.
- B. All data components used in this method are declared and created here, which include the Command object sqlCommand, DataAdapter object sqlDataAdapter, DataTable object sqlDataTable, and the next form object selForm.
- C. The Command object is initialized with the Connection object, CommandType, and CommandText properties.
- D. Two dynamic parameters are replaced by the real parameters stored in two textboxes, txtUserName and txtPassWord, respectively. The parameter's name must be identical with the name of the dynamic parameter in the SQL statement string. The Values of two parameters should be equal to the contents of two associated textbox controls, which will be entered by the user as the project runs.

```

SQLSelectRTOject.LogInForm | cmdTabLogIn_Click()
private void cmdTabLogIn_Click(object sender, EventArgs e)
{
A   string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
   cmdString += "WHERE (user_name=@Param1 ) AND (pass_word=@Param2)";
B   SqlDataAdapter LogInDataAdapter = new SqlDataAdapter();
   DataTable sqlDataTable = new DataTable();
   SqlCommand sqlCommand = new SqlCommand();
   SelectionForm selForm = new SelectionForm();
C   sqlCommand.Connection = sqlConnection;
   sqlCommand.CommandType = CommandType.Text;
   sqlCommand.CommandText = cmdString;
D   sqlCommand.Parameters.Add("@Param1", SqlDbType.Char).Value = txtUserName.Text;
   sqlCommand.Parameters.Add("@Param2", SqlDbType.Char, 8).Value = txtPassWord.Text;
E   LogInDataAdapter.SelectCommand = sqlCommand;
   LogInDataAdapter.Fill(sqlDataTable);
F   if (sqlDataTable.Rows.Count > 0)
   {
       MessageBox.Show("LogIn is successful");
       selForm.Show();
       this.Hide();
   }
G   else
       MessageBox.Show("No matched username/password found!");
H   sqlDataTable.Dispose();
   sqlCommand.Dispose();
   LogInDataAdapter.Dispose();
}

```

Figure 5.108 Coding for the TabLogIn button Click method.

- E. The initialized Command object is then assigned to the SelectCommand property of the DataAdapter, and the Fill() method of the DataAdapter is executed to execute the Command to fill the LogIn table in the DataSet.
- F. If the Count property of the DataRow is greater than 0, which means that at least one row has been filled in the LogIn table, a successful message is displayed, and the next form window, SelectionForm, is displayed to allow users to continue to query the desired information. Then the LogInForm disappears from the screen by executing the Hide() method. You need to note that this MessageBox() is only used for the testing purpose, and it will be commented out when this project runs in the normal condition.
- G. If the Count property is 0, which means that no row has been filled in the LogIn table and this fill has failed, an error message is displayed.
- H. A cleaning job is performed to release all data components used in this method.

Now let's take a look at the codes for the second method.

5.19.2.1.3 Coding for Method 2: Using DataReader to Query Data Open the LogIn form window by clicking on the View Designer button from the Solution Explorer window, and then double-click on the ReadLogIn button to open its Click method. Enter the codes shown in Figure 5.109 into this Click method.

Most codes in the top section are identical with those codes in the TabLogIn button's Click method with two exceptions. First, a DataReader object is created to replace the DataAdapter to perform the data query, and second the DataTable is removed from this method since we do not need it for our data query.

SQLSelectRTOject.LogInForm	cmdReadLogIn_Click()
<pre> private void cmdReadLogIn_Click(object sender, EventArgs e) { A string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "; B cmdString += "WHERE (user_name=@name) AND (pass_word=@word)"; C SqlCommand sqlCommand = new SqlCommand(); D SelectionForm selForm = new SelectionForm(); E SqlDataReader sqlDataReader; F sqlCommand.Connection = sqlConnection; G sqlCommand.CommandType = CommandType.Text; H sqlCommand.CommandText = cmdString; sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = txtUserName.Text; sqlCommand.Parameters.Add("@word", SqlDbType.Char, 8).Value = txtPassWord.Text; sqlDataReader = sqlCommand.ExecuteReader(); if (sqlDataReader.HasRows == true) { MessageBox.Show("LogIn is successful"); selForm.Show(); this.Hide(); } else G MessageBox.Show("No matched username/password found!"); H sqlCommand.Dispose(); sqlDataReader.Close(); } </pre>	

Figure 5.109 Coding for the ReadLogIn button Click method.



When using the Parameters collection to add dynamic parameters, the order of the dynamic parameters is very important. The order in which you add the dynamic parameters into the Parameters collection must be identical to the order in which the nominal parameters are placed in the SQL statement string.

Let's see how this piece of code works.

- A. The only modification for this query string is that we changed two nominal dynamic parameters' names to @name and @word, respectively. This is fine for the query string as long as the same parameters' names are used for both the query string and the Add() method for the Parameters collection.
- B. Two data components, Command and DataTable, are created here and they will be used in this method.
- C. The Command object is initialized with Connection object, CommandType, and CommandText properties.
- D. Two nominal dynamic parameters are replaced by two real parameters whose values are stored in two textboxes, txtUserName and txtPassWord.
- E. The ExecuteReader() method is called to perform the data query, and the returned data should be filled in the DataReader.
- F. If the returned DataReader contains some queried data, its HasRows property should be true, and then the project should go to the next step and the Selection form should be displayed.
- G. Otherwise an error message is displayed to indicate this situation.
- H. A cleaning job is performed to release all data components used in this method.

The coding for the Cancel command button's Click method is identical with the coding we did in the last section with no modification. For your convenience, we show this coding in Figure 5.110 again. This coding is straightforward without any tricks.

The final coding is for the user-defined method getLogInForm(). The coding for this method is shown in Figure 5.111. The purpose of this method is to allow users to access the LogInForm from other objects or methods defined in this project to use the global connection sqlConnection object to access the database.

SQLSelectRTObject.LogInForm	cmdCancel_Click()
<pre>private void cmdCancel_Click(object sender, EventArgs e) { if (sqlConnection.State == ConnectionState.Open) { sqlConnection.Close(); sqlConnection.Dispose(); } Application.Exit(); }</pre>	

Figure 5.110 Coding for the Cancel button Click method.

SQLSelectRTOject.LogInForm	▼	getLogInForm()	▼
<pre>public LogInForm getLogInForm() { return this; }</pre>			

Figure 5.111 Coding for the getLogInForm method.

SQLSelectRTOject.SelectionForm	▼	cmdExit_Click()	▼
<pre>private void cmdExit_Click(object sender, EventArgs e) { LogInForm logForm = new LogInForm(); logForm = logForm.getLogInForm(); if (logForm.sqlConnection.State == ConnectionState.Open) logForm.sqlConnection.Close(); logForm.Close(); courseForm.Close(); facultyForm.Close(); studentForm.Close(); Application.Exit(); }</pre>			

Figure 5.112 Modified coding for the Exit button Click method.

5.19.2.2 Coding for Selection Form

Most coding in this form is identical with the coding of the Selection form in the last project. The only difference is the coding for the Exit command button. In the last project, a Microsoft Access 2007 database was used and all Data Provider–dependent objects were preceded by the prefix `acc` such as `accConnection`. In this project we use a SQL Server database so the connection object should be preceded by the prefix `sql`.

When the Exit button is clicked, we need to check whether the connection object has already been closed and released. Since the connection object is created in the `LogIn` class, we need first to create a new instance of the `LogInForm` class and call the method `getLogInForm()` we defined in that class to get the original object created in the `LogInForm` in the last section. Then we can assign the returned original `LogInForm` object to the new created instance. In this way, we can access and use the global connection object via the original `LogInForm` object. The only modification you need to do is to change the prefix `acc` to `sql` for the connection instance, which is shown in Figure 5.112.

5.19.2.3 Query Data Using General Runtime Objects for Faculty Form

First let's add a new namespace `System.Data.SqlClient` that contains the SQL Data Provider to the namespace declaration part in the code window of this form. This new added namespace has been highlighted in Figure 5.113. Most parts of the coding for the constructor of the `FacultyForm` is identical with the coding in the same form we did in `AccessSelectRTOject`. For your convenience, we show the modified codes for this part in Figure 5.113.

```

SQLSelectRTOject.FacultyForm | FacultyForm()
.....
A using System.Data;
  using System.Data.SqlClient;
  using System.Data.OleDb;
  using System.Drawing;
  using System.Linq;
  using System.Text;
  using System.Windows.Forms;
namespace SQLSelectRTOject
{
  public partial class FacultyForm : Form
  {
    private Label[] FacultyLabel = new Label[7];
    public FacultyForm()
    {
      InitializeComponent();
      ComboName.Items.Add("Ying Bai");
      ComboName.Items.Add("Satish Bhalla");
      ComboName.Items.Add("Black Anderson");
      ComboName.Items.Add("Steve Johnson");
      ComboName.Items.Add("Jenney King");
      ComboName.Items.Add("Alice Brown");
      ComboName.Items.Add("Debby Angles");
      ComboName.Items.Add("Jeff Henry");
      ComboName.SelectedIndex = 0;
      B //this.cmdSelect_Click(this.cmdSelect, null);
      C ComboMethod.Items.Add("DataAdapter Method");
      ComboMethod.Items.Add("DataReader Method");
      this.ComboMethod.SelectedIndex = 0;
    }
  }
}

```

Figure 5.113 Modified coding for the constructor of the FacultyForm.

- A. A new namespace `System.Data.SqlClient`, which contains the SQL Data Provider-dependent objects is added into this code window since we need to use those data components for this method.
- B. Remove the instruction of calling the Select button's Click method since we do not need it in this project.
- C. Because of the limitation on accessing the same SQL Server Express database simultaneously among multiple users, we will develop a separate project to discuss the LINQ to SQL method. Therefore we can remove this method from this constructor now.

Our next coding is for the Select button Click method. When this button is clicked, the faculty name selected by the user from the ComboName box will be used as the query criterion to retrieve back all related information for that selected faculty in the five labels and display the matched faculty image in a PictureBox control PhotoBox. Most of codes in this method are identical with those in the same form we developed for `AccessSelectRTOject`. The main difference is the names of the Data Provider-dependent objects used in this method.

Open the FacultyForm window by clicking on the View Designer button from the Solution Explorer window, and then double-click on the Select button to open its Click method. Make the following modifications to the original codes and your finished coding should match that shown in Figure 5.114.

```

SQLSelectRTObject.FacultyForm cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
A   string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty ";
B   cmdString += "WHERE faculty_name LIKE @name";
   SqlDataAdapter FacultyDataAdapter= new SqlDataAdapter();
   SqlCommand sqlCommand = new SqlCommand();
   SqlDataReader sqlDataReader;
   DataTable sqlDataTable = new DataTable();
   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();
C   sqlCommand.Connection = logForm.sqlConnection;
   sqlCommand.CommandType = CommandType.Text;
   sqlCommand.CommandText = cmdString;
D   sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = ComboName.Text;
   string strName = ShowFaculty(ComboName.Text);
   if (strName == "No Match")
       MessageBox.Show("No Matched Faculty Image found!");
   if (ComboMethod.Text == "DataAdapter Method")
   {
       FacultyDataAdapter.SelectCommand = sqlCommand;
       FacultyDataAdapter.Fill(sqlDataTable);
       if (sqlDataTable.Rows.Count > 0)
           FillFacultyTable(ref sqlDataTable);
       else
           MessageBox.Show("No matched faculty found!");
       sqlDataTable.Dispose();
       FacultyDataAdapter.Dispose();
   }
   else if (ComboMethod.Text == "DataReader Method")
   {
       sqlDataReader = sqlCommand.ExecuteReader();
       if (sqlDataReader.HasRows == true)
           FillFacultyReader(sqlDataReader);
       else
           MessageBox.Show("No matched faculty found!");
       sqlDataReader.Close();
E   }
   else //Invalid method selected
       MessageBox.Show("Invalid Method Selected!");
}
}

```

Figure 5.114 Modified coding for the Select button Click method.

Let's have a closer look at this piece of code to see how it works.

- A.** The query string is basically identical with the one we used for the last project and the only modification is that we changed the nominal name of the dynamic parameter from @Param1 to @name in the WHERE clause. Also you have to change the assignment operator from = to LIKE for the WHERE clause since this is the requirement of data query operation in the SQL Server database.
- B.** The names of all Data Provider–dependent classes have been modified by changing the prefix from OleDb to Sql, and the names for all Data Provider–dependent objects have also been modified by changing the prefix from acc to sql, respectively. The DataSet object has been removed since we do not need it in this method.
- C.** The name of the Command object has been modified by changing the prefix from acc to sql.

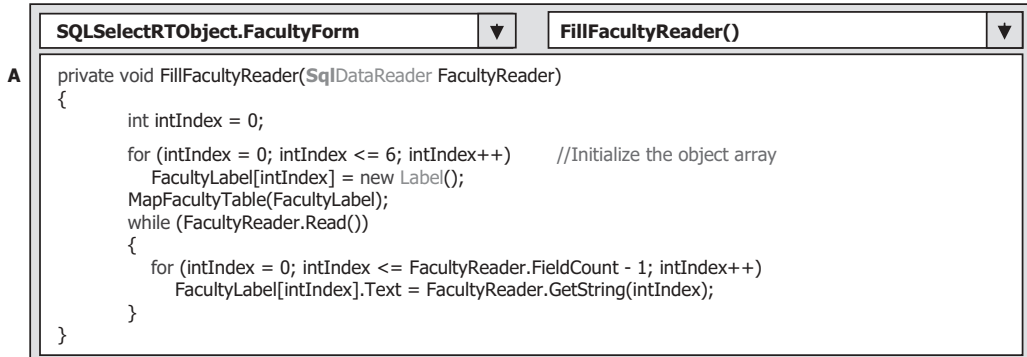


Figure 5.115 Modified coding for the FillFacultyReader method.

- D.** The data type of the dynamic parameter has been changed from the OleDbType to SqlDbTypeType since a SQL Server database is used for this project.
- E.** The LINQ to SQL method has been removed from this piece of codes since we need to develop a separate project to handle that issue later.

The following codes are similar to those codes we developed in the same method for the last project. The only modifications you need to perform are change the prefix from acc to sql, for example, accCommand to sqlCommand, accDataTable to sqlDataTable, and accDataReader to sqlDataReader.

Yes, it is that easy!

Now let's take care of the coding for four user-defined methods, FillFacultyTable(), MapFacultyTable(), ShowFaculty(), and FillFacultyReader(). For the first three methods, there is no modification to be made at all. For the fourth method FillFacultyReader(), the only modification is to change the nominal argument's type from OleDbDataReader to SqlDataReader. Figure 5.115 (A) shows this modification.

The coding for the Back button Click method is identical with the coding we did for the last project with no any modification. Refer to Figure 5.88 for this detailed coding and the explanations.

You can test the FacultyForm by running the project now. However, before you can do this, you need to perform the following operations:

- Copy all faculty and student image files from the folder Images located at the accompanying ftp site (see Chapter 1) and paste these image files into any folder on your computer. The ideal folder is the default folder, which is the folder in which your project executable file is located. In this application, it is C:\Book6\Chapter 5\SQLSelectRTOject\bin\Debug.
- Comment out two testing MessageBox() methods in the LogInForm code window.

Now you can run the project to test the functionalities of the Faculty Form object.

5.19.2.4 Query Data Using General Runtime Objects for Course Form

As we did for the Faculty Form, first we need to add a new namespace System.Data.SqlClient that contains the SQL Data Provider to the namespace declaration part in the code window of this form.

```

.....
using System.Data;
A using System.Data.SqlClient;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace SQLSelectRTOject
{
    public partial class CourseForm : Form
    {
        private TextBox[] CourseTextBox = new TextBox[6];
        B //DataSet ds = new DataSet();
        public CourseForm()
        {
            InitializeComponent();
            ComboName.Items.Add("Ying Bai");
            ComboName.Items.Add("Satish Bhalla");
            ComboName.Items.Add("Black Anderson");
            ComboName.Items.Add("Steve Johnson");
            ComboName.Items.Add("Jenney King");
            ComboName.Items.Add("Alice Brown");
            ComboName.Items.Add("Debby Angles");
            ComboName.Items.Add("Jeff Henry");
            ComboName.SelectedIndex = 0;
            ComboMethod.Items.Add("DataAdapter Method");
            ComboMethod.Items.Add("DataReader Method");
            C //ComboMethod.Items.Add("LINQ & DataSet Method");
            ComboMethod.SelectedIndex = 0;
        }
    }
}

```

Figure 5.116 Modified coding for the constructor of the CourseForm.

Then let's do the coding for the constructor of the CourseForm class. Basically, the coding of this constructor is similar to that we did for the same constructor in the last project. The only modifications are (refer to Figure 5.116):

- A.** The System.Data.SqlClient namespace is added since we need to use a group of data components that are SQL Server Data Provider dependent in this section.
- B.** The DataSet instance is removed since we do not need this component in this project, and we will develop a separate project to handle the LINQ to SQL.
- C.** Remove the LINQ method with the same reason as in step **B**.

The next coding is for the Select button Click method. This coding is very similar to that we did for the CourseForm class in the last project AccessSelectRTOject. However, one significant coding improvement compared with the last one is that an inner join query is utilized to simplify the data query. Recall that in the last project, we used two queries to retrieve back all courses (course_id) taught by the selected faculty in the Course form. This is because there is no faculty name column available in the Course table, and each course or course_id is related to a faculty_id in the Course table. In order to get the faculty_id associated with the selected faculty name, we must first perform a query to the Faculty table to obtain it. In order to simplify this data action, a join query is used in this method and it is a desired method to complete this functionality.

Table 5.8 Part of Faculty and Course Data Table**Faculty Table**

faculty_id	faculty_name	office
A52990	Black Anderson	MTC-218
A77587	Debby Angles	MTC-320
B66750	Alice Brown	MTC-257
B78880	Ying Bai	MTC-211
H99118	Jeff Henry	MTC-336
J33486	Steve Johnson	MTC-118
K69880	Jenney King	MTC-324

Course Table

course_id	faculty_id	classroom
CSC-131A	A52990	TC-109
CSC-131C	A52990	TC-109
CSC-132A	J33486	TC-303
CSC-132B	B78880	TC-302
CSC-230	A77587	TC-301
CSC-232B	A77587	TC-303
CSC-233A	H99118	TC-302

5.19.2.5 Retrieve Data from Multiple Tables Using Joined Tables Method

To have a clear picture of why we need to use the Join query method for this data action, let's first look at the data structure in our sample database. A part of the Faculty and Course data table in the CSE_DEPT database is shown in Table 5.8.

The `faculty_id` in the Faculty table is a primary key, but it is a foreign key in the Course table. The relationship between the Faculty and the Course table is one-to-many. We want to pick up all `course_id` data from the Course table based on the selected faculty name located in the Faculty table. The problem is that no faculty name is available in the Course table, and we cannot directly get all `course_id` data based on the faculty name. An efficient way to do this is to use a query with two joined tables, which means that we need to perform a query by joining two different tables—Faculty and Course to pick up the `course_id` information. To join these two tables, we need to use the primary key and the foreign key, `faculty_id`, to set up this relationship. In other words, we want to obtain all courses, that is, all `course_id`, from the Course table based on the faculty name in the Faculty table. But in the Course table, we only have course name and the associated `faculty_id` information available. Similarly, in the Faculty table, we only have faculty name and the associated `faculty_id` information available. The result is: We cannot set up a direct relationship between the faculty name in the Faculty table and the `course_id` in the Course table, but we can build an indirect relationship between them via `faculty_id` since the `faculty_id` works as a bridge to connect two tables together using the primary and foreign keys.

An SQL statement with two joined tables, Faculty and Course, can be represented as:

```
SELECT Course.course_id, Course.course FROM Course, Faculty
WHERE (Course.faculty_id LIKE Faculty.faculty_id) AND
(Faculty.faculty_name LIKE @name)
```

The `@name` is a dynamic parameter, and it will be replaced by the real faculty name selected by the user as the project runs.

One point to note is that the syntax of this SQL statement is defined in the ANSI 89 standard and is relatively out of date. Microsoft will not support this out-of-date syntax in the future. So it is highly recommended to use a new syntax for this SQL statement, which is defined in the ANSI 92 standard and it looks like:


```
SELECT Course.course_id, Course.course FROM Course JOIN
Faculty ON (Course.faculty_id LIKE Faculty.faculty_id) AND
(Faculty.faculty_name LIKE @name)
```

Now let's use this inner join method to develop our query for this method. The modified codes are shown in Figure 5.117.

Let's take a closer look at those modified codes in detail:

- A. The joined table query string is declared at the beginning of this method. Here two columns are queried. The first one is the `course_id` and the second is the course name. The reason for this is that we need to use the `course_id`, not course name, as the identifier to pick up each course's detailed information from the `Course` table when the user clicks and selects the `course_id` from the `CourseList` box. Actually, we only need the `course_id` column for this query, but it does not matter if other columns such as the course is included in this query. The assignment operator `LIKE` is used to replace the original equals symbol

SQLSelectRTOBJECT.CourseForm	cmdSelect_Click()
<pre>private void cmdSelect_Click(object sender, EventArgs e) { A string strCourse = "SELECT Course.course_id, Course.course FROM Course JOIN Faculty "; B strCourse += "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.faculty_name LIKE @name)"; B SqlDataAdapter CourseDataAdapter = new SqlDataAdapter(); sqlCmdCourse = new SqlCommand(); DataTable sqlCourseTable = new DataTable(); LogInForm logForm = new LogInForm(); logForm = logForm.getLogInForm(); SqlDataReader sqlDataReader; C sqlCmdCourse.Connection = logForm.sqlConnection; sqlCmdCourse.CommandType = CommandType.Text; sqlCmdCourse.CommandText = strCourse; D sqlCmdCourse.Parameters.Add("@name", SqlDbType.Char).Value = ComboName.Text; if (ComboMethod.Text == "DataAdapter Method") { CourseDataAdapter.SelectCommand = sqlCmdCourse; CourseDataAdapter.Fill(sqlCourseTable); if (sqlCourseTable.Rows.Count > 0) FillCourseTable(sqlCourseTable); else MessageBox.Show("No matched course found!"); sqlCourseTable.Dispose(); CourseDataAdapter.Dispose(); } else if (ComboMethod.Text == "DataReader Method") { sqlDataReader = sqlCmdCourse.ExecuteReader(); if (sqlDataReader.HasRows == true) FillCourseReader(sqlDataReader); else MessageBox.Show("No matched course found!"); sqlDataReader.Close(); sqlDataReader.Dispose(); } E else //Invalid Method is selected MessageBox.Show("Invalid Method Selected!"); sqlCmdCourse.Dispose(); CourseList.SelectedIndex = 0; }</pre>	

Figure 5.117 Modified coding for the Select button Click method.

for the criteria in the ON clause in the definition of the query string, and this is required by the SQL Server database operation.

- B.** All Data Provider–dependent objects, such as the CourseDataAdapter, Command, DataReader, and DataTable, should be prefixed by sql to indicate that all those components are related to the SQL Server Data Provider.
- C.** The sqlCommand object is initialized with the connection string, command type, command text, and command parameter. The parameter’s name must be identical with the dynamic nominal name @name, which is defined in the query string, and it is exactly located after the LIKE comparator in the ON clause. The parameter’s value is the content of the Faculty Name combobox, which should be selected by the user as the project runs.
- D.** The following codes are similar to those we developed in the last project. If the DataAdapter method is selected by the user, the Fill() method of the DataAdapter is executed to fill the Course table. The FillCourseTable() method is then called to fill the course_id into the CourseList box. Change the prefix for all Data Provider–dependent objects from acc to sql. Also change the data type of the nominal argument of the dynamic parameter @name in the Add() method from OleDbType to SqlDbType.
- E.** Remove the coding for the LINQ to DataSet method since we do not use this method in this coding. Finally some cleaning jobs are preformed to release objects used for this query.

The user-defined method FillCourseTable() and the Back button Click method have nothing to do with any object used in this project, so no modification is needed. The method FillCourseReader() needs only a small modification, which is to change the nominal argument’s type from the OleDbDataReader to SqlDataReader since now we are using an SQL Server data provider. The detailed explanations for methods FillCourseTable() and FillCourseReader() can be found in Figure 5.92.

Next we need to take care of the coding for the CourseList_SelectedIndexChanged() method. All detailed information related to the selected course_id from the CourseList box should be displayed in five textbox controls when the user clicks and selects a course_id from the CourseList box control. The coding for this method is similar with the one we did in the last project with the following modifications (Refer to Figure 5.118).

Let’s have a closer look at these modified codes to see how they work.

- A.** The query string is created with six queried columns such as course, credit, classroom, schedule, enrollment, and course_id. The query criterion is course_id. The reason why we query the course_id by using the course_id as a criterion is that we want to make this query complete and neat. The comparator LIKE is used to replace the original equals symbol for the criteria in the WHERE clause in the definition of the query string, and this is required by the SQL Server database operation. Also the nominal name of the dynamic parameter is changed to @courseid.
- B.** All data components related to SQL Server Data Provider are created, and these objects are used to perform the data operations between the database and our project. All of these classes should be prefixed by Sql and all objects should be prefixed by sql since in this project we used a SQL Server data provider.
- C.** The sqlCommand object is initialized with the connection string, command type, command text, and command parameter.
- D.** The parameter’s name must be identical with the dynamic nominal name @courseid, which is defined in the query string, exactly after the LIKE comparator in the WHERE clause. The parameter’s value is the course_id in the CourseList listbox control.

```

SQLSelectRTOject.CourseForm CourseList_SelectedIndexChanged()
private void CourseList_SelectedIndexChanged(object sender, EventArgs e)
{
A   string cmdString = "SELECT course, credit, classroom, schedule, enrollment, course_id FROM Course ";
B   cmdString += "WHERE course_id LIKE @courseid";
   SqlCommand sqlCommand = new SqlCommand();
   SqlDataAdapter CourseDataAdapter = new SqlDataAdapter();
   DataTable sqlDataTable = new DataTable();
   SqlDataReader sqlDataReader;
   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();
C   sqlCommand.Connection = logForm.sqlConnection;
   sqlCommand.CommandType = CommandType.Text;
   sqlCommand.CommandText = cmdString;
D   sqlCommand.Parameters.Add("@courseid", SqlDbType.Char).Value = CourseList.SelectedItem;
   if (ComboMethod.Text == "DataAdapter Method")
   {
E       CourseDataAdapter.SelectCommand = sqlCommand;
       CourseDataAdapter.Fill(sqlDataTable);
       if (sqlDataTable.Rows.Count > 0)
           FillCourseTextBox(sqlDataTable);
       else
           MessageBox.Show("No matched course information found!");
       sqlDataTable.Dispose();
       CourseDataAdapter.Dispose();
   }
F   else if (ComboMethod.Text == "DataReader Method")
   {
       sqlDataReader = sqlCommand.ExecuteReader();
       if (sqlDataReader.HasRows == true)
           FillCourseReaderTextBox(sqlDataReader);
       else
           MessageBox.Show("No matched course information found!");
       sqlDataReader.Close();
       sqlDataReader.Dispose();
   }
G   else //Invalid Method is selected
       MessageBox.Show("Invalid Method Selected!");
       sqlCommand.Dispose();
}

```

Figure 5.118 Modified codes for the CourseList_SelectedIndexChanged method.

- E.** If the DataAdapter method is selected by the user, the Fill() method is called to fill the Course table, and the user-defined method FillCourseTextBox() is executed to fill five textboxes to display the detailed course information for the selected course_id from the CourseList box.
- F.** Otherwise the DataReader method is selected. The ExecuteReader() method is executed to read back the detailed information for the selected course_id, and the user-defined method FillCourseReaderTextBox() is called to fill those pieces of information into the five textboxes.
- G.** The LINQ to DataSet method is removed from this method since we will develop a separate project to handle the data query job using the LINQ to SQL later.

Change the prefix for all Data Provider–dependent objects from acc to sql.

Two user-defined methods, FillCourseTextBox() and MapCourseTable(), have no relationship with any object used in this project, therefore no coding modification is needed. The method FillCourseReaderTextBox() needs a small modification, which is to

change the nominal argument's type from the `OleDbDataReader` to the `SqlDataReader` since a SQL Server data provider is utilized in this project. For the detailed line-by-line explanations of the methods `FillCourseTextBox()`, `FillCourseReaderTextBox()`, and `MapCourseTable()` refer to Figure 5.94.

You can test the codes we just developed for the `CourseForm` class by running the project now. Do not forget to copy all faculty and student image files to the folder in which your Visual C# project executable file is located before you can run this project. In this application, it is the `Debug` folder of the project.

5.19.2.6 Query Data Using General Runtime Objects for Student Form

Now let's finally come to the coding for the `Student` form window. The `Student` form window is shown in Figure 5.119 again for your convenience.

The function for this form is to pick up all pieces of information related to the selected student, such as the student id, gpa, credits, major, schoolYear, and email, and display them in six textboxes when the `Select` button is clicked by the user. Also the courses (course_id) taken by that student are displayed in the `CourseList` box. Apparently, this function needs to make two queries to the two different tables, the `Student` and the `StudentCourse` tables, respectively.

The codes for this form are similar to those we did for the `Faculty` form with one important difference, which is the query type. In order to improve the querying efficiency and make the coding simplified, two stored procedures are developed and implemented in this section. By using the stored procedures, the query coding can be significantly simplified and integrated, and the efficiency of the data query can also be improved.

First let's add a new namespace `System.Data.SqlClient` that contains the SQL Server Data Provider–dependent objects to the namespace declaration part in this form.

Now let's start our coding from the constructor of the `StudentForm`. Generally we need to perform all initialization jobs for this form object in this constructor. Open the

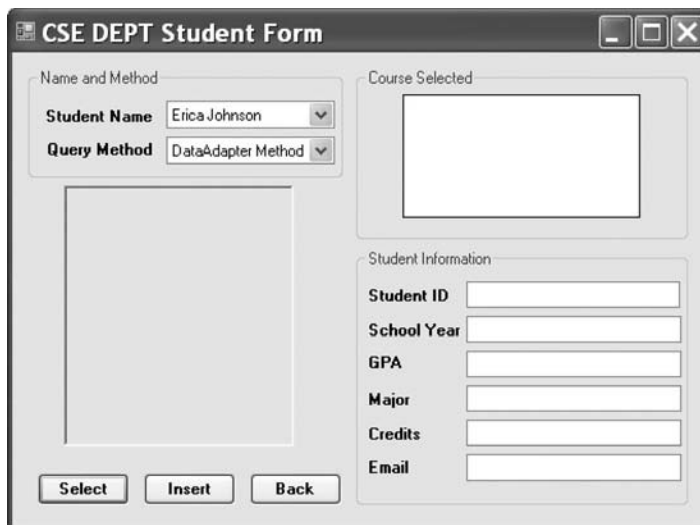


Figure 5.119 Student form window.

```

SQLSelectRTOject.StudentForm StudentForm_Load()
.....
using System.Data;
A using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace SQLSelectRTOject
{
    public partial class StudentForm : Form
    {
B        //DataSet ds = new DataSet();
C        private TextBox[] StudentTextBox = new TextBox[7]; //We query 7 columns from the Student table
        public StudentForm()
        {
D            InitializeComponent();
            ComboName.Items.Add("Erica Johnson");
            ComboName.Items.Add("Ashly Jade");
            ComboName.Items.Add("Holes Smith");
            ComboName.Items.Add("Andrew Woods");
            ComboName.Items.Add("Blue Valley");
            ComboName.SelectedIndex = 0;
E            ComboMethod.Items.Add("DataAdapter Method");
            ComboMethod.Items.Add("DataReader Method");
            ComboMethod.SelectedIndex = 0;
F            cmdSelect_Click(this.cmdSelect, null);
        }
    }

```

Figure 5.120 Coding for the Form_Load method.

code window of the StudentForm and the constructor of the StudentForm. Enter the codes shown in Figure 5.120 into this class method.

- A.** The namespace of the SQL Server data class library is added to provide the prototypes of all data components to be created and used in this method.
- B.** Remove the DataSet object since we do not have the LINQ to DataSet method in this part.
- C.** A class-level or field textbox array StudentTextBox[] is declared, and it is used to hold the detailed student's information as the project runs, and those pieces of information will be displayed in six textboxes in the Student form later.
- D.** All sampled students' names are added into the student name combobox, and the default student's name is the first one in this combo box by setting the SelectedIndex property to 0.
- E.** Two query methods, DataAdapter and DataReader, are added into the ComboMethod combobox to allow users to select either one to perform the associated data query as the project runs. The first method is selected as the default one by setting the SelectedIndex property to 0.
- F.** This code line is equivalent to clicking on the Select button to trigger the Select button's Click method. By adding this line, all information related to the default student can be displayed as this form is opened at the first time.

Next let's take a look at the coding for the Select button Click method. As we mentioned at the beginning of this section, when this Select button is clicked by the user, six

pieces of student information are displayed in six related textboxes, and the courses (course_id) taken by that student are displayed in the CourseList box. Two queries are needed for this operation. In order to save time and space, two stored procedures are developed for these two queries. Let's go a little deeper for the stored procedure.

5.19.2.7 Query Data Using Stored Procedures

Stored procedures are nothing more than functions or procedures applied in any project developed in any programming language. This means that stored procedures can be considered as functions or subroutines, and they can be called easily with any arguments and they can also return any data with certain type. One can integrate multiple SQL statements into a single stored procedure to perform multiple queries at a time, and those statements will be precompiled by the SQL Server to form an integrated target body. In this way, the precompiled body is insulated with your coding developed in the Visual C# environment. You can easily call the stored procedure from your Visual C# 2008 project as the project runs. The result of using the stored procedure is that the performance of our data-driven application can be greatly improved and the data query's speed can be significantly faster. Also when you develop a stored procedure, the database server automatically creates an execution plan for that procedure, and the developed plan can be updated automatically whenever a modification is made to that procedure by the database server.

Regularly there are three types of stored procedures: system stored procedures, extended stored procedures, and custom stored procedures. The system stored procedures are developed and implemented for administrating, managing, configuring, and monitoring the SQL server. The extended stored procedures are developed and applied in the dynamic linked library (dll) format. This kind of stored procedures can improve the running speed and save the running space since they can be dynamically linked to your project. The custom stored procedures are developed and implemented by users for their applications.

5.19.2.7.1 Create Stored Procedure Six possible ways can be used to create a stored procedure.

1. Using SQL Server Enterprise Manager
2. Using Query Analyzer
3. Using ASP Code
4. Using Visual Studio.NET—Real Time Coding Method
5. Using Visual Studio.NET—Server Explorer
6. Using Enterprise Manager Wizard

For Visual C# developers, I prefer to use the Server Explorer in Visual Studio.NET. A more complicated but flexible way to create the stored procedure is to use the real-time coding method from Visual Studio.NET. In this section, we will concentrate on the fifth method listed above. A typical prototype or syntax of creating a stored procedure is shown in Figure 5.121.

For SQL Server database, the name of the stored procedure is always prefixed by **dbo**. A sample stored procedure StudentInfo is shown in Figure 5.122.

```

CREATE PROCEDURE Stored Procedure's name
{
    @Param1's name  Param1's data type Input/Output,
    @Param2's name  Param2's data type Input/Output
    .....
}
AS
(DECLARE Your local variables.... If you have)
(Your SQL Statements)
RETURN

```

Figure 5.121 Prototype of a SQL Server stored procedure.

```

CREATE PROCEDURE dbo.StudentInfo
{
    @StudentName VARCHAR(50)
}
AS
SELECT student_id FROM Student
WHERE name LIKE @StudentName
RETURN

```

Figure 5.122 Sample SQL Server stored procedure.

The parameters declared inside the braces are either input or output parameters used for this stored procedure, and an @ symbol must be prefixed before the parameter in the SQL Server database. Any argument sent from the calling procedure to this stored procedure should be declared here. The other variables, which are created by using the keyword DECLARE located after the keyword AS, are local variables and they can only be used in this stored procedure. The keyword RETURN is used to return the queried data columns.

5.19.2.7.2 Call Stored Procedure When the stored procedure is created, it is ready to be called by the project developed in Visual C# 2008. You can use any possible ways to finish this calling. For example, you can use the Fill() method defined in the DataAdapter to fill a data table or you can use the ExecuteReader() method to return the reading result to a DataReader object. The above methods are good for the single-table query, which means that a group of SQL statements defined in that stored procedure are executed for only one data table. If you want to develop a stored procedure that makes multiple queries with multiple data tables, you need to use the ExecuteNonQuery() method.

To call a developed stored procedure from a Visual C# 2008 project, one needs to follow the syntax described below [Fill() method in the DataAdapter]:

1. Create a Connection object and open it.
2. Create a Command object and initialize it.
3. Create any Parameter object and add it into the Command object.
4. Execute the stored procedure by using the Fill() method in the TableAdapter class.

```

A  SqlDataAdapter StudentDataAdapter = new SqlDataAdapter();
   SqlCommand sqlCmdStudent = New SqlCommand();
   DataTable sqlStudentTable = new DataTable();

B  sqlCmdStudent.Connection = LogInForm.sqlConnection;
C  sqlCmdStudent.CommandType = CommandType.StoredProcedure;
D  sqlCmdStudent.CommandText = "dbo.StudentInfo";
E  sqlCmdStudent.Parameters.Add("@StudentName", SqlDbType.Char).Value = ComboName.Text;
   StudentDataAdapter.SelectCommand = sqlCmdStudent;
F  StudentDataAdapter.Fill(sqlStudentTable);
   if (sqlStudentTable.Rows.Count > 0)
       Collect the retrieved data columns.....
   else
       MessageBox.Show("No matched student found!");

```

Figure 5.123 Example of calling the stored procedure.

Figure 5.123 shows example code that illustrates how to call a stored procedure named `dbo.StudentInfo` (assuming a `Connection` object has been created).

- A. Some useful data components are declared here such as the `DataAdapter`, `Command`, and `DataTable`.
- B. The `Command` object is initialized by assigning the associated components to it. The first component is the `Connection` object.
- C. In order to execute a stored procedure, the keyword `StoredProcedure` must be used here and assigned to the `CommandType` property of the `Command` object to indicate that a stored procedure will be called when this `Command` object is executed.
- D. The name of the stored procedure must be assigned to the `CommandText` property of the `Command` object. This name must be identical with the name you used when you created the stored procedure.
- E. The stored procedure `dbo.StudentInfo` needs one input parameter `StudentName`. Thus, a real parameter that will be obtained from the `Student Name` combobox as the project runs is added into the `Parameters` collection, which is a property of the `Command` object. The initialized `Command` object is assigned to the `SelectCommand` property of the `DataAdapter` and it will be used later.
- F. The `Fill()` method is executed to call the stored procedure and fill the `Student` table. If this calling is successful, the returned data columns will be available, otherwise, an error message is displayed.

In the next section, we use our `Student` form to illustrate how to create two stored procedures and how to call them from Visual C# 2008 project.

5.19.2.7.3 Query Data Using Stored Procedures for Student Form First let's create two stored procedures for this `Student` form. The first stored procedure is used to get the `student_id` from the `Student` table based on the selected student name. The second one is used to obtain the courses taken by the selected student based on the `student_id`. We need to use two queries because we want to query all courses taken by the selected student based on the student's name, not the `student_id`, from the `StudentCourse` table. But there is only one `student_id` column available in the `StudentCourse` table and no student name in that table. The student name can only be obtained from the `Student`

table. So we need first to make a query to the Student table to get the student_id based on the student's name, and then make the second query to the StudentCourse table to get all courses (all course_id) based on the student_id.

The first stored procedure is named dbo.StudentInfo, and we will create this stored procedure using the Server Explorer in the Visual Studio.NET environment.

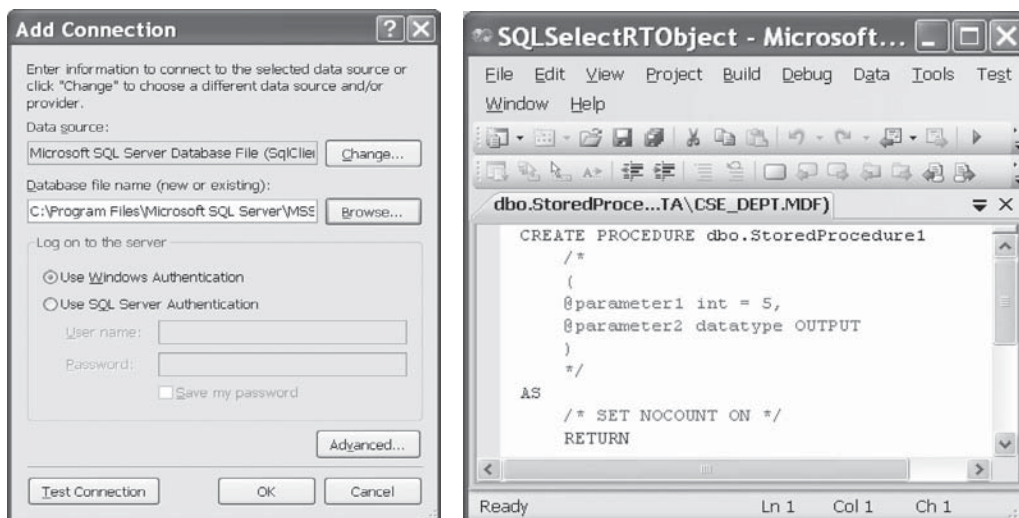
Open the Visual Studio.NET 2008 and open the Server Explorer window by clicking on View/Server Explorer menu item. To open our database CSE_DEPT, right-click on the Data Connections from the Server Explorer window and select the Add Connection item from the pop-up menu. On the opened Add Connection dialog box, perform the following actions to connect to our database:

1. Click on the Change button that is next to the Data source box.
2. Browse to and select the Microsoft SQL Server Database File item and click on OK.
3. Click on the Browse button to go to our database file folder: C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data, and select our database file CSE_DEPT.mdf by clicking on it, and then click on the Open button.
4. Clicking on the Test Connection button to confirm this connection.

Your finished Add Connection dialog box should match the one shown in Figure 5.124a.

The Use Windows Authentication radio button is selected since we want to use this security mode as our logon security checking method. On the opened Server Explorer window, you can find that our database CSE_DEPT has been connected to the server. Now let's begin to create our first stored procedure.

Right-click on the Stored Procedures folder and select the Add New Stored Procedure item to open a new stored procedure window, which is shown in Figure 5.124b.



(a)

(b)

Figure 5.124 Add Connection dialog box. www.FDL.ir

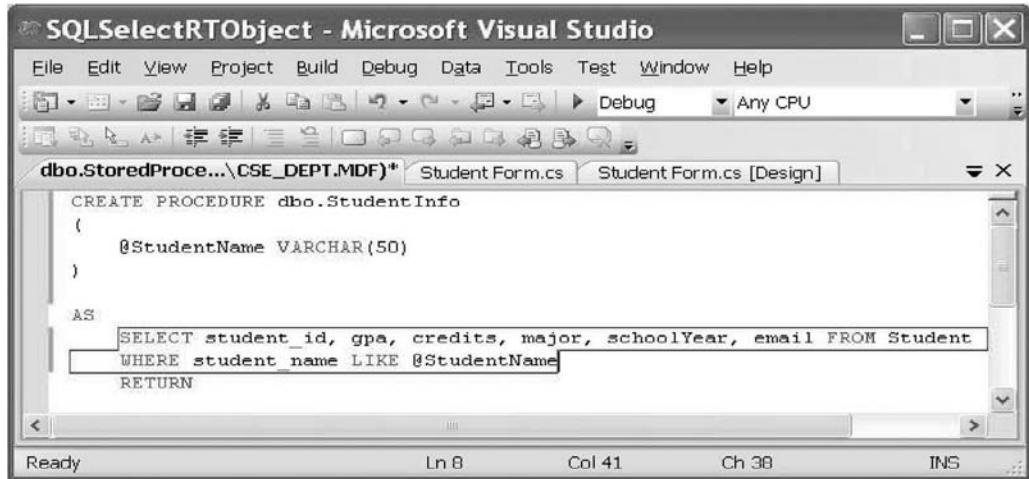


Figure 5.125 First stored procedure: dbo.StudentInfo.

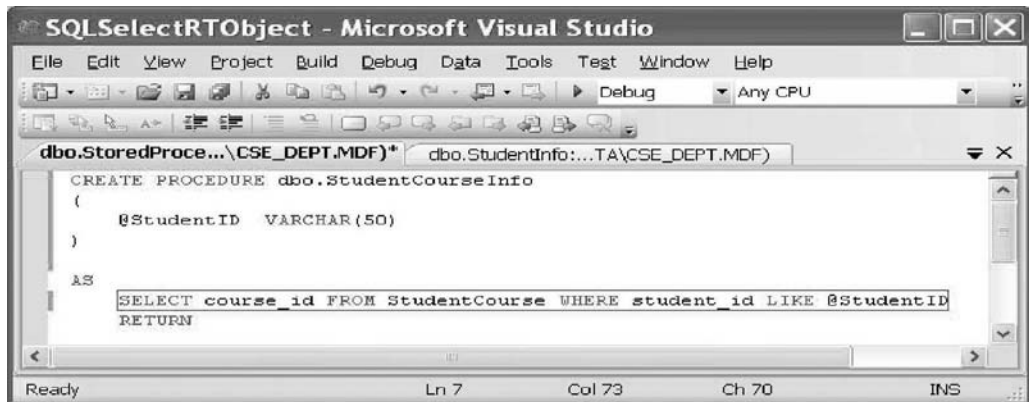


Figure 5.126 Second stored procedure: dbo.StudentCourseInfo.

The default name for a new stored procedure is `dbo.StoredProcedure1`, which is located immediately after the keyword `CREATE PROCEDURE`. The top green-color coding, which is commented out by the comment symbol `(/*...*/)`, is used to create the parameters or parameter list. The bottom green-color code is used to create the SQL statements under the keyword `AS`. To create our first stored procedure, remove the comment-out symbols and enter the codes into this procedure shown in Figure 5.125.

The `@StudentName` is our only input parameter to this stored procedure, and this stored procedure will return six pieces of information related to the selected student based on the input student name parameter. Don't forget to modify the stored procedure's name to `dbo.StudentInfo`. Now click the `File|Save StoredProcedure1` item to save our first stored procedure.

In a similar way, we can create our second stored procedure named `dbo.StudentCourseInfo`, which is shown in Figure 5.126.

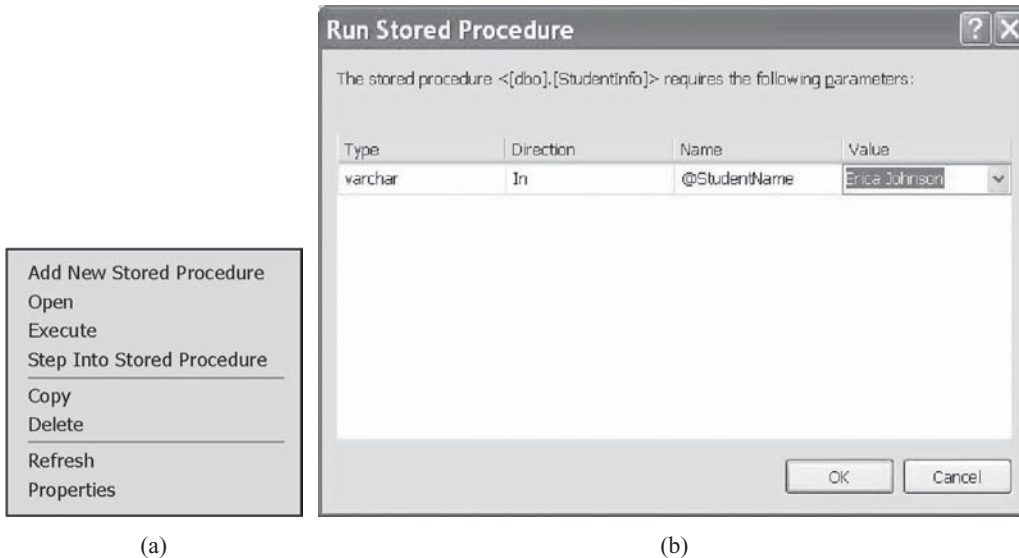


Figure 5.127 Pop-up menu and EXECUTE dialog box.

The only input parameter to this stored procedure is @StudentID, and this stored procedure will return all courses (course_id) taken by the selected student based on the input parameter student_id. Click on the File|Save StoredProcedure1 to save our second stored procedure.

Now we have finished creating our two stored procedures. Next we need to develop the coding for our Select command button located in the Student form window in Visual C# environment to call these two stored procedures.

But wait a moment. Before we can continue to develop our Visual C# 2008 coding to call these two stored procedures, is there any way for us to check whether these two stored procedures work fine or not? The answer is yes! The Server Explorer in Visual Studio.NET allows us to debug and test custom stored procedures by using some pop-up menu items, as shown in Figure 5.127a.

Open the Visual Studio.NET 2008 and connect the project to our database CSE_DEPT from the Server Explorer window. Expand to Stored Procedure folder and select one of our stored procedures, and then right-click on that selected stored procedure, a pop-up menu will be displayed, as shown in Figure 5.127a. The functionality for each item is explained below:

- 1. Add New Stored Procedure** Create a new stored procedure. We have used this item to create our two stored procedures before.
- 2. Open** Open an existing stored procedure to allow it to be edited or modified. The name of the modified stored procedure will be prefixed by ALTER.
- 3. Execute** Execute a stored procedure. One can debug and test a developed stored procedure using this item in the Server Explorer environment to make sure that the developed stored procedure works fine.
- 4. Step into Stored Procedure** Allow users to debug and run the developed stored procedure step by step.

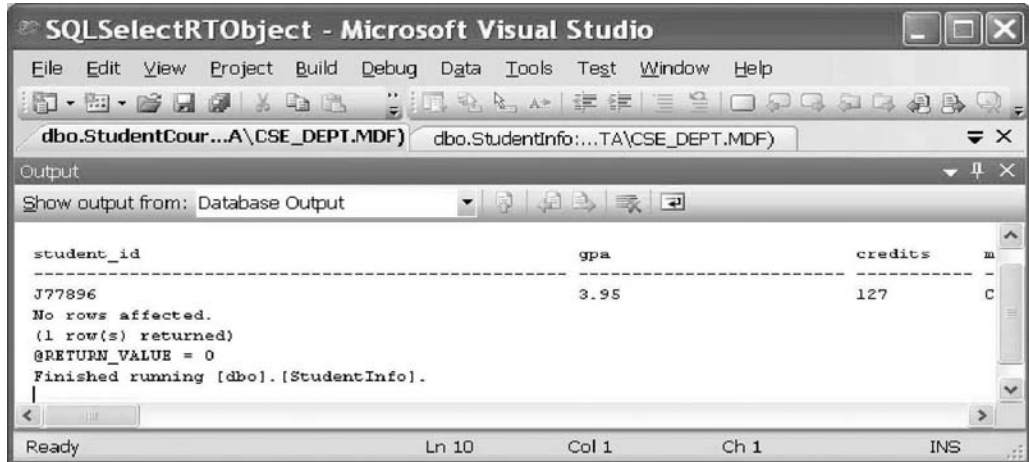


Figure 5.128 Testing result of our first stored procedure.

5. **Copy** Copy the stored procedure.
6. **Delete** Remove the whole stored procedure.
7. **Refresh** Update the content of the stored procedure.
8. **Properties** All properties of the stored procedure are listed.

Now let's run and test our two developed stored procedures. Right-click our first stored procedure `StudentInfo` and select `EXECUTE` item from the pop-up menu. A Run dialog box is displayed to allow you to enter any input parameter you have, which is shown in Figure 5.127b. Enter one of sample students' names, for example, Erica Johnson, into the Value box, and then click on the OK button to run our stored procedure. The testing result is displayed in the Output dialog box shown in Figure 5.128.

In total, there is one row with six columns returned: `student_id`, `gpa`, `credits`, `major`, `schoolYear`, and `email`. Click on the right-arrow bar to scroll right to view all columns.

In a similar way, you can try to run our second stored procedure. You need to enter the `student_id` as the input parameter to run it. Of course, you can use the `student_id` we obtained from our first stored procedure, which is `J77896`. The testing result for our second stored procedure `dbo.StudentCourseInfo` is shown in Figure 5.129.

Now that our two stored procedures have been tested successfully, it is time for us to develop our codes in Visual C# 2008 to call these two stored procedures. Open the Select button Click method by double-clicking on the Select button from the `StudentForm` window, and enter the codes shown in Figure 5.130 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. Two stored procedures' names are assigned to two string variables `strStudent` and `strStudentCourse`, respectively. The names must be identical to those names we created in two stored procedures: `dbo.StudentInfo` and `dbo.StudentCourseInfo`.
- B. All used data components are declared and created in the following section, which include two `TableAdapters`, two `DataTables`, two `Command` objects, and a local string variable `strName`.

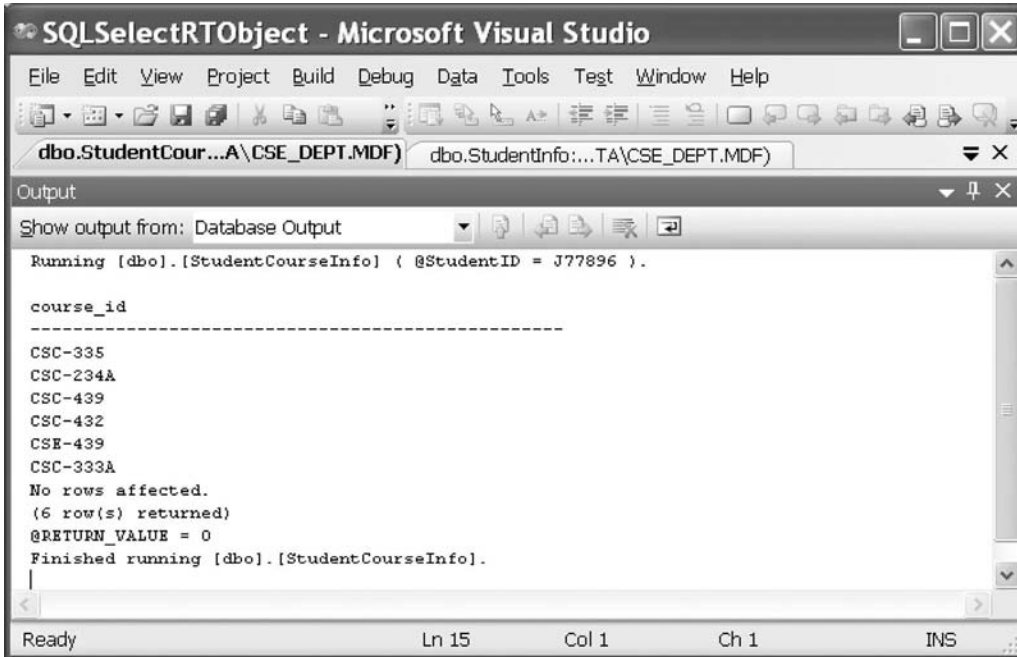


Figure 5.129 Testing result for our second stored procedure.

- C. The FindName() function is executed to get the student's photo file based on the student's name. The returned student's image file is assigned to the local string variable strName.
- D. The user-defined method BuildCommand() is called to initialize the first Command object with the correct Connection, CommandType, and CommandText properties. In order to execute our stored procedure, the properties should be set as follows:
 - CommandType = CommandType.StoredProcedure
 - CommandText = "dbo.StudentInfo"

The content of the CommandText must be equal to the name of the stored procedure we developed above.

- E. The unique input parameter to the stored procedure dbo.StudentInfo is StudentName, which will be selected by the user from the student name combobox (ComboName.Text) as the project runs. This dynamic parameter must be added into the Parameters collection



One point you need to note is that if you are using SQL Server Management Studio Express to build your database, in some situations, you cannot connect the server to open the database if you are performing some tasks with the Server Explorer such as creating stored procedures because your server has been connected and the database is open when you create stored procedures. An error message would be displayed if you try to do that since this version only allows one instance of the server to be connected at a time. You have to disconnect that connection first by rebooting your computer.

```

SQLSelectRTOject.StudentForm cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
A   string strStudent = "dbo.StudentInfo";
B   string strStudentCourse = "dbo.StudentCourseInfo";
   SqlDataAdapter StudentDataAdapter = new SqlDataAdapter();
   SqlDataAdapter StudentCourseDataAdapter = new SqlDataAdapter();
   SqlCommand sqlCmdStudent = new SqlCommand();
   SqlCommand sqlCmdStudentCourse = new SqlCommand();
   DataTable sqlStudentTable = new DataTable();
   DataTable sqlStudentCourseTable = new DataTable();
   SqlDataReader sqlStudentReader, sqlStudentCourseReader;
   string strName = string.Empty;
C   strName = FindName(ComboName.Text);
   if (strName == "No Match")
       MessageBox.Show("No Matched Student's Image Found!");
D   BuildCommand(ref sqlCmdStudent, strStudent);
E   sqlCmdStudent.Parameters.Add("@StudentName", SqlDbType.Char).Value = ComboName.Text;
   StudentDataAdapter.SelectCommand = sqlCmdStudent;
F   if (ComboMethod.Text == "DataAdapter Method")
       {
G       StudentDataAdapter.Fill(sqlStudentTable);
       if (sqlStudentTable.Rows.Count > 0)
           FillStudentTextBox(sqlStudentTable);
       else
           MessageBox.Show("No matched student found!");
H       BuildCommand(ref sqlCmdStudentCourse, strStudentCourse);
I       sqlCmdStudentCourse.Parameters.Add("@StudentID", SqlDbType.Char).Value = txtID.Text;
       StudentCourseDataAdapter.SelectCommand = sqlCmdStudentCourse;
J       StudentCourseDataAdapter.Fill(sqlStudentCourseTable);
K       if (sqlStudentCourseTable.Rows.Count > 0)
           FillCourseList(sqlStudentCourseTable);
       else
           MessageBox.Show("No matched course_id found!");
       }
   else //DataReader Method is selected
       {
L       sqlStudentReader = sqlCmdStudent.ExecuteReader();
M       if (sqlStudentReader.HasRows == true)
           FillStudentReader(sqlStudentReader);
N       else
           MessageBox.Show("No matched student found!");
O       BuildCommand(ref sqlCmdStudentCourse, strStudentCourse);
       sqlCmdStudentCourse.Parameters.Add("@StudentID", SqlDbType.Char).Value = txtID.Text;
       StudentCourseDataAdapter.SelectCommand = sqlCmdStudentCourse;
P       sqlStudentCourseReader = sqlCmdStudentCourse.ExecuteReader();
Q       if (sqlStudentCourseReader.HasRows == true)
           FillCourseReader(sqlStudentCourseReader);
       else
           MessageBox.Show("No matched course_id found!");
R       sqlStudentReader.Close();
       sqlStudentCourseReader.Close();
       }
S   sqlStudentTable.Dispose();
   sqlStudentCourseTable.Dispose();
   StudentDataAdapter.Dispose();
   StudentCourseDataAdapter.Dispose();
   sqlCmdStudent.Dispose();
   sqlCmdStudentCourse.Dispose();
}

```

Figure 5.130 Coding for the Select button Click method.

that is the property of the Command class by using the Add() method before the stored procedure can be executed. The initialized Command object sqlCmdStudent is assigned to the SelectCommand property of the DataAdapter to make it ready to be used in the next step.

- F. If the user selected the DataAdapter Method, the Fill() method of the DataAdapter is called to fill the Student table, which actually calls our first stored procedure to fill the Student table.
- G. If this calling is successful, the Count property should be greater than 0, which means that at least one row is filled into the Student table, and the user-defined method FillStudentTextBox() is called to fill six textboxes in the Student form with six pieces of retrieved columns from the stored procedure. Otherwise, an error message is displayed if this fill has failed.
- H. The user-defined method BuildCommand() is called again to initialize our second Command object sqlCmdStudentCourse. The values to be assigned to the properties of the Command object are:

- CommandType = CommandType.StoredProcedure
- CommandText = "dbo.StudentCourseInfo"

The content of the CommandText must be equal to the name of the stored procedure we developed above.

- I. The unique input parameter to the stored procedure dbo.StudentCourseInfo is the StudentID, which is obtained from the calling of the first stored procedure and stored in the student ID textbox txtID. This dynamic parameter must be added into the Parameters collection that is the property of the Command class by using the Add() method before the stored procedure can be executed. The initialized Command object sqlCmdStudentCourse is assigned to the SelectCommand property of the another DataAdapter to make it ready to be used in the next step.
- J. The Fill() method of the DataAdapter is called to fill the StudentCourse table, which calls our second stored procedure to fill the StudentCourse table.
- K. If this calling is successful, the Count property should be greater than 0, which means that at least one row is filled into the StudentCourse table, and the user-defined method FillCourseList() is called to fill the CourseList box in the Student form with all courses (course_id) retrieved from the stored procedure. Otherwise, an error message is displayed if this fill has failed.
- L. If the user selects the DataReader method, the ExecuteReader() method is called to retrieve back all information related to the selected student and assign it to the sqlStudentReader object.
- M. If this method is executed successfully, which means that at least one row of data is read out and assigned to the DataReader, the HasRows property should be true, and the user-defined method FillStudentReader() is executed to fill the related information to six textboxes in the Student form.
- N. Otherwise, an error message is displayed to indicate this situation.
- O. The BuildCommand() method is called to build the StudentCourse Command object with our second stored procedure. The unique input parameter to this stored procedure is the StudentID that is added to the Parameters collection that is a property of the Command object. The finished Command object is assigned to the SelectCommand property of the StudentCourse DataAdapter.

- P. The `ExecuteReader()` method is called to run our second stored procedure to read out all courses taken by the selected student, and assign them to the `StudentCourse DataReader` object.
- Q. If the method `ExecuteReader()` runs successfully, the `HasRows` property of the `DataReader` object should be true, the user-defined method `FillCourseReader()` is executed to fill all courses (`course_id`) into the `CourseList` listbox. Otherwise if this method has failed, an error message is displayed.
- R. Two student `DataReader` objects are released before we can exit this method.
- S. The cleaning job is performed to release all other data objects used in this method.

The code for the method `FindName()` is identical to what we developed for the same method in Section 5.18.5.2. Refer to Figure 5.99 to get detailed information for this coding.

In order to pick up the correct student's image file from this subroutine, note that you must store all students' image files in the folder in which your Visual C# 2008 project's executable file is located. In our application, this folder is `C:\Book6\Chapter 5\SQLSelectRTOject\bin\Debug`. If you place those students' image files in another folder, you must provide a full name, which includes the drive name, path, and the image file name, for that student's image file to be accessed, and assign it to the returning string variable `strName` in this method. The coding for the `BuildCommand()` method is shown in Figure 5.131.

This coding is straightforward and easy to be understood. First, a new `LogInForm` instance is created, and the `getLogInForm()` method is called to pick up the global connection object and assign it to the `Connection` property of the `Command` object. Then the `Command` object is initialized by assigning the related properties such as the `CommandType` and `CommandText` to it. The point is that the value assigned to the `CommandType` must be `StoredProcedure`, and the value assigned to the `CommandText` must be equal to the name of the stored procedure we developed in the last section.

The codes for the methods `FillStudentTextBox()`, `MapStudentTextBox()`, and `FillCourseList()` are shown in Figure 5.132. In total we need six pieces of information for the selected student, so a six-dimension textbox array `StudentTextBox[5]` is used (the index is based on 0).

In the method `FillStudentTextBox()`, a double foreach loop is used to retrieve each queried column from the returned row. The outer loop is only executed one time since we have only retrieved back one data row from the `Student` table. Each fetched column is assigned to the associated textbox to be displayed in the `Student` form. The method

```

SQLSelectRTOject.StudentForm BuildCommand()
private void BuildCommand(ref SqlCommand cmdObj, string cmdString)
{
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    cmdObj.Connection = logForm.sqlConnection;
    cmdObj.CommandType = CommandType.StoredProcedure;
    cmdObj.CommandText = cmdString;
}

```

Figure 5.131 Coding for the `BuildCommand` method.


```

SQLSelectRTObject.StudentForm FillStudentTextBox()
private void FillStudentTextBox(DataTable StudentTable)
{
    int pos1 =0;
    for (int pos2 = 0; pos2 <= 6; pos2++) //Initialize the textbox array
        StudentTextBox[pos2] = new TextBox();
    MapStudentTextBox(StudentTextBox);
    foreach (DataRow row in StudentTable.Rows)
    {
        foreach (DataColumn column in StudentTable.Columns)
        {
            StudentTextBox[pos1].Text = row[column].ToString();
            pos1++;
        }
    }
}
private void MapStudentTextBox(Object[] sTextBox)
{
    sTextBox[0] = txtID; //The order must be identical with the
    sTextBox[1] = txtGPA; //order in the query string - strStudent
    sTextBox[2] = txtCredits;
    sTextBox[3] = txtMajor;
    sTextBox[4] = txtSchoolYear;
    sTextBox[5] = txtEmail;
}
private void FillCourseList(DataTable StudentCourseTable)
{
    CourseList.Items.Clear();
    foreach (DataRow row in StudentCourseTable.Rows)
    {
        CourseList.Items.Add(row[0]); //the 1st column is course_id - strStudentCourse
    }
}
}

```

Figure 5.132 Coding for three user-defined methods.

MapStudentTextBox() is to set up a one-to-one correct relationship between each element in the textbox array and each associated textbox control in the Student form. The functionality of the method FillCourseList() is to pick up each matched row (course_id) from the filled StudentCourse table and add them into the CourseList listbox. The codes for the methods FillStudentReader() and FillCourseReader() are shown in Figure 5.133.

In the method FillStudentReader(), first textbox object array is initialized, and then the MapStudentTextBox() method is executed to set up a correct relationship between each element in the textbox array and each associated textbox control in the Student form. A while and a for loop are utilized to pick up each fetched column and assign them to the associated textbox control. The point is the difference between the system method GetString() and GetValue() that belong to the DataReader class. The former can be used to get only string data stored in the database, but the latter can be used to get any kind of data stored in the database. The issue is that another system method, ToString(), must be attached to this GetValue() to convert the data from any other data type to the string and assign it to the Text property of each TextBox control. Therefore, the latter is more popular and more powerful in a data-driven application. You can try to compare these two methods yourself if you like.

```

SQLSelectRTOject.StudentForm FillStudentReader()
private void FillStudentReader(SqlDataReader StudentReader)
{
    int intIndex = 0;
    for (int pos2 = 0; pos2 <= 6; pos2++) //Initialize the textbox array
        StudentTextBox[pos2] = new TextBox();
    MapStudentTextBox(StudentTextBox);
    while (StudentReader.Read())
    {
        for (intIndex = 0; intIndex <= StudentReader.FieldCount - 1; intIndex++)
            StudentTextBox[intIndex].Text = StudentReader.GetValue(intIndex).ToString();
    }
}
private void FillCourseReader(SqlDataReader StudentCourseReader)
{
    int pos = 0;
    CourseList.Items.Clear();
    while (StudentCourseReader.Read())
    {
        for (pos = 0; pos <= StudentCourseReader.FieldCount - 1; pos++)
            CourseList.Items.Add(StudentCourseReader.GetValue(pos).ToString());
    }
}

```

Figure 5.133 Coding for another two user-defined methods.

The functionality of the method `FillCourseReader()` is similar to the method `FillStudentReader()`. The only difference between them is that the `Add()` method is used to add each fetched row (`course_id`) to the listbox in the `FillCourseReader()` method.

We almost completed all coding development for this project. Finally don't forget to make coding for the Back command button. Enter `this.Hide()` into this method.

Now we can begin to test calling those two stored procedures from our Visual C# project. Build the project and click on the Start Debugging button to run our project, enter user name and password, and select the Student Information item to open the Student form window, which is shown in Figure 5.134.

Select Ashly Jade from the Student Name combobox and click on the Select button. All information related to this student and the courses are displayed in the six textboxes and the CourseList box, which is shown in Figure 5.134.

Our calling to two stored procedures are very successful! Some readers may find that these two stored procedures are relatively simple, and each procedure only contains one SQL statement. Let's dig a little deeper and develop some sophisticated stored procedures and try to call them from our Visual C# project. Next we will develop a stored procedure that contains more SQL statements.

5.19.2.7.4 Query Data Using More Complicated Stored Procedures We want to get all courses (all `course_ids`) taken by the selected student based on student name from the StudentCourse table. To do that, we must first go to the Student table to obtain the associated `student_id` based on the student name since there is no student name column available in the StudentCourse table. Then we can go to the StudentCourse table to pick up all `course_id` based on the selected `student_id`. We need to perform two queries to complete this data-retrieving operation. Now we try to combine these two queries into a

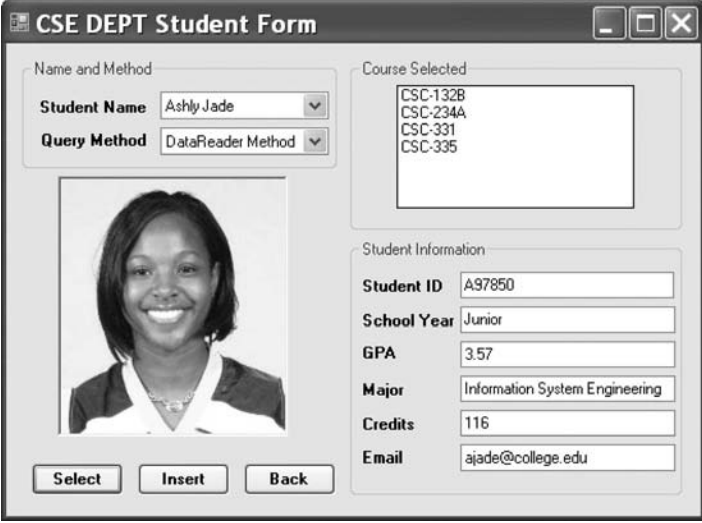
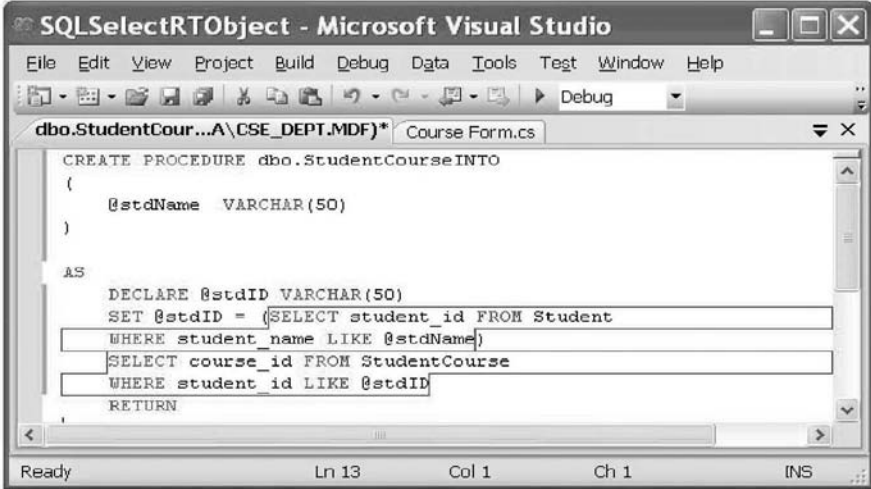


Figure 5.134 Running status of the Student form.



```

A CREATE PROCEDURE dbo.StudentCourseINTO
B {
C     @stdName VARCHAR(50)
D }
E AS
F DECLARE @stdID VARCHAR(50)
SET @stdID = (SELECT student_id FROM Student
WHERE student_name LIKE @stdName)
SELECT course_id FROM StudentCourse
WHERE student_id LIKE @stdID
RETURN

```

Figure 5.135 New stored procedure: StudentCourseINTO.

single stored procedure to simplify our data-querying operation. First let's create our stored procedure.

Open Visual Studio.NET and open the Server Explorer window; click on the plus symbol icon that is next to the CSE_DEPT database folder to connect to our database if this database was added into the Server Explorer before. Otherwise you need to right-click on the Data Connections folder to add and connect to our database.

Right-click on the Stored Procedures folder and select the Add New Stored Procedure item to open the Add Procedure dialog box, and then enter the codes shown in Figure 5.135 into this new procedure.

Let's give a detailed discussion for this piece of coding.

- A. The stored procedure is named `dbo.StudentCourseINTO`.
- B. The input parameter is the student name, `@stdName`, which is a varying-char variable with the maximum characters of 50. All parameters, no matter input or output, must be declared inside the braces.
- C. The local variable `@stdID` is used to hold the returned query result from the first SQL statement that retrieves the `student_id`.
- D. The first SQL statement is executed to get the `student_id` from the `Student` table based on the input parameter `@stdName`. A `SET` command must be used to assign the returned result from the first SQL query to the local variable (or intermediate variable) `@stdID`. The first SQL statement must be covered by the parenthesis to indicate that this whole query will be returned as a single data.
- E. The second SQL statement is executed, and this query is used to retrieve all courses (`course_id`) taken by the selected student from the `StudentCourse` table based on the `student_id` (`@stdID`) obtained from the first query.
- F. Finally the queried result, all courses or `course_id`, is returned.

Go to File|Save StoredProcedure1 to save this stored procedure.

Now let's test our stored procedure in the Server Explorer window. Right-click on our new created stored procedure `StudentCourseINTO` and select the `Execute` item from the pop-up menu. On the opened dialog box, enter the student's name, Erica Johnson; then click on the `OK` button to run the procedure. The running result is shown in Figure 5.136.

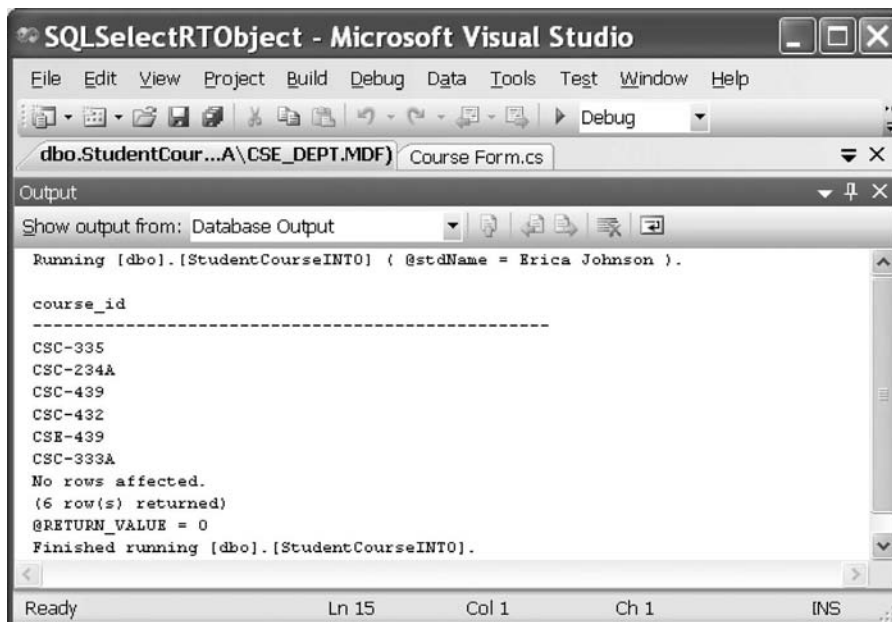


Figure 5.136 Running result of the stored procedure.

Next we need to develop a Visual C# project to call this stored procedure to test the functionality of the stored procedure. To save time and space, we added a new form window into this project and named it SPForm.

Open our project SQLSelectRTOject and select Project | Add Windows Form item, enter SP Form.cs into the Name: box and click on the Add button to add this new form into our project. Enter “SPForm” into the name property as the name for this form.

Enlarge the size of this SP Form by dragging the border of the form window, and then open the Student form window. We need to copy all controls on the Student form to our new SP form. On the opened Student form, select Edit|Select All and Edit|Copy items, and then open the SP form and select Edit|Paste to paste all controls we copied from the Student form.

To save time, next we need to copy most codes from the Student code window to our new SP form code window. But first you need to add the SQL Server-related namespace System.Data.SqlClient to the namespace declaration section on this SP form code window. The only difference is the codes for the Select button Click method, cmdSelect_Click(). Don't copy this piece of code since we need to develop new codes to test our stored procedure later. To copy all other codes, open the code window of the Student form, select those codes, copy, and then paste them to our new SP form code window. One point you need to note is that when you copy the codes for two methods, such as the constructor of the SP Form and the Back button Click, you must first open the constructor or the Back method in the SP form, and only copy the body of those methods without including the header and ender of those methods.

Now let's develop the codes for our Select button Click method. Most codes are identical to those we developed for the Student form, such as the methods FindName() and FillCourseReader(). Open the Select button Click method by double-clicking on the Select button from the Designer window, and enter the code shown in Figure 5.137 into this method. Let's discuss this piece of code step by step to see how it works.

- A. The name of our stored procedure, “dbo.StudentCourseINTO”, must be declared first, and this name must be identical with the name we used when we created the stored procedure in the Server Explorer window.
- B. All data components, including the Command, DataAdapter, DataTable, and DataReader objects, are declared here since we need to use them for this data query operation.
- C. The method FindName() is called to get and display the matched student image file, and the returned name of the image file is stored in the local string variable strName.
- D. The Command object is initialized with suitable properties by executing the user-defined method BuildCommand(). The CommandType property must be StoredProcedure to indicate that this query is to execute a stored procedure. The CommandText property must be equal to the name of our stored procedure, “dbo.StudentCourseINTO”, which is stored in a string variable strStudentCourse.
- E. The input parameter to the stored procedure is the student name, which is obtained from the student combobox, and it should be added into the Parameters collection property of the Command object. You need to note that the nominal name @stdName must be identical with the input parameter name we defined in the parameter braces in our stored procedure dbo.StudentCourseINTO. The real parameter is entered by the user as the project runs. The finished Command object is assigned to the SelectCommand property of the DataAdapter, which will be used later to fetch the desired course_id from the StudentCourse table.

```

SQLSelectRTOBJect.SPForm cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
    A   string strStudentCourse = "dbo.StudentCourseINTO";
    B   SqlDataAdapter StudentCourseDataAdapter = new SqlDataAdapter();
       SqlCommand sqlCmdStudentCourse = new SqlCommand();
       DataTable sqlStudentCourseTable = new DataTable();
       SqlDataReader sqlStudentCourseReader;
       string strName = string.Empty;

    C   strName = FindName(ComboName.Text);
       if (strName == "No Match")
           MessageBox.Show("No Matched Student's Image Found!");
    D   BuildCommand(ref sqlCmdStudentCourse, strStudentCourse);
    E   sqlCmdStudentCourse.Parameters.Add("@stdName", SqlDbType.Char).Value = ComboName.Text;
       StudentCourseDataAdapter.SelectCommand = sqlCmdStudentCourse;
       if (ComboMethod.Text == "DataAdapter Method")
       {
    F       StudentCourseDataAdapter.Fill(sqlStudentCourseTable);
    G       if (sqlStudentCourseTable.Rows.Count > 0)
           FillCourseList(sqlStudentCourseTable);
           else
           {
               MessageBox.Show("No matched course_id found!");
           }
       }
       else //DataReader Method is selected
       {
    H       sqlStudentCourseReader = sqlCmdStudentCourse.ExecuteReader();
    I       if (sqlStudentCourseReader.HasRows == true)
           FillCourseReader(sqlStudentCourseReader);
           else
           {
               MessageBox.Show("No matched course_id found!");
           }
       }
    J   sqlStudentCourseTable.Dispose();
       StudentCourseDataAdapter.Dispose();
       sqlCmdStudentCourse.Dispose();
    }
}

```

Figure 5.137 Coding for the Select button Click method.

- F.** If the user selected the DataAdapter Method, the Fill() method is called to fill the StudentCourse table with the desired course_id.
- G.** If the Count property is greater than 0, this fill is successful. The user-defined method FillCourseList() is called to fill the fetched course_id into the CourseList listbox. Otherwise an error message is displayed to indicate this situation.
- H.** If the user selected the DataReader Method, the ExecuteReader() method is executed to invoke the DataReader to call our stored procedure.
- I.** If this call is successful, the queried result should be stored in the DataReader with certain rows. The user-defined method FillCourseReader() is executed to fill the returned course_id into the CourseList box in that situation. Otherwise an error message is displayed if this call is failed.
- J.** The cleaning job is performed to release all objects used in this data query operation.

Most user-defined methods, such as BuildCommand() and FindName(), are identical with those methods in the Student form without any modification. For your convenience, we show the other two user-defined methods FillCourseList() and FillCourseReader() again in Figure 5.138.

SQLSelectRTOject.SPForm	▼	FillCourseList()	▼
-------------------------	---	------------------	---

```

private void FillCourseList(DataTable StudentCourseTable)
{
    CourseList.Items.Clear();
    foreach (DataRow row in StudentCourseTable.Rows)
    {
        CourseList.Items.Add(row[0]);    //the 1st column is course_id - strStudentCourse
    }
}
private void FillCourseReader(SqlDataReader StudentCourseReader)
{
    int pos = 0;
    CourseList.Items.Clear();
    while (StudentCourseReader.Read())
    {
        for (pos = 0; pos <= StudentCourseReader.FieldCount - 1; pos++)
            CourseList.Items.Add(StudentCourseReader.GetValue(pos).ToString());
    }
}

```

Figure 5.138 Coding for two user-defined methods.

Before you can test this method, you need to perform the following operations to the Selection form to allow users to select this form. Open the code window of the Selection form and add the following codes:

1. Create a new field-level instance of the SP Form by entering **SPForm spForm = new SPForm();** inside the SPForm class.
2. Add a new item SP Information into the ComboSelection combobox by entering **this.ComboSelection.Items.Add("SP Information");** into the constructor of this form.
3. Add a selection branch for the SP Information by entering the following code into the OK button Click method:

```

else if (this.ComboSelection.Text == "SP Information")
    spForm.Show();

```

4. Add **spForm.Close()** into the Exit button Click method.

Now you can run the project to test this stored procedure. One important issue is that you may encounter a database connection error when you run this project. The reason for this error is that the SQL Server 2005 Express only allows the user to create a single instance of the database and allows single connection to this database. In other words, multiple accessing of this database at the same time is prohibited. Therefore you need to disconnect the connection you set up before using the Server Explorer by rebooting the computer, and then reopen the project to run it. A possible solution to this problem is that you can copy the completed database file CSE_DEPT.mdf into another folder in your computer; in our case, it is C:\database folder, and connect your project to this database file using the Server Explorer. You can use the same database file CSE_DEPT.mdf that is located at the default SQL Server 2005 database folder, C:\Program Files\Microsoft SQL Server 2005\MSSQL.1\MSSQL\Data, as your target database and connect it to your project using the codes. The prerequisite to use this possible solution is that two database files must be identical, and both files are final versions without any further modification.

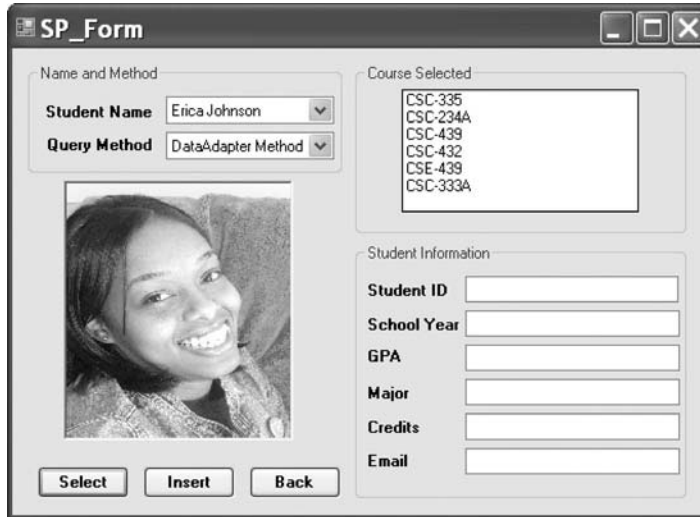


Figure 5.139 Running status of calling stored procedure.

As the project runs, enter the suitable username and password, and then select the SP Information from the Selection form to open the SP Form window. Select a student name from the student combobox and click on the Select button. All courses taken by the selected student will be displayed in the CourseList box, which is shown in Figure 5.139.

The testing result for our stored procedure is very good. Is that correct? The answer is yes, but we may do a little more to develop a more sophisticated stored procedure to meet our special requirement. What is our special requirement? We want to develop some nested stored procedures and call the nested stored procedure from our main or parent stored procedure. Sounds complicated? Yes, but we try to make this complicated issue simple with the following example.

5.19.2.7.5 Query Data Using Nested Stored Procedures The so-called nested stored procedure is very similar to subroutines or subqueries, which means that a main stored procedure can be considered as a parent stored procedure, and a substored procedure can be considered as a child stored procedure. The parent stored procedure can call the child stored procedure as it likes. In this section, we try to create two stored procedures; one is a main procedure and the other is a child stored procedure. The main stored procedure, StudentAndCourse, is used to get all courses taken by the selected student from the StudentCourse table based on the student_id, and the child stored procedure, StudentInfoID, is used to get the student_id from the Student table based on the input student name.

Now open the Server Explorer window and connect to our database CSE_DEPT. After the database is connected, right-click on the Stored Procedures folder and select the Add New Stored Procedure item to open the Add Procedure dialog box.

First, let's create our main stored procedure—dbo.StudentAndCourse. Enter the codes shown in Figure 5.140 into our main stored procedure.

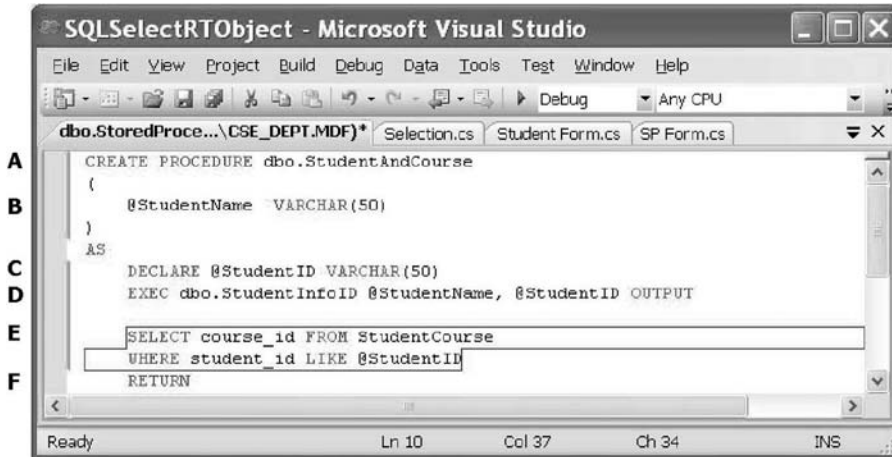


Figure 5.140 Main stored procedure.

The functionality of each line of the coding is:

- A. The name of the main stored procedure is declared first, which is `dbo.StudentAndCourse`. This name must be identical with the name of the stored procedure used in our Visual C# 2008 project later.
- B. The input parameter `@StudentName` is declared here.
- C. The local variable `@StudentID` is used as an output parameter for the child stored procedure that will return this parameter, `student_id`, to our main procedure.
- D. Call the child stored procedure to execute it using the command `EXEC`. This calling passes two parameters to the child stored procedure: the input parameter to the child procedure `@StudentName` and the output parameter `@StudentID`. The latter must be indicated with the keyword `OUTPUT`. Later in the child stored procedure, you must also declare this parameter as an output parameter using the keyword `OUTPUT` to match its definition defined in this main stored procedure.
- E. After the child stored procedure is executed, it returns the `student_id`. Now we can perform our main query to obtain all courses taken by the selected student from the `StudentCourse` table based on the `student_id` returned by the child stored procedure.
- F. The retrieved courses are returned to the calling procedure developed in Visual C# 2008.

Click on the File | Save StoredProcedure1 item to save the main stored procedure.

Second, let's create our child stored procedure. Right-click on the Stored Procedures folder and select the Add New Stored Procedure item from the pop-up menu. On the opened dialog box, enter the codes shown in Figure 5.141 into this new stored procedure.

The functionality for each coding line is explained below:

- A. The name of the child stored procedure is declared first, which is `dbo.StudentInfoID`. This name must be identical with the name used by our main stored procedure when this child procedure is called.
- B. Two parameters are declared here: The first one, `@sName`, is the input, and the second, `@sID`, is the output parameter. The default type of parameter is `INPUT`, so the keyword `OUTPUT` must be attached for the second parameter since it is an output and will be returned to the main stored procedure.

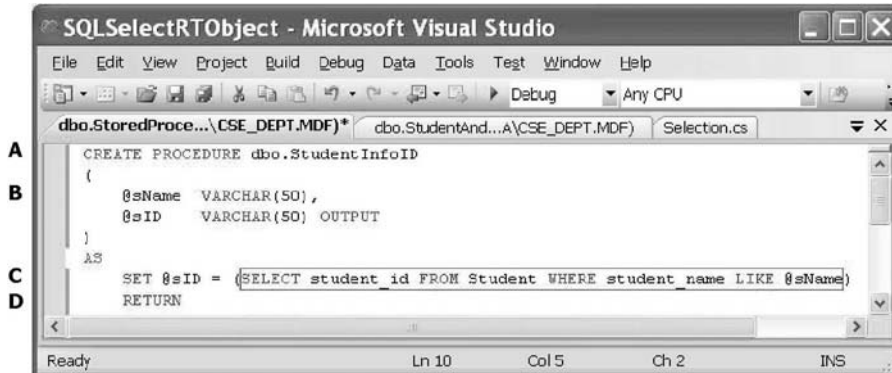


Figure 5.141 Child stored procedure.

- C. The SQL statement is executed to get the desired `student_id` from the `Student` table based on the input student name. The returned `student_id` will be assigned to the output parameter `@sID` by using the `SET` command.

- D. The output parameter `@sID` is returned to the main stored procedure.

Click on the `File|Save StoredProcedure1` item to save our child stored procedure.

To test both main and child stored procedure in the Server Explorer window, right-click on the main stored procedure `StudentAndCourse` item, and then select `Execute` item to open the `Run Stored Procedure` dialog.

Enter a student name, such as `Erica Johnson`, and click on the `OK` button to run both stored procedures. The running result is shown in Figure 5.142. Our nested stored procedures work fine!

To call this nested stored procedure, we need to develop a Visual C# 2008 project. In order to save time and space, you can use the coding we developed for the `SPForm` window in the last section. All coding is the same and the only modifications are the stored procedure's name declared in the `cmdSelect_Click()` method and the nominal input parameter name to the stored procedure. Change the name of the stored procedure from "`dbo.StudentCourseINTO`" to "`dbo.StudentAndCourse`" (refer to line **A** in Figure 5.137), and change the nominal parameter's name from "`@stdName`" to "`@StudentName`" (refer to line **E** in Figure 5.137). Then you can run this project to get the same result, which is shown in Figure 5.139, as we got from the last project.

At this point, we finished developing the data-driven project using the general real-time object for the SQL Server database. A complete project named `SQLSelectRTObject` can be found in the folder `DBProjects|Chapter 5` located at the accompanying ftp site (see Chapter 1).

Now let's go to the next part in this chapter—develop a data-driven application using the `LINQ to SQL` with the SQL Server database.

5.19.3 Query Data Using LINQ to SQL Technique

As we discussed in Chapter 4, `LINQ to SQL` is an application programming interface (API) that allows users to easily and conveniently access the SQL Server database from the Standard Query Operators (SQO) related to the `LINQ`. To use this API, you must

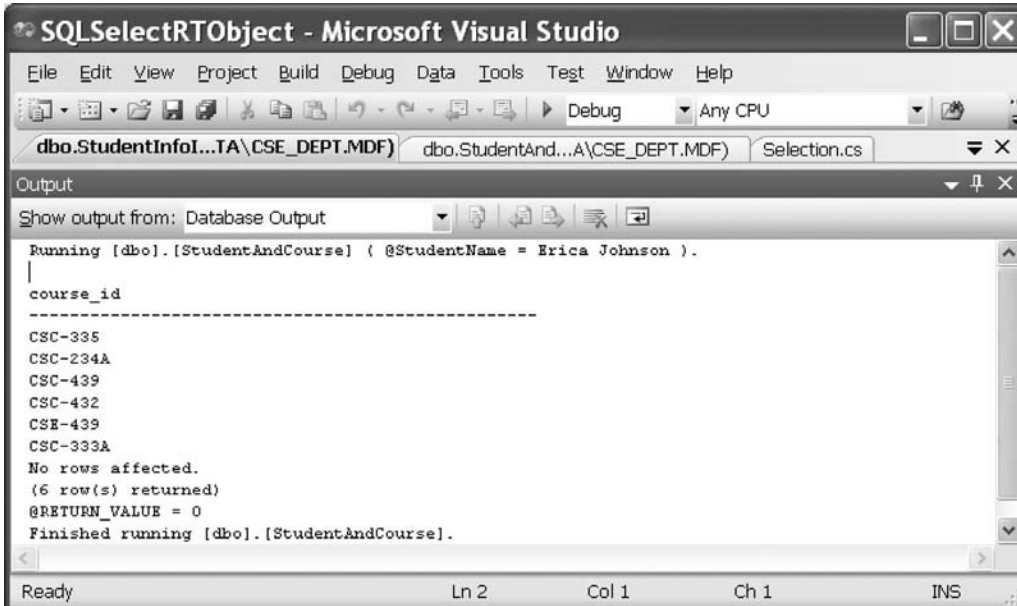


Figure 5.142 Running result of the nested stored procedure.

first set up a mapping relationship between your relational database and a group of objects that are instantiated from entity classes. The LINQ to SQL or the Standard Query Operators will interface to these entity classes to perform the real database operations. In other words, each entity class can be mapped or is equivalent to a physical data table in the database, and each entity class's property can be mapped or is equivalent to a data column in that table. Once this mapping relationship has been set up, one can use the LINQ to SQL to access and manipulate data against the databases.

After entity classes are created and the mapping relationships between each physical table and each entity class has been built, the conversion for data operations between the entity class and the real data table is needed. The class `DataContext` will function in this role. Basically, the `DataContext` is a connection class that is used to establish a connection between your project and your database. In addition to this connection role, the `DataContext` also provides the conversion function to convert or interpret operations of the Standard Query Operators for the entity classes to the SQL statements that can be run in real databases.

The procedure to use LINQ to SQL to perform data actions against SQL Server database can be described in the sequence listed below:

1. Add the `System.Data.Linq.dll` assembly into the project that will use LINQ to SQL by adding the reference `System.Data.Linq`.
2. Create an entity class for each data table by using one of two popular tools: `SQLMetal` or `Object Relational Designer`.
3. Add a connection to the selected database using the `DataContext` class or the derived class from the `DataContext` class.
4. Use LINQ to SQL to access the database to perform desired data actions.

The difference between the SQLMetal and the Object Relational Designer is that the former is a console-based application but the latter is a window-based application. This means that the SQLMetal provides a DOS-like template, and the operations are performed by entering single commands into a black-white window. The Object Relational Designer provides a graphic user interface (GUI) and allows users to drag-place tables represented by graphic icons into the GUI. Obviously, the second method or tool is more convenient and easier compared with the first one.

In this section, we will develop a new sample Visual C# 2008 project named SQLSelectRTOBJECTLINQ and use it to illustrate how to perform the data query using the LINQ to SQL method step by step. Now let's create a new Visual C# project in our default folder C:\Book6\Chapter 5. For your convenience, a blank project with five form windows has been created, and you can directly use this blank project to develop your codes to perform the LINQ to SQL data query. This project, SQLSelectRTOBJECTLINQ, is located at the folder DBProjects\Chapter 5 at the accompanying ftp site (see Chapter 1), and you can copy and paste it into your folder to use it.

5.19.3.1 Create Entity Classes and Connect DataContext to Database

We have provided a very detailed discussion about the entity classes and DataContext object as well as how to use the Object Relational Designer to create and add these components to a data-driven project to perform LINQ to SQL queries in Section 4.6.1. Before we can continue to go to the next step, refer to that section to get a clear picture of how to create and use these components.

5.19.3.2 Query Data Using LINQ to SQL for LogIn Form

When we finished reviewing for Section 4.6.1 in Chapter 4, we can continue to complete our sample project. First, we need to create a field variable `cse_dept` based on our derived DataContext class `CSE_DEPTDataContext`. As we discussed in the last section, this object is used to connect to our sample database. Four overloaded constructors are available for this DataContext class, but in this application we will use the simplest one to simplify our coding process. Open the code window of the `LogInForm` and enter the codes shown in Figure 5.143 into this code section.

Let's look at this piece of code to see how it works.

- A.** The namespace `System.Data.Linq` is added into this code section since we need to use all data components related to LINQ to SQL, and this namespace contains all of them.
- B.** A new instance of our derived class `CSE_DEPTDataContext` is created with the first constructor. A trick here is that no connection string is included in this connection object. Yes, but where is the connection string? How can we connect to our database without connection string? However, the connection string is in there. Where? Let's see.

Open the application configuration file named `app.config` located at the Solution Explorer window by double-clicking on it. A sample file is shown in Figure 5.144.

It can be found from this XML file that the connection string is already in there under the `ConnectionString` tag, which includes the location of the sample database file `CSE_DEPT.mdf`. In this application, it is the default folder of the SQL Server 2005 database file, `C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data`.

```

SQLSelectRTOBJECTLINQ.LogInForm   LogInForm()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
A using System.Data.Linq;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SQLSelectRTOBJECTLINQ
{
    public partial class LogInForm : Form
    B     {
        public CSE_DEPTDataContext cse_dept = new CSE_DEPTDataContext();

        public LogInForm()
        {
            InitializeComponent();
        }
    }
}

```

Figure 5.143 Coding for the creation of the field variable.

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="MSSQLSelectRTOBJECTLINQ.Properties.Settings.CSE_DEPTConnectionString"
        connectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=&quot;C:\Program Files
          \Microsoft SQL Server\MSSQL.1\MSSQL\Data\CSE_DEPT.mdf&quot;;Integrated
          Security=True;Connect Timeout=30;User Instance=False"
        providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>

```

Figure 5.144 Sample coding for the app.config file.

When you create a new instance of the derived DataContext class to connect to the sample database, the system can automatically locate this connection string from this configuration file and use it to do this connection even we did not clearly indicate this connection string. The instance `cse_dept` has finished the connection when you create it, and save this connection in this instance when it is created. Now we can use this connection object to perform the data actions against the database.

Now let's develop the codes for the login process. Since we only deal with this login process using the LINQ to SQL, we only need one LogIn button in this LogInForm window. Delete one button from this form window and rename the left button as `cmdLogIn` and change its Text property to LogIn.

Double-click on the LogIn button from the form window to open its Click method, and enter the codes shown in Figure 5.145 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** Two local-level string variables, `username` and `password`, are created, and these two variables are used to hold the returned queried data from the LogIn table.

```

SQLSelectRTOBJECTLINQ.LogInForm  cmdLogIn_Click()
private void cmdLogIn_Click(object sender, EventArgs e)
{
A   string username = string.Empty;
B   string password = string.Empty;
B   SelectionForm selfForm = new SelectionForm();
C   IQueryable<LogIn> loginfo = from lg in cse_dept.LogIns
                                where lg.user_name == txtUserName.Text &&
                                   lg.pass_word == txtPassWord.Text
                                select lg;
D   foreach (LogIn log in loginfo)
    {
        username = (string)log.user_name;
        password = (string)log.pass_word;
    }
E   if (txtUserName.Text == string.Empty || txtPassWord.Text == string.Empty)
        MessageBox.Show("Enter a valid username/password");
F   else if (username == txtUserName.Text && password == txtPassWord.Text)
    {
        MessageBox.Show("The LogIn is successful!");
        selfForm.Show();
        this.Hide();
    }
G   else
        MessageBox.Show("The LogIn is failed!");
}

```

Figure 5.145 Codes for the LogIn button Click method.

- B.** An instance of the next form class SelectionForm, selfForm, is created and this form will be displayed if the login process is completed successfully.
- C.** As we discussed in the last section when we built the entity classes for this project, all five entity classes are related or bounded by using the associations (primary-foreign keys). Also recall that in Chapter 4 we discussed that most normal LINQ queries are performed on arrays or collections that apply the IEnumerable<T> or IEnumerable interfaces, but in LINQ to SQL queries, they are performed on entity classes that apply the IQueryable<T> interface. This means that besides the Standard Query Operators, LINQ to SQL queries have additional query operators available since IQueryable<T> applies IEnumerable<T>. Here querying an associated entity class in LINQ to SQL is used with the IQueryable<T> interface, and the <T> is the type of member variable of the related entity class. In this case, it is our LogIn table. An iteration variable lg is used to iterate over the result of this query from the LogIn table. Then a similar SQL SELECT statement is executed with the WHERE clause. Two criteria are used for this query, user_name and pass_word, and these two criteria are connected with a logic AND operator. Note the member variable of our entity class LogIn, which is named LogIns in this query. The relationships or the associations between the LogIn, Faculty, and Student tables are many-to-one, which means that many faculty_ids and students_ids can exist in the LogIn table, but only a single or unique faculty_id can be in the Faculty table and a unique student_id in the Student table. In other words, the LogIn class is in the many (child) side of a one-to-many relationship. Therefore, generally, the member variable of this kind of entity class is named LogIns, and an “s” is attached to the name of the related entity class.
- D.** The foreach loop is utilized to pick up each column from the selected data row log, which is obtained from the loginfo we get from the LINQ query. Then, assign two columns, log.user_name and log.pass_word, to our two local string variables, username and password.

The purpose of this assignment operation is to avoid the possible overhead cycles when identifying the validity of the username and password entered by the user. In other words, we prefer to do this validation outside of this foreach loop. You can try to do it inside this loop but definitely you would encounter some bugs. Since we are using a typed data table and database, we can directly access each column by using its name without using the field<string> and the column's name as the position for each of them.

- E. An error message will be displayed if any of input box is empty to remind user to enter valid username and password.
- F. If both username and password are correct and matched to both columns queried from the LogIn table, a successful message is displayed and our next form, SelectionForm, is displayed and the current form is removed from the screen. The point is that this successful message is only for the testing purpose and it should be commented out during the normal running process of this project.
- G. If no matched username and password can be found, an error message is displayed to indicate this situation.

It looks like this query is quite simple, and all columns in the LogIn table have been in there even if we did not explicitly perform any query to that table. The reason behind this simple query is that no login information is actually retrieved until all columns in the LogIn class are referenced, and this is called deferred loading. This terminology is used to describe the type of loading in which columns are not actually loaded from the database until they are required or referenced.

The code for the Cancel button Click method is easy and it is shown in Figure 5.146. The coding for the user-defined method getLogInForm() is also shown in this figure. This is all the coding developments for the LogIn form. Before we can test this coding, we prefer to finish the coding for the next form, SelectionForm.

5.19.3.3 Coding for Selection Form

The coding for this form is basically identical with the same form we did for the last SQLSelectRTOBJECT project, and the only difference is that the database connection object we used in this project is an instance of the derived class CSE_DEPTDataContext. Another point is that we do not need to add the namespace System.Data.Linq to the namespace declaration section in this form since the LINQ to SQL will not be used for this form.

	SQLSelectRTOBJECT.LINQ.LogInForm	cmdCancel_Click()
A B	<pre>private void cmdCancel_Click(object sender, EventArgs e) { cse_dept.Connection.Close(); Application.Exit(); } public LogInForm getLogInForm() { return this; }</pre>	

Figure 5.146 Codes for the Cancel button Click method.


```

SQLSelectRTOBJECTLINQ.SelectionForm SelectionForm()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace SQLSelectRTOBJECTLINQ
{
    public partial class SelectionForm : Form
    {
        FacultyForm facultyForm = new FacultyForm();
        CourseForm courseForm = new CourseForm();
        StudentForm studentForm = new StudentForm();
        public SelectionForm()
        {
            InitializeComponent();
            this.ComboSelection.Items.Add("Faculty Information");
            this.ComboSelection.Items.Add("Course Information");
            this.ComboSelection.Items.Add("Student Information");
            this.ComboSelection.SelectedIndex = 0;
        }
        private void cmdOK_Click(object sender, EventArgs e)
        {
            if (this.ComboSelection.Text == "Faculty Information")
                facultyForm.Show();
            else if (this.ComboSelection.Text == "Course Information")
                courseForm.Show();
            else if (this.ComboSelection.Text == "Student Information")
                studentForm.Show();
            else
                MessageBox.Show("Invalid Selection!");
        }
        private void cmdExit_Click(object sender, EventArgs e)
        {
            LogInForm logForm = new LogInForm();
            logForm = logForm.getLogInForm();
            if (logForm.cse_dept.Connection.State == ConnectionState.Open)
                logForm.cse_dept.Connection.Close();
            logForm.Close();
            courseForm.Close();
            facultyForm.Close();
            studentForm.Close();
            Application.Exit();
        }
    }
}

```

Figure 5.147 Codes for the SelectionForm.

Open the SelectionForm and enter the codes shown in Figure 5.147 into the code window of this form. Most of code is identical with the same form we did for the SQLSelectRTOBJECT project except for the following codes:

- A. When we click on the Exit button, first we need to check whether our connection object is still active or open. If it is open, we need to close this connection before we can exit the project. Otherwise a bug would be created in this project if you exit the project without closing the database connection.

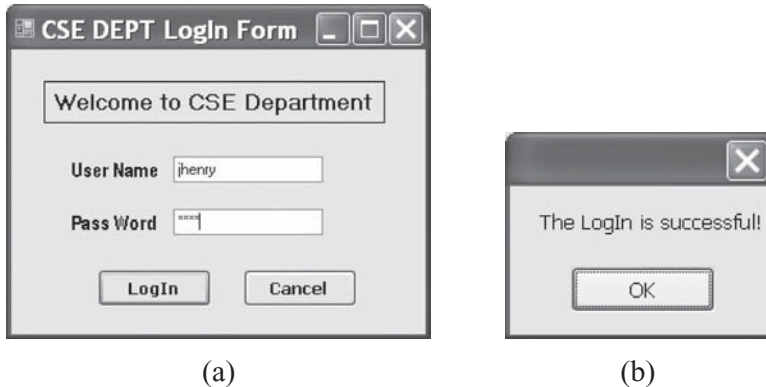


Figure 5.148 Running status of the LogInForm.

- B.** All other forms are closed by executing the `Close()` method, and the project is exited by executing the `Application.Exit()` method.

Now let's run the project to test the codes we did for the two forms, `LogInForm` and `SelectionForm`. Make sure that you have commented out the testing code: `MessageBox.Show("The LogIn is successful!")` since we do not need it if the project runs in the normal conditions. Click on the Start Debugging button to run the project and the `LogInForm` is displayed first, as shown in Figure 5.148a.

Enter a valid username and a password, such as **jhenry** and **test** into the `UserName` and `PassWord` boxes; click on the `LogIn` button. A successful login message is displayed, which is shown in Figure 5.148b. Click on the `OK` button and our next form, `SelectionForm`, is displayed. Our login process is successful! Click on the `Exit` button on the `SelectionForm` to exit the project. Of course, you can test the login process by entering some incorrect username or password, and you will see what happens.

Next let's take care of the coding for the `Faculty` form.

5.19.3.4 Query Data Using LINQ to SQL for Faculty Form

We want to use two methods to perform the data query using the LINQ to SQL in this form. The first method is a general LINQ to SQL method, as we did for the `LogInForm`. However, the second method is to query a single record from the `Faculty` table.

Open the code window of the `FacultyForm` and add the `System.Data.Linq` namespace to the namespace declaration section of this form. Then enter the codes shown in Figure 5.149 into the constructor of this form to perform the initialization jobs for this form.

Let's have a look at this piece of code to see how it works.

- A.** The namespace `System.Data.Linq` is added into the namespace declaration section on this form to allow us to use all LINQ-related components defined in that namespace.
- B.** This coding line is equivalent to clicking on the `Select` button and executing its `Click` method. The purpose of this line is to execute a query for the default faculty and display the queried result in the `FacultyForm` when this form is opened the first time. Anyway, we try to avoid a blank `FacultyForm` when it is opened the first time by calling this method.

```

SQLSelectRTOBJECTLINQ.FacultyForm FacultyForm()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
A using System.Data.Linq;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace SQLSelectRTOBJECTLINQ
{
    public partial class FacultyForm : Form
    {
        public FacultyForm()
        {
            InitializeComponent();
            ComboName.Items.Add("Ying Bai");
            ComboName.Items.Add("Satish Bhalla");
            ComboName.Items.Add("Black Anderson");
            ComboName.Items.Add("Steve Johnson");
            ComboName.Items.Add("Jenney King");
            ComboName.Items.Add("Alice Brown");
            ComboName.Items.Add("Debby Angles");
            ComboName.Items.Add("Jeff Henry");
            ComboName.SelectedIndex = 0;
B this.cmdSelect_Click(this.cmdSelect, null);
C ComboMethod.Items.Add("LINQ to SQL Method");
            ComboMethod.Items.Add("LINQ - Single Record");
            this.ComboMethod.SelectedIndex = 0;
        }
    }
}

```

Figure 5.149 Coding for the constructor of the FacultyForm.

- C. Two query methods are added into the ComboMethod combobox and the first method is selected at the default method.

The next coding is for the Select button's Click method. When the user clicks on this button as the project runs, all faculty information, including the faculty title, office, phone, graduated college, and email, will be queried and displayed in related labels with the faculty photo. We can use codes similar to those for the LogInForm to perform the query in this form; however, before we can do that, we have to indicate that a potential bug exists in the LINQ, especially in the Object Relational Mapping (ORM) engine. As you know, in this query we use the faculty_name as the query criterion and the data type of this column is *text* (refer to Chapter 2 for Faculty table in our sample database CSE_DEPT.mdf). The bug is that the LINQ cannot convert this *text* data type to a data type that is compatible to the entity class. You would encounter an unhandled exception (SQL Server does not handle comparison of NText, Text, Xml, or Image data types) if you develop a similar query as we did for the LogInForm to perform a query to the Faculty table. The reason this query works for the LogIn table is that the data type of the user_name used as the query criterion is NVARCHAR(50), and this data type is recognized and converted to the data type that is compatible with the entity class.

To solve this bug, we need to perform the following two operations:

SQLSelectRTOjectLINQ.FacultyForm		cmdSelect_Click()	
		private void cmdSelect_Click(object sender, EventArgs e)	
		{	
A		LogInForm logForm = new LogInForm();	
		logForm = logForm.getLogInForm();	
B		string strName = ShowFaculty(ComboName.Text);	
		if (strName == "No Match")	
		MessageBox.Show("No Matched Faculty Image found!");	
C		if (ComboMethod.Text == "LINQ to SQL Method")	
		{	
		var f_info = from fi in logForm.cse_dept.Faculties	
		where fi.faculty_name == ComboName.Text	
		select fi;	
D		foreach (var f in f_info)	
		{	
		titleLabel.Text = f.title;	
		OfficeLabel.Text = f.office;	
		PhoneLabel.Text = f.phone;	
		CollegeLabel.Text = f.college;	
		EmailLabel.Text = f.email;	
		}	
E		} else	
		{	
		Faculty faculty = logForm.cse_dept.Faculties.Single(fi => fi.faculty_name == ComboName.Text);	
		titleLabel.Text = faculty.title;	
		OfficeLabel.Text = faculty.office;	
		PhoneLabel.Text = faculty.phone;	
		CollegeLabel.Text = faculty.college;	
		EmailLabel.Text = faculty.email;	
		}	
F		logForm.Close();	
		}	

Figure 5.150 Coding for the constructor of the FacultyForm.

1. Change the data type for the column `faculty_name` in the Faculty table from the `text` to `NVARCHAR(50)`. To do that, open the Faculty table from our sample database `CSE_DEPT` using Microsoft SQL Server Management Studio Express and perform this data type change. Do not forget to save this change when it is done.
2. Open the Design File `CSE_DEPT.designer.cs` located under the folder `CSE_DEPT.dbml` in the Solution Explorer window by double-clicking on this file. Then scroll down until you find the line: `[Column(Storage = "_faculty_name"...)]`. Change the data type from `DbType = "text"` to `DbType = "NVarChar(50)"`.

Now open the Select button's Click method by double-clicking on this button from the design view of the FacultyForm, and enter the code shown in Figure 5.150 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. An instance of the `LogInForm` is created and the `getLogInForm()` method is called to get the original `LogInForm` object we created in this project in Section 5.19.3.2, and assign it to this new instance. In this way, we can access and use the global connection object `CSE_DEPTDataContext` we created in that form.
- B. The user-defined method `ShowFaculty()` is executed to find and display the selected faculty's image. An error message will be displayed if this search has failed.

- C. If the user selected the first method, LINQ to SQL method, a standard LINQ query structure is adopted with an implicit typed local variable `f_info` with a data type `var`. The Visual C# 2008 can automatically convert this `var` to any suitable data type; in this case, it is a collection. An iteration variable `fi` is used to iterate over the result of this query from the Faculty table. Then a similar SQL SELECT statement is executed with the WHERE clause.
- D. The foreach loop is utilized to pick up each column from the selected data row `f`, which is obtained from the `f_info` we get from the LINQ query. Then, assign each column to the associated label control in the FacultyForm window to display them. Since we are using a typed database, we do not have to indicate each column clearly with the `field<string>` and the column's name as the position for each of them. Instead we can directly use each column's name to access each of them.
- E. If the user selected the LINQ–Single Record method, a single LINQ query method is adopted and executed by calling the `Single()` method. The implicit variable `fi` is an `IQueryable<Faculty>` variable, and the queried result is assigned to the `faculty` that is a typed Faculty variable. Each column of the returned data is then assigned to the associated label in the FacultyForm to display each of them one by one.
- F. Finally the new created instance `logForm` is closed. The point is that this object has nothing to do with the original object we created in Section 5.19.3.2.

While it looks like quite simple for this piece of code, it is indeed much simpler and shorter than those codes we did for the same method using the regular SQL Server database queries. No Connection string, Command object, Connection object, or even DataAdapter and DataReader used for this query. That is the reason the LINQ to SQL is implemented more often to replace the old-style SQL Server queries.

The coding for the user-defined method `ShowFaculty()` is identical with what we did for the same method in the project `AccessSelectRTOObject`. Refer to Figure 5.87 for the detailed codes for this method. You can copy the codes from that method and paste them into this FacultyForm code window.

The code for the Back button's Click method is easy. Just put the code line `this.Hide();` into this method.

Before you can test this coding, do not forget to copy all faculty image files from the folder Images located at the accompanying ftp site (see Chapter 1) and paste them to any folder on your computer. In order to avoid providing the full name for these image files, it is recommended that you paste these image files into the folder in which your project executable file is located. In this application, this folder is `C:\Book 6\Chapter 5\SQLSelectRTOObjectLINQ\bin\Debug`.

Now let's run the project to test our coding for this form. Click on the Start Debugging button (or F5 key) to run the project. Enter the suitable username and password, such as `jhenry` and test AND CLICK on Faculty Information from the Selection form to open the FacultyForm window, which is shown in Figure 5.151. The information about the default faculty has been queried and displayed in five labels and PhotoBox. You can try to perform this query using the second method, LINQ–Single Record, with the different faculty members. A sample of queried faculty information for the faculty member Jenney King is shown in Figure 5.151.

Our coding for the LINQ to SQL is successful. Click on the Back button to close this form and on the Exit button to exit the project.

Next let's take care of the coding for the CourseForm window.

Figure 5.151 Running sample of the FacultyForm.

5.19.3.5 Query Data Using Joined LINQ to SQL for Course Form

Only one method, LINQ to SQL, is used for this data query in this CourseForm. Like we did before, first we need to add a `System.Data.Linq` namespace to the namespace declaration section in the CourseForm code window. Then open the code window of the CourseForm and enter the codes into the constructor of the CourseForm shown in Figure 5.152.

Let's take a look at this piece of code to see how it works.

- A. The namespace `System.Data.Linq` is added into the namespace declaration section of this form to allow us to use all LINQ-related components defined in that namespace.
- B. Eight faculty members are added into the `ComboName` combobox control in this form to allow users to select one of them to perform the related course information query. The first faculty member selected is the default one by setting the `SelectedIndex` as 0.
- C. The LINQ to SQL method is added into the `ComboMethod` combobox control, and this is the only method we want to use to perform the data query in this form.

The next coding is for the `Select` button's `Click` method. As this button is clicked on, all courses (`course_id`) taught by the selected faculty will be displayed in the `CourseList` box. As you know, there is no `faculty_name` column available in the `Course` table and each course in the `Course` table is identified by the associated `faculty_id` column. To query the course taught by the selected faculty based on the faculty name, first we need to perform a query to the `Faculty` table to get the `faculty_id`, and then we can query all `course_id` from the `Course` table based on the `faculty_id` we obtained from the first query. This means that we need to perform at least two queries from two tables to finish this `course_id` query. In this part, we try to use a joined LINQ to SQL method to perform this query to simplify this process.

Open the `Select` button's `Click` method by double-clicking on the `Select` button from the design view of the CourseForm window, and enter the codes shown in Figure 5.153 into this method.

```

SQLSelectRTOBJECTLINQ.CourseForm CourseForm()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
A using System.Data.Linq;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace SQLSelectRTOBJECTLINQ
{
    public partial class CourseForm : Form
    {
        public CourseForm()
        {
            InitializeComponent();
B            ComboName.Items.Add("Ying Bai");
            ComboName.Items.Add("Satish Bhalla");
            ComboName.Items.Add("Black Anderson");
            ComboName.Items.Add("Steve Johnson");
            ComboName.Items.Add("Jenney King");
            ComboName.Items.Add("Alice Brown");
            ComboName.Items.Add("Debby Angles");
            ComboName.Items.Add("Jeff Henry");
C            ComboName.SelectedIndex = 0;
            ComboMethod.Items.Add("LINQ to SQL Method");
            ComboMethod.SelectedIndex = 0;
        }
    }
}

```

Figure 5.152 Coding for the constructor of the CourseForm.

```

SQLSelectRTOBJECTLINQ.CourseForm cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
A     LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
B     CourseList.Items.Clear();
C     var courseinfo = from ci in logForm.cse_dept.Courses
                       join fi in logForm.cse_dept.Faculties on ci.faculty_id equals fi.faculty_id
                       where fi.faculty_name == ComboName.Text
                       select new
                       {
                           course_id = ci.course_id
                       };
D     foreach (var cid in courseinfo)
     {
         CourseList.Items.Add(cid.course_id);
     }
E     CourseList.SelectedIndex = 0;
}

```

Figure 5.153 Coding for the Select button Click method.

Let's take a closer look at this piece of code to see how it works.

- A.** An instance of the LogInForm is created, and the getLogInForm() method is called to get the original LogInForm object we created in this project in Section 5.19.3.2, and assign it to this new instance. In this way, we can access and use the global connection object CSE_DEPTDataContext we created in that form.

- B.** The CourseList box control is cleaned up before it can be used to add queried course_id and display them in this control. This coding is very important and necessary, otherwise many duplicated course_ids will be added and displayed in this control without this cleaning action.
- C.** A standard LINQ to SQL structure is generated with the implicit local variable courseinfo. The variables ci and fi are used to represent two iteration variables for two entity classes, Courses and Faculties, to iterate over the result of this query from both tables. The keyword *join* is used to connect these two classes together with the joined condition *on* both equaled faculty_ids in two classes, respectively. The where clause is used to indicate the query criterion, which is the faculty_name located in the Faculty entity class. Then a select new statement is executed, and the selected collection from the Course class is assigned to the column course_id. Note that the relationship between the Faculty and the Course table is one-to-many; therefore, a collection of Courses properties are stored in the Faculty entity class, but a single reference property Faculties is stored in the Course entity class.
- D.** The foreach loop is utilized to pick up each course_id from the selected data collection courseinfo, which is obtained using the joined LINQ to SQL query. Then, add each course_id to the CourseList box control in the CourseForm window to display them. Since we are using a typed database, we do not have to indicate each column clearly with the field<string> and the column's name as the position for each of them. Instead we can directly use each column's name to access each of them.
- E.** This line is very important since there will be no default course_id selected when all course_ids are added into the CourseList box without this code. Another purpose of this line is to trigger the SelectedIndexChanged() method to retrieve and display the detailed course information for the selected or default course_id from the CourseList box when the Select button is clicked on by the user.

Compared with those codes we did for the same method in the previous project, it can be found that this coding is much simpler and shorter. Yes, it is very easy and convenient to develop this piece of code to perform this data query. You can find the advantages of using the LINQ to SQL technique from this sample coding.

The coding for the Back button's Click method is easy. As we did before, just enter the code line this.Hide(); to this method.

Now let's take care of the coding for the SelectedIndexChanged method. The function of this piece of code is after clicking on the Select button to execute its Click method, all course_id taught by the selected faculty will be added and displayed in the CourseList box control. To get the detailed information for each course, the user can click each course_id from the CourseList box control, and all six pieces of detailed information related to the selected course_id are displayed in six textboxes in the Course form.

Open this method by double-clicking on the CourseList box from the CourseForm window, and enter the code shown in Figure 5.154 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** An instance of the LogInForm is created and the getLogInForm() method is called to get the original LogInForm object we created in this project in Section 5.19.3.2, and assign it to this new instance. In this way, we can access and use the global connection object CSE_DEPTDataContext we created in that form.
- B.** A standard LINQ to SQL query structure is adopted, and the implicit variable ci is an IQueryable<Course> variable, and the queried result is assigned to the cinfo, which is a typed Course variable. The iteration variable ci is used to iterate over the result of this

```

SQLSelectRTOjectLINQ.CourseForm CourseList_SelectedIndexChanged()
private void CourseList_SelectedIndexChanged(object sender, EventArgs e)
{
A   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();
B   IQueryable<Course> cinfo = from ci in logForm.cse_dept.Courses
                               where ci.course_id == (string)CourseList.SelectedItem
                               select ci;
C   foreach (Course c in cinfo)
   {
       txtName.Text = c.course1;
       txtSchedule.Text = c.schedule;
       txtClassRoom.Text = c.classroom;
       txtCredits.Text = c.credit.ToString();
       txtEnroll.Text = c.enrollment.ToString();
   }
}

```

Figure 5.154 Coding for the SelectedIndexChanged method.

query from the Course table. Then a similar SQL SELECT statement is executed with the WHERE clause. The query criterion is the course_id selected from the CourseList box.

- C. The foreach loop is utilized to pick up each column from the selected data row *c*, which is obtained from the *cinfo* we get from the LINQ query above. Then, assign each column to the associated textbox control in the CourseForm window to display them. Since we are using a typed database, we do not have to indicate each column clearly with the field<string> and the column's name as the position for each of them. Instead we can directly use each column's name to access each of them. A trick is that the name of the first column on the Course entity class has been changed from **course** to **course1**. Recall that in Chapter 2, the name of this column was **course** in the Course table in our sample database. The reason for this change is that when we built the object-relational model for our sample database in Section 5.19.3.1 to create the Course entity class, there is a duplication between our Course table's name (Course) and the second column's name (course). Note that since SQL Server table and column names aren't case sensitive, in order to avoid this name duplication, the name of the second column in the Course table has been modified from course to course1 in the entity class (refer to Figure 4.43 in Chapter 4).

Compare the codes in Figures 5.153 and 5.154 with those codes developed in Figures 5.117 and 5.118. It can be found that the codes we developed in this section have been significantly simplified and shortened even though they have the same functionalities. This is evidence of the advantage of using the LINQ to SQL over the traditional database query methods.

The last coding we need to do for this form is the Back button's Click method. As we did before, open this method and enter this.Hide(); into this method.

Now we have finished all coding jobs for this form. Let's run the project to test the codes we developed for this form. Click on the Start Debugging button to run the project. Complete the login process and select the Course Information from the SelectionForm to open the CourseForm, which is shown in Figure 5.155.

Keep the default faculty selected and click on the Select button. All courses (course_id) taught by the selected faculty are displayed in the CourseList box. Also the detailed information about the default course_id is displayed in the five textboxes in this form,

Figure 5.155 Running status of the CourseForm.

shown in Figure 5.155. You can try to retrieve and display other faculty's courses by selecting different faculty names from the ComboName combobox with the Select button.

Our joined LINQ to SQL query is very successful! Next let's handle the coding for our last form, StudentForm.

5.19.3.6 Query Data Using LINQ to SQL Stored Procedures for Student Form

In this section, we want to show readers how to call stored procedures with LINQ to SQL to perform the data query. The stored procedures discussed here are two procedures developed in Section 5.19.2.7.3, `dbo.StudentInfo` and `dbo.StudentCourseInfo`. In order to call these two stored procedures in the LINQ to SQL environment, first we need to add these two procedures into the Object Relational Designer, specifically to the Methods pane.

Open the Object Relational Designer by expanding the folder `CSE_DEPT.dbml` and double-clicking on the item `CSE_DEPT.dbml.layout` from the Solution Explorer window. Click on No to the pop-up MessageBox. The opened Object Relational Designer is shown in Figure 5.156.

Right-click on the designer canvas and select the item Show Methods Pane from the pop-up menu to open the Methods pane if it has not yet been opened. We need to create our desired stored procedure by dragging each of them from the Server Explorer window to this Methods pane in this Object Relational Designer since the stored procedure is considered as a method in LINQ to SQL.

Now open the Server Explorer window if it has not yet been opened and connect to our sample database `CSE_DEPT.mdf` if it has not yet been connected. Then expand the folders `CSE_DEPT.mdf` and Stored Procedures to display all stored procedures we developed for the previous projects. We need to use two stored procedures, `StudentInfo` and `StudentCourseInfo`. Just drag each of them from the Server Explorer window and place them in the Methods pane one by one. Your finished Object Relational Designer should match the one shown in Figure 5.156.

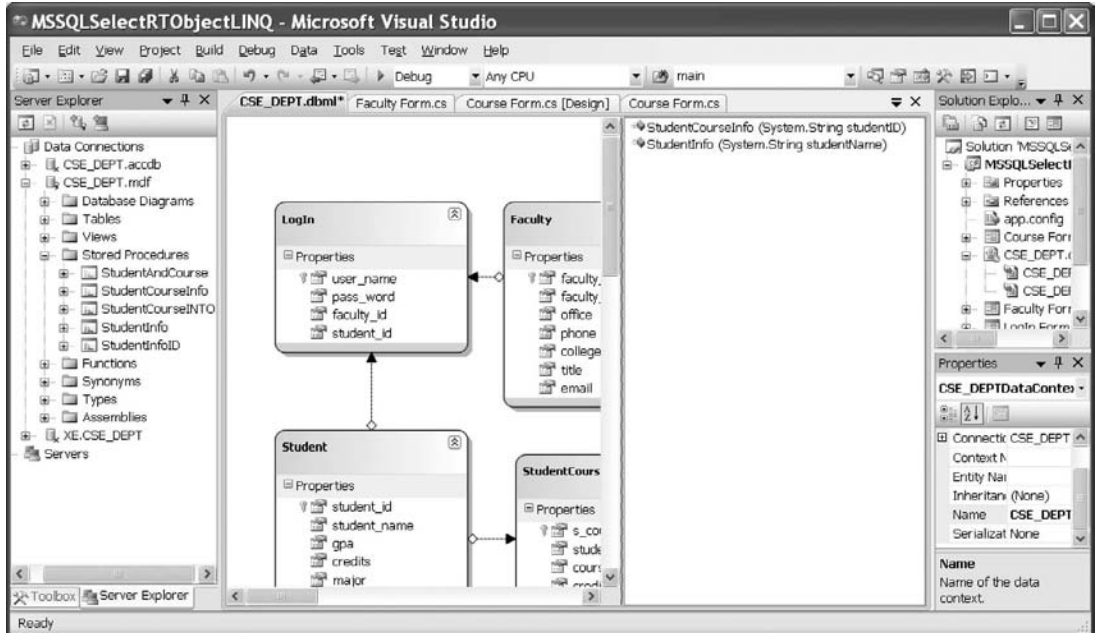


Figure 5.156 Finished Object Relational Designer.

Now let's develop the codes for this form. First, we need to add namespace `System.Data.Linq` to the namespace declaration section in this form. Then open the code window of the `StudentForm` and enter the codes into the constructor of this form shown in Figure 5.157.

Let's take a look at this piece of code to see how it works.

- A.** The namespace `System.Data.Linq` is added into the namespace declaration section on this form to allow us to use all LINQ-related components defined in that namespace.
- B.** Five sample students are added into the `ComboName` combobox to allow users to select one to query the detailed information for the selected student as the project runs. The first student is selected as the default by setting the `SelectedIndex` to 0.
- C.** Only one method, LINQ to SQL, is used for this query.
- D.** This code line is important and it is equivalent to clicking on the `Select` button to execute its `Click` method. In this way, the default student's information can be retrieved and displayed in the `StudentForm` as this form is opened for the first time.

Now let's do the coding for the `Select` button's `Click` method. As you know, two queries are performed to two tables in this `StudentForm`, `Student` and `StudentCourse`. Two stored procedures we developed in Section 5.19.2.7.3 have been added into the `Methods` pane, and they are considered as stored procedures that can be accessed by LINQ to SQL. Thus, as we drag each stored procedure from the `Server Explorer` window and place it in the `Methods` pane, the necessary codes that are used to call the stored procedure from LINQ to SQL are generated by the designer.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
A using System.Data.Linq;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace SQLSelectRTOBJECT.LINQ
{
    public partial class StudentForm : Form
    {
        public StudentForm()
        {
B            InitializeComponent();
            ComboName.Items.Add("Erica Johnson");
            ComboName.Items.Add("Ashly Jade");
            ComboName.Items.Add("Holes Smith");
            ComboName.Items.Add("Andrew Woods");
            ComboName.Items.Add("Blue Valley");
            ComboName.SelectedIndex = 0;
C            ComboMethod.Items.Add("LINQ to SQL Method");
            ComboMethod.SelectedIndex = 0;
D            cmdSelect_Click(this.cmdSelect, null);
        }
    }
}

```

Figure 5.157 Coding for the constructor of the StudentForm.

Open the Select button's Click method by double-clicking on this button from the design view of the StudentForm window, and then enter the codes shown in Figure 5.158 into this method.

Let's take a close look at this piece of code to see how it works.

- A.** An instance of the LogInForm is created, and the getLogInForm() method is called to get the original LogInForm object we created in this project in Section 5.19.3.2, and assign it to this new instance. In this way, we can access and use the global connection object CSE_DEPTDataContext we created in that form.
- B.** The user-defined method FindName() is called to find and display a matched student's image in the PhotoBox control in the StudentForm. The returned name of the student's image file is assigned to the local string variable strName. An error message will be displayed if no matched student's image can be found.
- C.** The stored procedure StudentInfo() is called to query the detailed information for the selected student. The returned queried collection is assigned to an implicit local variable sinfo. The argument of this stored procedure is the student's name stored in the ComboName combobox.
- D.** A foreach loop is utilized to pick up each column from the selected data row si, which is obtained from the sinfo we get from the LINQ query above. Then, assign each column to the associated textbox control in the StudentForm window to display them. Since we are using a typed database, we do not have to indicate each column clearly with the field<string> and the column's name as the position for each of them. Instead we can directly use each column's name to access each of them.

```

SQLSelectRTOBJECTLINQ.StudentForm  cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
A   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();
   string strName = string.Empty;
B   strName = FindName(ComboName.Text);
   if (strName == "No Match")
       MessageBox.Show("No Matched Student's Image Found!");
C   var sinfo = logForm.cse_dept.StudentInfo(ComboName.Text);
D   foreach (var si in sinfo)
   {
       txtID.Text = si.student_id;
       txtSchoolYear.Text = si.schoolYear;
       txtGPA.Text = si.gpa.ToString();
       txtMajor.Text = si.major;
       txtCredits.Text = si.credits.ToString();
       txtEmail.Text = si.email;
   }
E   CourseList.Items.Clear();
F   var scinfo = logForm.cse_dept.StudentCourseInfo(txtID.Text);
G   foreach (var sc in scinfo)
   {
       CourseList.Items.Add(sc.course_id);
   }
}

```

Figure 5.158 Coding for the Select button's Click method.

- E.** The CourseList box is cleaned up before we can add any queried course_id into this control.
- F.** The stored procedure StudentCourseInfo() is called to query the detailed course information for the selected student. The returned queried collection is assigned to an implicit local variable scinfo. The argument of this stored procedure is the student_id we obtained from calling the first stored procedure, and it is stored in the txtID textbox.
- G.** A foreach loop is utilized to pick up each course_id from the selected data rows sc, which is obtained from the scinfo we get from the LINQ query above. Then, each course_id is added into the CourseList box control in the StudentForm window to display them.

The coding for the user-defined method FindName() is identical with one we did for the same method in Section 5.18.5.2. Refer to Figure 5.99 to get the detailed codes for this method.

The coding for the Back button's Click method is easy. Just open that method and enter the code **this.Hide();** into this method. At this point, we have finished all code development for this project. Let's run the project to test our coding. Click the Start Debugging button to run the project, complete the login process and select the item Student Information from the SelectionForm to open the StudentForm, which is shown in Figure 5.159.

Select the detailed student information by first choosing the desired student name from the Student Name combobox and then clicking on the Select button. All information related to the selected student is displayed in six textboxes, and all courses taken by that student are displayed in the CourseList box, which is shown in Figure 5.159.

Click on the Back and then on the Exit button to close our project. Our project is successful!

Figure 5.159 Running status of the StudentForm.

A complete project, `MSSQLSelectRTOObjectLINQ`, which used LINQ to SQL query methods and stored procedures, can be found in the folder `DBProjects/Chapter 5` that is located at the accompanying ftp site (see Chapter 1). You can copy and paste it on your computer to run it. Of course, the sample database file `CSE_DEPT.mdf` must have been created and located at the default database folder.

Now let's go to the last part in this chapter—develop a data-driven application using the real-time object with the Oracle database.

5.20 QUERY DATA USING RUNTIME OBJECTS TO ORACLE DATABASE

For your convenience, in this section we will use our sample database `CSE_DEPT` developed in Chapter 2 with the Oracle Database 10g Express Edition Release 2 installed in our local computer. Refer to Appendix C to get details on how to download and install Oracle Database 10g on your machine and Chapter 2 on how to build the sample database `CSE_DEPT`.

Three query methods, `DataAdapter`, `DataReader`, and `LINQ to DataSet`, are discussed in this section. The first two methods belong to the general runtime objects, and the third method is the `LINQ to DataSet` since there is no direct connection between the `LINQ` and Oracle database.

5.20.1 Oracle Database 10g Express Edition Release 2

In this section, we use the Oracle Database 10g Express Edition Release 2 (Oracle Database 10g XE R2) as our database provider. Oracle Database 10g Express Edition (Oracle Database XE R2) is an entry-level, small-footprint starter database with the following advantages:

- Free to download and install on your local computer or remote computers
- Free to develop and deploy data-driven applications
- Free to distribute (including Independent Software Vendors (ISVs))

The Oracle Database XE R2 is built using the same code base as Oracle Database 10g Release 2 product line—Standard Edition One, Standard Edition, and Enterprise Edition, and is available on 32-bit Windows and Linux platforms.

Although there are limitations for the Oracle Database 10g XE R2, such as up to 4 GB upper bound of the user data and the single instance only on any server, it is still an ideal and convenient tool to develop professional and leading-edge data-driven applications with the following specific functionalities:

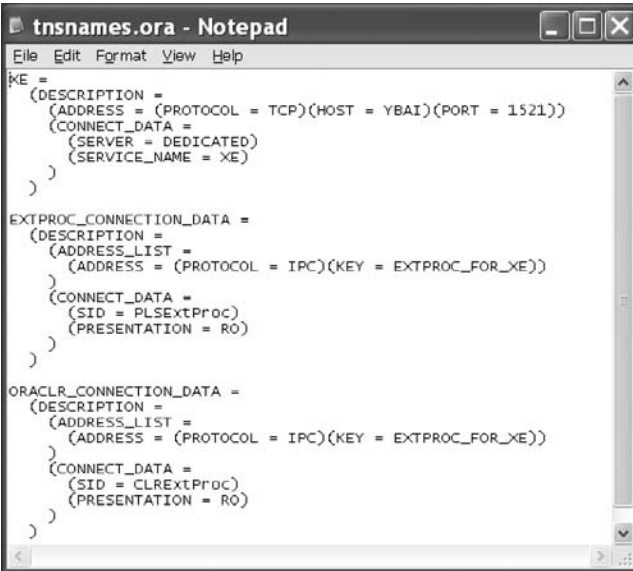
- The Oracle Database 10g XE R2 can be easily upgraded to Standard Edition One, Standard Edition, and Enterprise Edition.
- Any application developed for Oracle Database XE will run completely unchanged with Oracle Database 10g Standard Edition One, Standard Edition, or Enterprise Edition, and the application development investment is guaranteed.
- With Oracle Database XE R2, ISVs have the industry's leading database technology to power their applications. Distributing Oracle Database XE R2 in their applications or products without additional cost offers even greater value to their customers.
- Oracle Database XE R2 can be freely distributed as a stand-alone database or as part of a third-party application or product.

For most applications, you only need to download and install the Oracle Database XE R2 Server component since it provides both an Oracle database and tools for managing this database. It also includes the Client component of Oracle Database XE, so that you can connect to the database from the same computer on which you installed the Server, and then administer the database and develop Visual C# 2008 applications. Before we can start to build our sample projects, we should have a clear picture about the structure of the connection string for the Oracle Database 10g XE R2.

5.20.2 Configure Oracle Database Connection String

As we mentioned in Section 5.19.1, there are different ways to build a connection string for the Oracle database connection. One way is to use the database alias defined in the `tnsnames.ora` file. This file is created automatically after you install the Oracle database 10g XE R2 on your machine. During the installation process, you will be prompted to enter your username and password. Normally the username is **SYSTEM** or **SYS**, which is defined by the Oracle system, and you need to select your password. Remember, you need these two pieces of information to access your database each time you want to create, edit, and manipulate your database in the future.

In order to use the database alias defined in the `tnsnames.ora` file, first you need to open this file to take a look at the content of this definition. This file should be located at the folder `C:\oracle\app\oracle\product\10.2.0\server\NETWORK\ADMIN` after the Oracle Database 10g XE R2 is installed. A sample file is shown in Figure 5.160. You can open this file using any text editor such as NotePad, WordPad, or MS Office Word. The



```

XE =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = YBAI)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = XE)
    )
  )

EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC_FOR_XE))
    )
    (CONNECT_DATA =
      (SID = PLSExtProc)
      (PRESENTATION = RO)
    )
  )

ORAclr_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC_FOR_XE))
    )
    (CONNECT_DATA =
      (SID = CLRExtProc)
      (PRESENTATION = RO)
    )
  )

```

Figure 5.160 Sample of the file tnsnames.ora.

database alias for our application is XE, and the top block of this file is the definition of the database alias (refer to Figure 5.160).

Close this file and now let's create our connection string for the Oracle database 10g XE R2 using the database alias XE. The connection string can be defined as:

```

string oraString = "Data Source=XE;" + _
                  "User ID=system;" + _
                  "Password=reback"

```

where the password reback is the password we used when we installed the Oracle Database 10g XE R2 in our computer.

Another way to create the connection string is to copy the top block from the tnsnames.ora file and paste it as the value of the Data Source parameter, which is:

```

string oraString = "Data Source=(DESCRIPTION=" +
                  "(ADDRESS=(PROTOCOL=TCP)(HOST=YBAI)(PORT=1521))" +
                  "(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME = XE));" +
                  "User ID=system;Password=reback;"

```

In the following sample projects, we used the first way to create our connection string. With the connection string ready, now we can start to develop our sample projects. First let's start with the general runtime objects method.

5.20.3 Query Data Using General Runtime Objects

First, we need to create a new Visual C# 2008 Windows-Based project named OracleSelectRTOject. For your convenience, a blank project with five forms has been

developed, and you can directly use this project to develop your application. This blank project OracleSelectRTOBJECT is located at the folder DBProjects\Chapter 5 in the accompanying ftp site (see chapter 1).

5.20.3.1 Query Data Using General Runtime Objects for LogIn Form

Open the LogIn form window by clicking on the View Designer button from the Solution Explorer window. Then we need to add the Oracle Client namespace in which the Oracle Data Provider and associated classes are located, and make this namespace a reference for our project since we need to use those classes for our Oracle database operations later. Right-click on the OracleSelectRTOBJECT from the Solution Explorer window and select the Add Reference item from the pop-up menu to open the Add reference window. With the .NET tab selected, scroll down the list until you find the item System.Data.OracleClient, click on it to select it, and click on OK to add this reference to our project.

Some readers may have found a problem, which is that we did not perform this adding reference job for our previous projects, either AccessSelectRTOBJECT or SQLSelectRTOBJECT. The reason for that is because the namespaces for those two Data Providers are default namespaces and have been added automatically by the Visual C#.NET 2008 as you open a new project.

5.20.3.1.1 Declare Runtime Objects As we mentioned in Chapter 3, all components related to the Oracle Data Provider supplied by ADO.NET are located at the namespace System.Data.OracleClient. To access the Oracle database, you need to use this Data Provider. Besides adding this namespace reference, you also need to declare this namespace at the top line of your code window to allow Visual C#.NET 2008 to know that you want to use this specified Data Provider. Open the Code Window by clicking on the View Code button from the Solution Explorer window and enter the codes to the top of this code window, as shown in Figure 5.161.

The first thing we need to do is to connect our project with our sample database CCSE_DEPT we built in Chapter 2.

5.20.3.1.2 Connect to Data Source with Runtime Objects Since the connection job is the first thing we need to do before we can make any data query, we need to do the connection job in the constructor of the LogInForm class to allow the connection to be made first as the project runs. Open the constructor and enter the codes shown in Figure 5.162 into this constructor.

The screenshot shows a code window titled 'OracleSelectRTOBJECT.LogInForm' with a dropdown menu showing 'LogInForm()'. The code inside the window is as follows:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OracleClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

Figure 5.161 Declaration of the namespace for the Oracle Data Provider.


```

OracleSelectRTOject.LogInForm | LogInForm()
namespace OracleSelectRTOject
{
    public partial class LogInForm : Form
    {
        A public OracleConnection oraConnection;
        public LogInForm()
        {
            B InitializeComponent();
            string oraString = "Data Source = XE;" +
                "User ID = CSE_DEPT;" +
                "Password = reback";
            C oraConnection = new OracleConnection(oraString);
            D try
            {
                oraConnection.Open();
            }
            catch (OracleException e)
            {
                MessageBox.Show("Oracle Error");
                MessageBox.Show("Error Code = " + e.ErrorCode);
                MessageBox.Show("Error Message = " + e.Message);
            }
            catch (InvalidOperationException e)
            {
                E MessageBox.Show("Invalid Message = " + e.Message);
            }
            if (oraConnection.State != ConnectionState.Open)
            {
                MessageBox.Show("Database connection is Failed");
                Application.Exit();
            }
        }
    }
}

```

Figure 5.162 Coding for the constructor of the LogInForm.

Let's take a look at this piece of code to see how it works.

- A.** A new instance of the OracleConnection class is declared with the accessing mode of public, which means that we want to use this connection object as a global object and allow all objects and methods defined in our whole project to use it.
- B.** The Oracle database connection string is defined first. Refer to Section 5.20.2 to get a detailed description about this connection string. An addition operator + can be used to concatenate multiple substrings to form a complete connection string for the Oracle database.
- C.** A new Oracle Connection instance is created and initialized with the connection string oraString we created in step B.
- D.** A try...catch block is utilized to try to catch up any mistake or error caused by opening this connection. The advantage of using this kind of strategy is to avoid unnecessary system debug process and simplify this debug procedure.
- E.** This step is used to confirm that our database connection is successful. If not, an error message is displayed and the project is exited.

After a database connection is successfully made, we need to use this connection to access the Oracle database to perform our data query job. We will use two methods to perform the data query for this LogInForm: DataAdapter and DataReader methods.

5.20.3.1.3 Coding for Method 1: Using DataAdapter to Query Data Open the LogIn form window by clicking on the View Designer button, and then double-click on the TabLogIn button to open its Click method. Enter the codes shown in Figure 5.163 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** The SELECT statement for Oracle is basically the same as for the SQL Server database, but a little difference exists between the Oracle and SQL Server database. The difference is assigning the parameter in the WHERE clause. In both Microsoft Access and SQL Server, an @ symbol is prefixed before the parameter and an equals the word symbol or LIKE is used to assign a parameter to the column. In Oracle, an equals symbol is also used, but a colon must be prefixed before the parameter. In our case, two dynamic parameters, UserName and PassWord, are assigned to two columns, user_name and pass_word, in the form of user_name=:UserName and pass_word=:PassWord.
- B.** Two Oracle Parameter objects are created, and these two objects will be attached to the Command object to construct a complete command object that can be used to perform the data query later. Other data components used in this method, such as the DataAdapter, Command, and DataTable, are also created here. A new instance of our SelectionForm

```

OracleSelectRTOject.LogInForm
cmdTabLogIn_Click()

private void cmdTabLogIn_Click(object sender, EventArgs e)
{
  A  string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
  B  cmdString += "WHERE user_name=:UserName AND pass_word=:PassWord";
  OracleParameter oraUserName = new OracleParameter();
  OracleParameter oraPassWord = new OracleParameter();
  OracleDataAdapter LogInDataAdapter = new OracleDataAdapter();
  OracleCommand oraCommand = new OracleCommand();
  C  DataTable oraDataTable = new DataTable();
  SelectionForm selForm = new SelectionForm();

  oraUserName.ParameterName = "UserName";
  oraUserName.OracleType = OracleType.Char;           //very important in some applications
  oraUserName.Value = txtUserName.Text;
  oraPassWord.ParameterName = "PassWord";
  oraPassWord.OracleType = OracleType.Char;           //very important in some applications
  oraPassWord.Value = txtPassWord.Text;
  D  oraCommand.Connection = oraConnection;
  oraCommand.CommandType = CommandType.Text;
  oraCommand.CommandText = cmdString;
  E  oraCommand.Parameters.Add(oraUserName);
  oraCommand.Parameters.Add(oraPassWord);
  F  LogInDataAdapter.SelectCommand = oraCommand;
  LogInDataAdapter.Fill(oraDataTable);
  G  if (oraDataTable.Rows.Count > 0)
  {
    selForm.Show();
    this.Hide();
  }
  H  else
    MessageBox.Show("No matched username/password found!");
  I  oraDataTable.Dispose();
  oraCommand.Dispose();
  LogInDataAdapter.Dispose();
}

```

Figure 5.163 Coding for the TabLogIn button click method.

class, selForm, is also created since it is our next form to be displayed if the login process is successful.



The SELECT statement used for the Oracle database is different from that used for SQL Server and Microsoft Access. The difference is assigning parameters to the columns in the WHERE clause. A colon must be prefixed before the parameter to be assigned to the column.

- C. Two dynamic parameters are assigned to the OracleParameter objects, paramUserName and paramPassWord, separately. The parameter's name must be identical with the name of the dynamic parameter in the SQL statement string. The parameter's type (OracleType.Char) must be indicated clearly, although it may work without this indication. The Values of two parameters should be equal to the contents of two associated textbox controls, which will be entered by the user as the project runs.
- D. The Command object is initialized with the Connection object, CommandType and CommandText properties.
- E. Now two parameter objects are added into the Parameters collection, which is the property of the Command object using the Add() method, and the command object is ready to be used.
- F. It is then assigned to the property SelectCommand of the DataAdapter, and the Fill() method is called with the oraDataTable as the argument to fill the LogIn table.
- G. By checking the Rows.Count property of the oraDataTable, we can determine whether this fill is successful or not. If the value of this property is greater than 0, which means that the LogIn table is filled by at least one row, the fill is successful and the next form window, Selection form, will be displayed to continue the project to the next step.
- H. Otherwise an error message will be displayed.
- I. A cleaning job is performed to release all data objects used in this method.

Next let's perform the coding for our second method.

5.20.3.1.4 Coding for Method 2: Using DataReader to Query Data Open the LogIn form window by clicking the View Designer button from the Solution Explorer window, and then double-click on the ReadLogIn button to open its Click method. Enter the codes shown in Figure 5.164 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. As the coding for Method 1, the parameter must be preceded by a colon for the WHERE clause in the second query string, and this is the requirement of the data query format for the Oracle database.
- B. Two OracleParameter objects are created, and they will be used to fill two dynamic parameters used in this application. The dynamic parameters will be entered by the user when the project runs.
- C. Two Parameter objects are filled by two dynamic parameters. Note that the ParameterName property is used to hold the nominal value of the dynamic parameter, UserName. The nominal value must be identical with that defined in the SQL query statement. The same situation is true for the second nominal parameter's value.

```

OracleSelectRTOObject.LogInForm cmdReadLogIn_Click()
private void cmdReadLogIn_Click(object sender, EventArgs e)
{
A   string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
B   cmdString += "WHERE user_name=:UserName AND pass_word=:PassWord";
   OracleParameter oraUserName = new OracleParameter();
   OracleParameter oraPassWord = new OracleParameter();
   OracleCommand oraCommand = new OracleCommand();
   OracleDataReader oraDataReader;
   SelectionForm selForm = new SelectionForm();
C   oraUserName.ParameterName = "UserName";
   oraUserName.OracleType = OracleType.Char;
   oraUserName.Value = txtUserName.Text;
   oraPassWord.ParameterName = "PassWord";
   oraPassWord.OracleType = OracleType.Char;
   oraPassWord.Value = txtPassWord.Text;
   oraCommand.Connection = oraConnection;
   oraCommand.CommandType = CommandType.Text;
   oraCommand.CommandText = cmdString;
   oraCommand.Parameters.Add(oraUserName);
   oraCommand.Parameters.Add(oraPassWord);
D   oraDataReader = oraCommand.ExecuteReader();
   if (oraDataReader.HasRows == true)
   {
       selForm.Show();
       this.Hide();
   }
   else
       MessageBox.Show("No matched username/password found!");
   oraCommand.Dispose();
   oraDataReader.Close();
}

```

Figure 5.164 Coding for the ReadLogIn button Click method.

```

OracleSelectRTOObject.LINQ.LogInForm cmdCancel_Click()
private void cmdCancel_Click(object sender, EventArgs e)
{
   oraConnection.Close();
   oraConnection.Dispose();
   Application.Exit();
}
public LogInForm getLogInForm()
{
   return this;
}

```

Figure 5.165 Coding for two methods.

- D.** The ExecuteReader() method is called to perform the data query and the returned data should be filled in the DataReader.

The rest of the coding is identical with the coding we did for the first method.

The coding for the Cancel command button Click method and the user-defined method getLogInForm() are shown in Figure 5.165.

The coding in Figure 5.165 is straightforward with no tricks. First, we need to close our connected database before we can exit our project. The method getLogInForm() is

used to return the current LogInForm object to allow other objects or methods to use it to access the global connection object oraConnection created in this form object. You can test these pieces of coding by running the project if you like. A SelectionForm window will be displayed if the login process is successful.

5.20.3.2 Coding for Selection Form

Most coding in this form is identical with the coding of the Selection form in the project SQLSelectRTOObject. The only difference is the coding for the Exit command button Click method. In that project, an SQL Server database is used, and all Data Provider–dependent objects are preceded by the prefix sql, such as sqlConnection. In this project we used an Oracle database; thus the connection object should be preceded by the prefix ora. When we click on the Exit button, we need to check whether the connection object has been closed and released. Since our global connection object oraConnection is created in the LogInForm class, we need first to use the getLogInForm() method to get that object. For your convenience, we show the coding for this form again in Figure 5.166.

```

namespace OracleSelectRTOObject
{
    public partial class SelectionForm : Form
    {
        FacultyForm facultyForm = new FacultyForm();
        CourseForm courseForm = new CourseForm();
        StudentForm studentForm = new StudentForm();
        public SelectionForm()
        {
            InitializeComponent();
            this.ComboSelection.Items.Add("Faculty Information");
            this.ComboSelection.Items.Add("Course Information");
            this.ComboSelection.Items.Add("Student Information");
            this.ComboSelection.SelectedIndex = 0;
        }
        private void cmdOK_Click(object sender, EventArgs e)
        {
            if (this.ComboSelection.Text == "Faculty Information")
                facultyForm.Show();
            else if (this.ComboSelection.Text == "Course Information")
                courseForm.Show();
            else
                studentForm.Show();
        }
        private void cmdExit_Click(object sender, EventArgs e)
        {
            LogInForm logForm = new LogInForm();
            logForm = logForm.getLogInForm();
            if (logForm.oraConnection.State == ConnectionState.Open)
                logForm.oraConnection.Close();
            logForm.Close();
            courseForm.Close();
            facultyForm.Close();
            studentForm.Close();
            Application.Exit();
        }
    }
}

```

Figure 5.166 Coding for the SelectionForm.

5.20.3.3 Query Data Using Runtime Objects for Faculty Form

In this section, we try to use three query methods to perform the query for the faculty information—DataAdapter, DataReader, and LINQ to DataSet. First we need to add the namespace `System.Data.OracleClient` to the namespace declaration section on this form, and then let's take a look at the coding for the constructor of the `FacultyForm`.

- A. The namespace of the Data Provider, `System.Data.OracleClient`, is utilized for the last project since an SQL Server database is used. Since we use the Oracle database in this section, change the namespace to `System.Data.OracleClient`, which is shown in Figure 5.167.
- B. The `FacultyLabel` object array is created, and it is mapped to 7 columns in the Faculty table.
- C. Three query methods, DataAdapter, DataReader, and LINQ to DataSet, are added into the `ComboMethod` control to allow users to select one of them to perform the data query. The first method is selected as the default one.

Next coding is for the `Select` button `Click` method. Open this method and enter the codes into this method shown in Figure 5.168.

Let's take a closer look at this piece of code to see how it works.

```

OracleSelectRTOject.FacultyForm FacultyForm()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
A using System.Data.OracleClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace OracleSelectRTOject
{
    public partial class FacultyForm : Form
    {
    B private Label[] FacultyLabel = new Label[7];
        public FacultyForm()
        {
            InitializeComponent();
            ComboName.Items.Add("Ying Bai");
            ComboName.Items.Add("Satish Bhalla");
            ComboName.Items.Add("Black Anderson");
            ComboName.Items.Add("Steve Johnson");
            ComboName.Items.Add("Jenney King");
            ComboName.Items.Add("Alice Brown");
            ComboName.Items.Add("Debby Angles");
            ComboName.Items.Add("Jeff Henry");
            ComboName.SelectedIndex = 0;
            this.cmdSelect_Click(this.cmdSelect, null);
            C ComboMethod.Items.Add("DataAdapter Method");
            ComboMethod.Items.Add("DataReader Method");
            ComboMethod.Items.Add("LINQ to DataSet Method");
            this.ComboMethod.SelectedIndex = 0;
        }
    }

```

Figure 5.167 Coding for the constructor of the `FacultyForm`.

```

OracleSelectRTOject.FacultyForm  cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
A   string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty ";
B   cmdString += "WHERE faculty_name=:FacultyName";
   OracleParameter paramFacultyName = new OracleParameter();
   OracleDataAdapter FacultyDataAdapter = new OracleDataAdapter();
   OracleCommand oraCommand = new OracleCommand();
   DataTable oraDataTable = new DataTable();
   OracleDataReader oraDataReader;
   DataSet ds = new DataSet();
   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();
C   paramFacultyName.ParameterName = "FacultyName";
   paramFacultyName.OracleType = OracleType.Char;
   paramFacultyName.Value = ComboName.Text;
D   oraCommand.Connection = logForm.oraConnection;
   oraCommand.CommandType = CommandType.Text;
   oraCommand.CommandText = cmdString;
E   oraCommand.Parameters.Add(paramFacultyName);
F   string strName = ShowFaculty(ComboName.Text);
   if (strName == "No Match")
       MessageBox.Show("No Matched Faculty Image found!");
G   if (ComboMethod.Text == "DataAdapter Method")
   {
       FacultyDataAdapter.SelectCommand = oraCommand;
       FacultyDataAdapter.Fill(oraDataTable);
H   if (oraDataTable.Rows.Count > 0)
       FillFacultyTable(ref oraDataTable);
I   else
       MessageBox.Show("No matched faculty found!");
J   oraDataTable.Dispose();
       FacultyDataAdapter.Dispose();
   }
   else if (ComboMethod.Text == "DataReader Method")
   {
K   oraDataReader = oraCommand.ExecuteReader();
L   if (oraDataReader.HasRows == true)
       FillFacultyReader(oraDataReader);
M   else
       MessageBox.Show("No matched faculty found!");
N   oraDataReader.Close();
   }
   else //LINQ to DataSet Method is selected
   {
O   FacultyDataAdapter.SelectCommand = oraCommand;
       FacultyDataAdapter.Fill(ds, "Faculty");
P   var facultyinfo = (from fi in ds.Tables["Faculty"].AsEnumerable()
                       where fi.Field<string>("faculty_name").Equals(ComboName.Text)
                       select fi);
Q   foreach (var fRow in facultyinfo)
   {
       this.TitleLabel.Text = fRow.Field<string>("title");
       this.OfficeLabel.Text = fRow.Field<string>("office");
       this.PhoneLabel.Text = fRow.Field<string>("phone");
       this.CollegeLabel.Text = fRow.Field<string>("college");
       this.EmailLabel.Text = fRow.Field<string>("email");
   }
   }
}

```

Figure 5.168 Coding for the Select button Click method.

- A.** The query string is created first with 7 querying columns. The dynamic parameter must be prefixed by a colon for the WHERE clause, such as “WHERE faculty_name=:FacultyName”, since this is the requirement of the data query in the Oracle database.
- B.** An OracleParameter object is created, and it is used to hold the dynamic parameter’s name and value later. All other data components are also created in this part, which include DataAdapter, Command, DataTable, DataReader, and DataSet, used for the LINQ to DataSet query method. A new instance of the LogInForm class is used to allow us to access the global connection object created in that class.
- C.** The OracleParameter object is initialized by assigning it with the parameter’s name, type, and value. In this initialization process, assigning OracleType to this Parameter object is very important and a running error may occur if this assignment is missed.
- D.** The OracleCommand object is initialized by assigning it with three properties.
- E.** The initialized Parameter object is added into the Parameters collection of the Command object that is read to be used.
- F.** The user-defined method ShowFaculty() is called to find and display the matched faculty image in the PhotoBox in the FacultyForm. An error message will be displayed if no matched faculty image can be found.
- G.** If the user selected the DataAdapter Method, the initialized command object is assigned to the SelectCommand property of the DataAdapter, and the Fill() method is executed to fill the Faculty table based on the query string.
- H.** If the Count property of the Faculty table is greater than 0, which means that at least one row has been filled into the table and this fill is successful, the user-defined method FillFacultyTable() is executed to assign each queried column to the associated label in this form to display each of them.
- I.** Otherwise an error message is displayed to indicate that this fill has failed.
- J.** Some cleaning jobs are performed to release data components used in this part.
- K.** If the user selected the DataReader method, the ExecuteReader() method is called to perform a read-only operation to the Faculty table, and the queried columns are returned to the DataReader object.
- L.** By checking the HasRows property of the DataReader, we can confirm whether this reading is successful or not. The FillFacultyReader() method will be called to fill each queried column to the associated label in this form if this reading is fine.
- M.** Otherwise an error message is displayed to indicate that this reading has failed.
- N.** A cleaning job is performed to release the DataReader.
- O.** If the user selected the LINQ to SQL method, the initialized command object is assigned to the SelectCommand property of the DataAdapter, and the Fill() method is executed to initialize our DataSet and fill the Faculty table inside the DataSet. Since we create this DataSet by using the runtime command, it is a nontyped DataSet and we have to indicate this table by its name, not the column.
- P.** A typical LINQ query structure is created and executed to retrieve back all related information for the selected faculty member. The facultyinfo is a Visual C# 2008 implicitly typed local variable with a data type var. The C# 2008 will be able to automatically convert this var to any suitable data type. In this case, it is a data table, when it sees it. An iteration variable fi is used to iterate over the result of this query from the Faculty table. Then a similar SQL SELECT statement is executed with the WHERE clause. The key point for this structure is the operator AsEnumerable(). Since different database systems use dif-


```

OracleSelectRTObject.FacultyForm  FillFacultyReader()
A private void FillFacultyReader(OracleDataReader FacultyReader)
{
    int intIndex = 0;
    for (intIndex = 0; intIndex <= 6; intIndex++) //Initialize the object array
        FacultyLabel[intIndex] = new Label();
    MapFacultyTable(FacultyLabel);
    while (FacultyReader.Read())
    {
        for (intIndex = 0; intIndex <= FacultyReader.FieldCount - 1; intIndex++)
            FacultyLabel[intIndex].Text = FacultyReader.GetString(intIndex);
    }
}

```

Figure 5.169 Modified coding for the FillFacultyReader() method.

ferent collections and query operators, those collections must be converted to a type of `IEnumerable<T>` in order to use the LINQ technique because all data operations in LINQ use a SQO methods that can perform complex data queries on an `IEnumerable<T>` sequence. A compiling error would be encountered without this operator.

- Q.** The foreach loop is utilized to pick up each column from the selected data row `fRow`, which is obtained from the `facultyinfo` we get from the LINQ query. Then, assign each column to the associated label in the `FacultyForm` window to display them. Since we are using a nontyped `DataSet`, we must indicate each column clearly with the `field<string>` and the column's name as the position for each of them.

The coding for four user-defined methods, `FillFacultyTable()`, `MapFacultyTable()`, `ShowFaculty()`, and `FillFacultyReader()`, is basically identical with what we did for the same methods in the `AccessSelectRTObject` project developed in Section 5.18.3. For the first three methods, no modification is needed. Refer to Figures 5.84, 5.85, and 5.87 in that section for the detailed codes for those three methods. The only modification is to change the nominal argument's type from `OleDbDataReader` to `OracleDataReader` for the `FillFacultyReader()` method in Figure 5.86. Step **A** in Figures 5.169 shows this modification.

The coding for the Back button Click method is easy. Open this method and just enter the code line `this.Hide();` into that method. That's all!

Now you can test this piece of code by running the project. After the project runs, enter the suitable username and password to complete the login process, and select the Faculty Information item from the `SelectionForm` to open the `FacultyForm`. The default faculty's information should be retrieved and displayed already. You can try to use different methods to perform the data query for different faculty members.

Next let's handle the coding for the `CourseForm`.

5.20.3.4 Query Data Using Runtime Objects for Course Form

In this section, we try to use three query methods to perform the query for the course information: `DataAdapter`, `DataReader`, and LINQ to `DataSet`. We will use the stored procedures developed in the Oracle Database 10g environment to replace the general data query commands to simplify the query structure and improve the query efficiency.

```

OracleSelectRTOject.CourseForm CourseForm()
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
A using System.Data.OracleClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace OracleSelectRTOject
{
    public partial class CourseForm : Form
    {
B         private TextBox[] CourseTextBox = new TextBox[6];
            DataSet ds = new DataSet();

            public CourseForm()
            {
C                 InitializeComponent();
                    ComboName.Items.Add("Ying Bai");
                    ComboName.Items.Add("Satis Bhalla");
                    ComboName.Items.Add("Black Anderson");
                    ComboName.Items.Add("Steve Johnson");
                    ComboName.Items.Add("Jenney King");
                    ComboName.Items.Add("Alice Brown");
                    ComboName.Items.Add("Debby Angles");
                    ComboName.Items.Add("Jeff Henry");
                    ComboName.SelectedIndex = 0;
                    ComboMethod.Items.Add("DataAdapter Method");
                    ComboMethod.Items.Add("DataReader Method");
                    ComboMethod.Items.Add("LINQ & DataSet Method");
                    ComboMethod.SelectedIndex = 0;
            }
    }
}

```

Figure 5.170 Coding for the constructor of the CourseForm.

First let's add the namespace `System.Data.OracleClient` to the namespace declaration section on this form and add the code shown in Figure 5.170 into the constructor of this form.

Let's take a look at this piece of code to see how it works.

- A. The namespace `System.Data.OracleClient` is added into the namespace declaration of this form since all Data Provider–dependent components are located at that namespace, and we need to add this namespace first before we can use any component defined in that namespace.
- B. A `TextBox` array `CourseTextBox[]` with 6 elements is created here, and this array is used to map and save five queried columns in the Course table. Also a `DataSet` instance is created and this is for our LINQ to `DataSet` method.
- C. All faculty members and three query methods are added into the `ComboName` and `ComboMethod` comboboxes, respectively. The first item of each of them has selected the default one by setting the `SelectedIndex` to 0.

The next coding is for the Select button Click method. After the user selected the desired data query method from the Method combobox and the faculty member from

the Faculty Name combobox, the Select button is used to trigger its Click method to retrieve all courses taught by the selected faculty.

Note that two queries are needed to perform this query because there is no faculty name column available in the Course table. Thus we must first make a query to the Faculty table to find the faculty_id that is related to the faculty name selected by the user from the Faculty Name combobox in the Course form. Then we need to make the second query, which is used to find all course_id and related courses taught by the selected faculty from the Course table. All queried course_id are displayed in the CourseList box, and the detailed course information for each course can be displayed in five textboxes when the user clicks on the associated course_id from the CourseList box.

In order to save time and space, we have two ways to simplify these queries: One way is to use the joined table query we discussed in Section 5.19.2.5, and the second way is to combine these two queries into one stored procedure and call that stored procedure to perform these two queries. We have already discussed how to use the stored procedures in the SQL Server database environment to perform the data query operations with the Student table in Section 5.19.2.7.3. In this section, we want to use the stored procedure in the Oracle database environment to perform these queries. You need to note that the stored procedures in the SQL Server database and the Oracle database are different. Thus in the following section we need first to provide a discussion about these two different kinds of stored procedures in two database environments.

5.20.3.5 Stored Procedures in Oracle Database Environment

Different database vendors provide different tools to support the developments and implementations of stored procedures. For the SQL Server 2005, Microsoft provides the SQL Server Management Studio and the SQL Server Management Studio Express. For the Oracle database, Oracle provides Oracle Database 10g and Oracle Database 10g Express Edition. Different methods can be used to create stored procedures; for example, six methods are shown in Section 5.19.2.7.1 to create stored procedures for the SQL Server database. Similarly, Oracle also provides many methods to create stored procedures. For example, one can use the Object Browser page or SQL Commands page in the Oracle Database 10g Express Edition to create stored procedures.

Another important point one needs to understand is that the stored procedures are categorized based on the query type or SQL statement type used by the stored procedure in the Oracle database. In SQL Server 2005, there is no difference between stored procedures using the SELECT, INSERT, UPDATE, or DELETE statements, and all of these statements can be integrated into a standard stored procedure. But in the Oracle database, it is different. If a stored procedure needs to return queried data—a SELECT statement is embedded in that stored procedure. That stored procedure must be embedded into a package. The package in Oracle is a class and it can contain variables, functions, and procedures. Therefore, the stored procedures in the Oracle must be divided into two parts: stored procedures and packages. The stored procedures that don't need to return any data (by executing the INSERT, UPDATE, and DELETE statements) can be considered as a pure stored procedure, and the stored procedures that need to return data (by executing the SELECT statement) must be embedded into the package and therefore a package should be used.

```

CREATE OR REPLACE PROCEDURE Procedure's name
{
    Param1's name    Param1's data type,
    Param2's name    Param2's data type,
    .....
}
AS
BEGIN
    (Your SQL Statements, such as INSERT, UPDATE or DELETE);
END;

```

Figure 5.171 Syntax of creating a stored procedure in Oracle.

```

CREATE OR REPLACE PROCEDURE InsertProcedure
{
    studentId  VARCHAR2,
    name       CHAR,
    credit     NUMBER
}
AS
BEGIN
    INSERT INTO Student(student_id, s_name, s_credit)
    VALUES(studentId, name, credit, SYSDATE);
END;

```

Figure 5.172 Sample of the stored procedure in Oracle.

5.20.3.6 Syntax of Creating a Stored Procedure in Oracle

The syntax of creating a stored procedure in the Oracle database is shown in Figure 5.171. The keyword **REPLACE** is used for the modified stored procedures. Recall that in the SQL Server 2005, the keyword **ALTER** is used for any stored procedure that has been modified since it was created. In Oracle, the keyword **CREATE OR REPLACE** is used to represent any procedure that is either newly created or modified.

Following the procedure's name, all input or output parameters are declared inside the braces. After the keyword **AS**, the stored procedure's body is displayed. The body begins with the keyword **BEGIN** and ends with the keyword **END**. You need to note that a semicolon must be followed after each SQL statement and the keyword **END**.

An example of creating a stored procedure in Oracle is shown in Figure 5.172. The length of the data type for each parameter is not necessary since this allows those parameters to have a varying-length value.

5.20.3.7 Syntax of Creating a Package in Oracle

To create a stored procedure that returns data, one needs to embed the stored procedure into a package. The syntax of creating a package is shown in Figure 5.173.

The syntax of creating a package contains two parts: the package definition part and the package body part. The returned data type, **Cursor**, is first defined since the cursor can be used to return a group of data. Following the definition of the cursor, the stored procedure, that is, the protocol of the stored procedure, is declared with the input and

```

CREATE OR REPLACE PACKAGE Package's name
AS
    Definition for the returned Cursor;
    Definition for the stored procedure
END;
CREATE OR REPLACE PACKAGE BODY Package's name
AS
    Stored procedure prototype
AS
BEGIN
    OPEN Returned cursor FOR
    (Your SQL SELECT Statements);
END;
END;

```

Figure 5.173 Syntax of creating a package in Oracle.

```

CREATE OR REPLACE PACKAGE FacultyPackage
AS
    TYPE CURSOR_TYPE IS REF CURSOR;
    PROCEDURE SelectFacultyID (FacultyName IN CHAR,
                               FacultyID OUT CURSOR_TYPE);
END;
CREATE OR REPLACE PACKAGE BODY FacultyPackage
AS
    PROCEDURE SelectFacultyID (FacultyName IN CHAR,
                               FacultyID OUT CURSOR_TYPE)
AS
BEGIN
    OPEN FacultyID FOR
    SELECT faculty_id, title, office, email FROM Faculty
    WHERE name = FacultyName;
END;
END;

```

Figure 5.174 Example of creating a Faculty Package in Oracle.

the output parameters (cursor is the output argument). Following the package definition part is the body part. The protocol of the stored procedure is redeclared at the beginning, and then the body begins with the opening of the cursor and assigns the returning result of the following SELECT statement to the cursor. Similarly, each statement must end with a semicolon, including the keyword END.

An example of creating a FacultyPackage in Oracle is shown in Figure 5.174. The stored procedure is named SelectFacultyID with two parameters: the input parameter FacultyName and the output parameter FacultyID. The keywords IN and OUT follow the associated parameter to indicate the input/output direction of the parameter. The length of the stored procedure name is limited to 30 letters in Oracle. Unlike the stored procedure name created in SQL Server 2005, there is no prefix applied for each procedure's name.

Fine. We have discussed and understood the syntax and structure of stored procedures and packages developed in the Oracle environment. Now let's return to our project and our CourseForm. As we mentioned, we want to combine two queries into a stored

procedure or package to get all courses (`course_id`) taught by the selected faculty: The first query is used to get the `faculty_id` from the Faculty table based on the selected faculty name, and the second query is to get all courses taught by the selected faculty based on the `faculty_id` from the Course table. Since there is no faculty name available in the Course table, we have to perform two queries.

Many different ways can be used to create packages, such as using the Object Browser page or the SQL Commands page in the Oracle Database 10g Express Edition (XE). In this application, we prefer to use the Object Browser page to do this job since it provides a graphic user interface.

Unlike the SQL Server database, Visual Studio.NET 2008 does not provide a graphic user interface to help users to directly create, edit, and manipulate the Oracle database components such as tables, views, and stored procedures inside the Visual Studio.NET environment. The Oracle Database 10g Express Edition or Oracle Database 10g XE did provide an Oracle Development Tools (ODT) for .NET to allow users to create and manipulate database components such as tables, views, indexes, stored procedures, and packages directly inside the Visual Studio.NET environment by using an Oracle Explorer that is similar to the Server Explorer for the SQL Server database. To use this tool, one needs to install the Oracle Developer Tools for Visual Studio.NET.

In this section, we will use the Object Browser page provided by Oracle Database 10g XE to create our packages without installing and using the ODT.

5.20.3.8 Create Faculty_Course Package for Course Form

Open the Oracle Database 10g XE home page by going to Start|All Programs|Oracle Database 10g Express Edition|Go To Database Home Page items. Login as a user by entering `CSE_DEPT` into the Username box and the password reback into the Password box. Click on the Object Browser and select Create|Package item to open the Create Package window, which is shown in Figure 5.175.

Each package has two parts: the definition or specification part and the body part. First, let's create the specification part by checking the Specification radio button and click on the Next button to open the Name page, which is shown in Figure 5.176.

Enter the package name `Faculty_Course` into the name box and click on the Next button to go to the specification page, which is shown in Figure 5.177.

A default package specification prototype, which includes a procedure and a function, is provided on this page, and you need to use your real specifications to replace those default items. Since we don't need any function for our application, so remove the default function prototype, and change the default procedure's name from the `test` to our procedure name—`SelectFacultyCourse`. Your finished coding for the specification page should match that shown in Figure 5.178.

The coding language used in this section is called Procedural Language Extension for SQL, or PL-SQL, which is a popular language and widely used in Oracle database programming.

In line 2, we defined the returned data type as a `CURSOR_TYPE` by using:

```
TYPE CURSOR_TYPE IS REF CURSOR;
```

since you must use a cursor to return a group of data and the IS operator is equivalent to an equal operator.

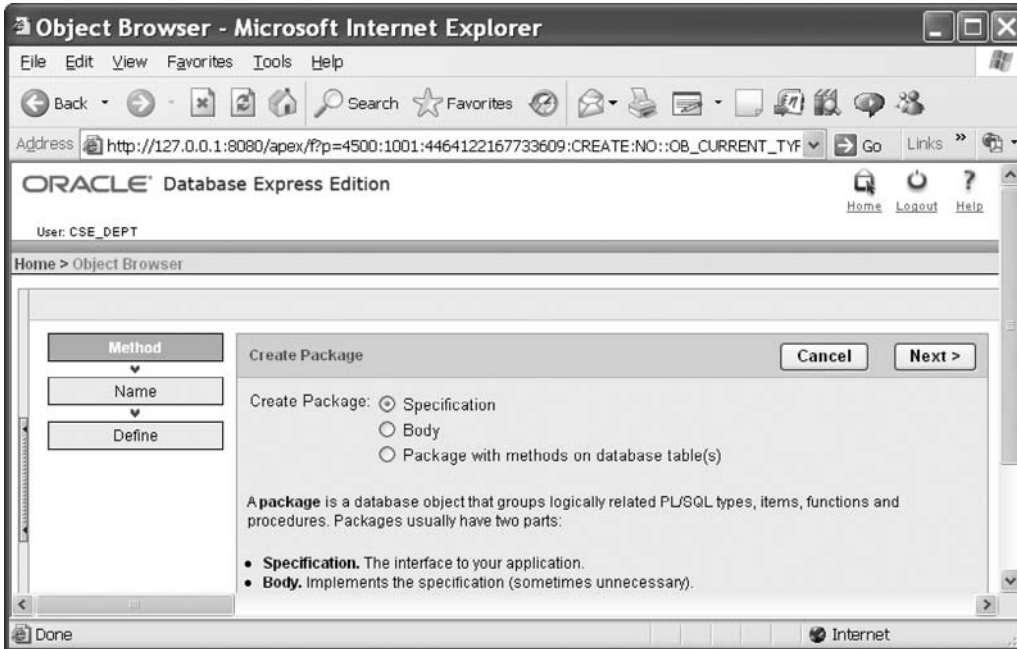


Figure 5.175 Opened Create Package window.

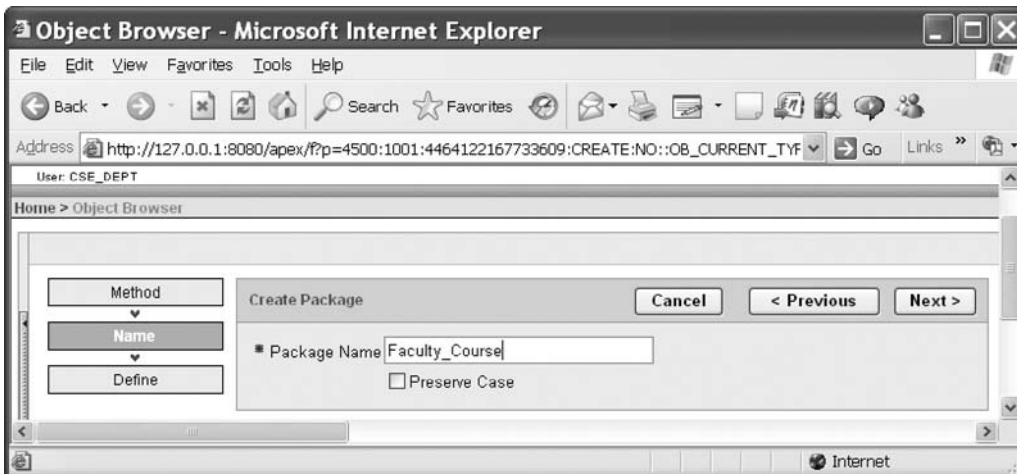


Figure 5.176 Name page of the Package window.

The prototype of the procedure `SelectFacultyCourse()` is declared in line 3. Two arguments are used for this procedure: input parameter `FacultyName`, which is indicated as an input by using the keyword `IN` followed by the data type of `VARCHAR2`. The output parameter is a cursor named `FacultyCourse` followed by the keyword `OUT`. Each PL-SQL statement must end with a semicolon, and this rule also applies to the `END` statement.

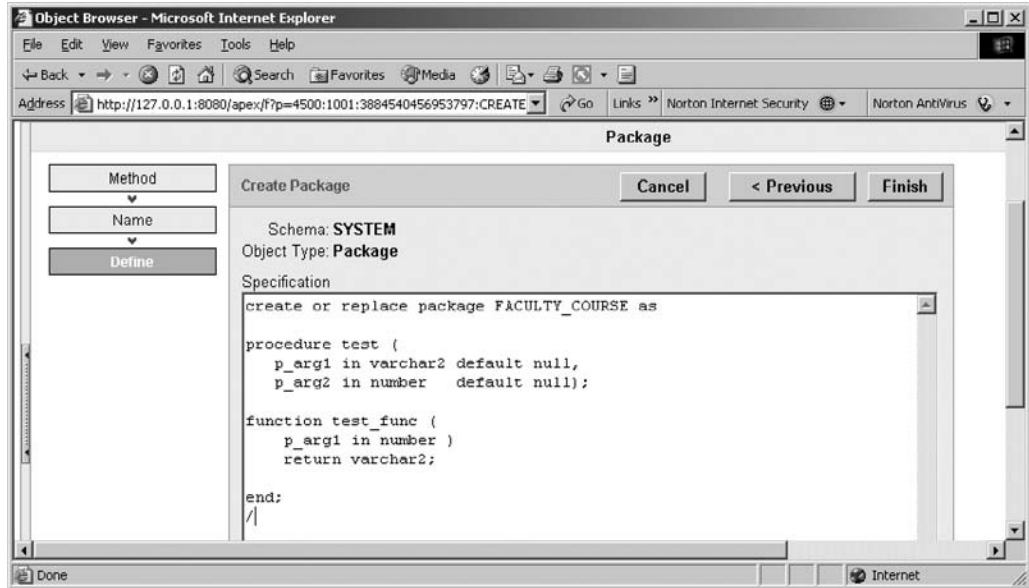


Figure 5.177 Opened Specification page.

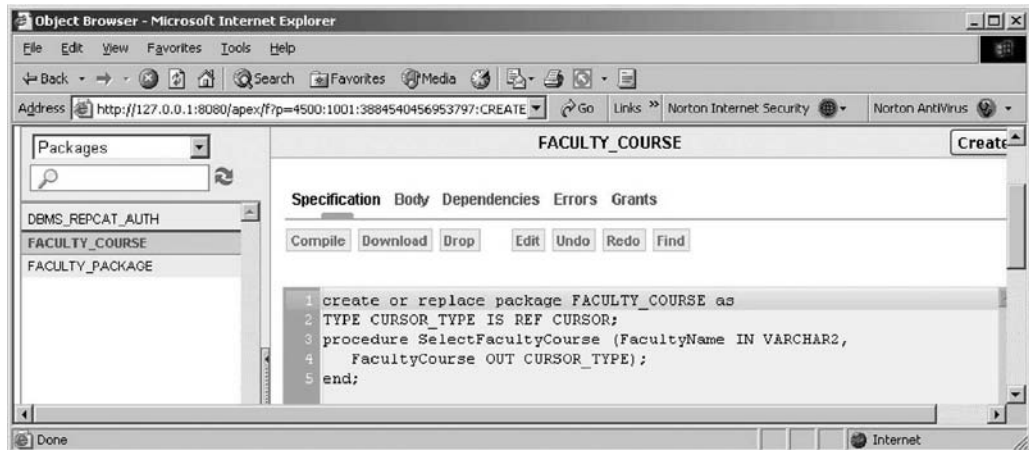


Figure 5.178 Coding for the Specification page.

Click on the Finish button to complete this step. You can click on the Compile button to compile this specification block if you like. Next we need to create the body block of this package. Click on the Body tab to open the Body page, which is shown in Figure 5.179.

Click on the Edit button to begin to create our body part. Enter the PL-SQL codes into this body shown in Figure 5.180.

The procedure prototype is redeclared in line 2. But an IS operator is attached at the end of this prototype and it is used to replace the AS operator to indicate that this proce-

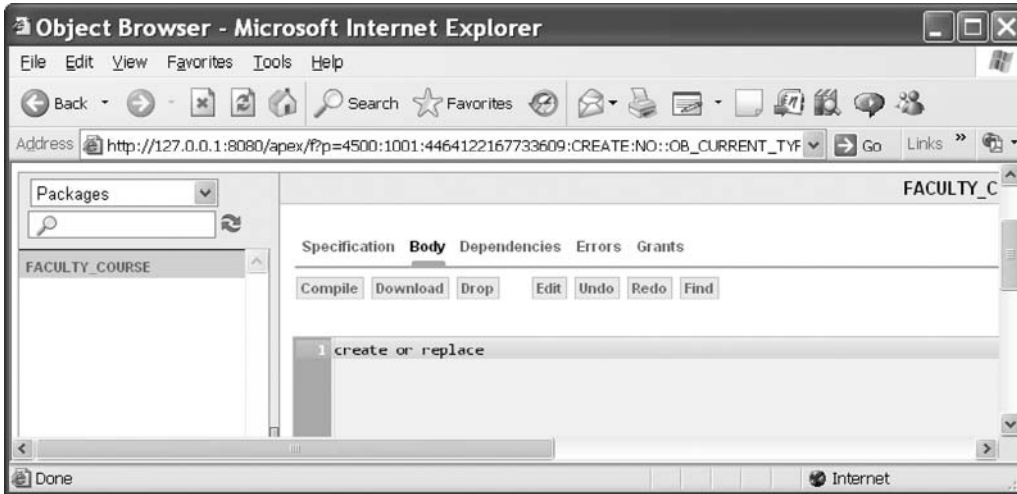


Figure 5.179 Opened Body page of the package.

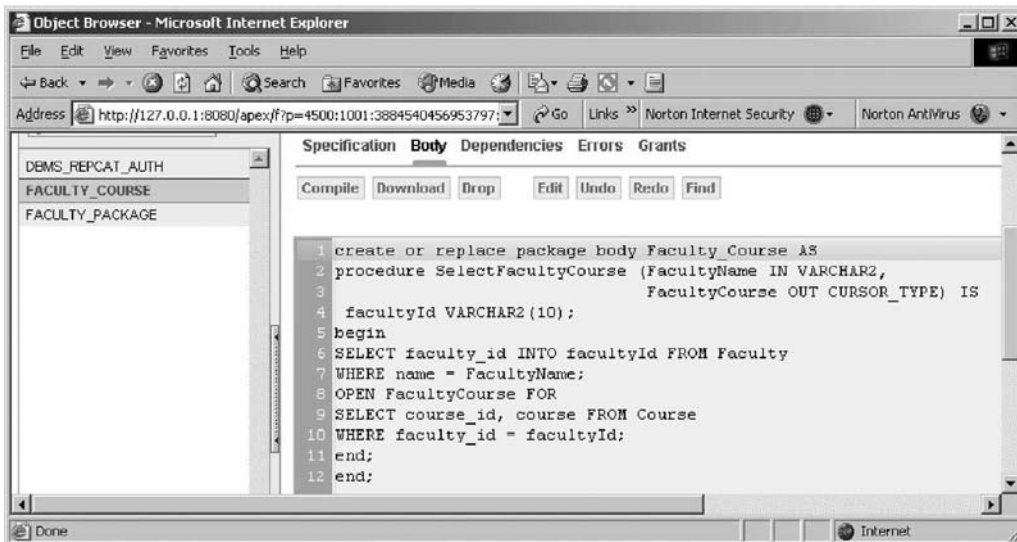


Figure 5.180 Coding for the Body part of the package.

procedure needs to use a local variable `facultyId`, and this variable will work as an intermediate variable to hold the returned `faculty_id` from the first query, which is located at line 6.

Starting from `BEGIN`, our real SQL statements are included in lines 6 and 7. The first query is to get the `faculty_id` from the `Faculty` table based on the input parameter `FacultyName`, which is the first argument of this procedure. A `SELECT ... INTO` statement is utilized to temporarily store the returned `faculty_id` into the intermediate variable `facultyId`.

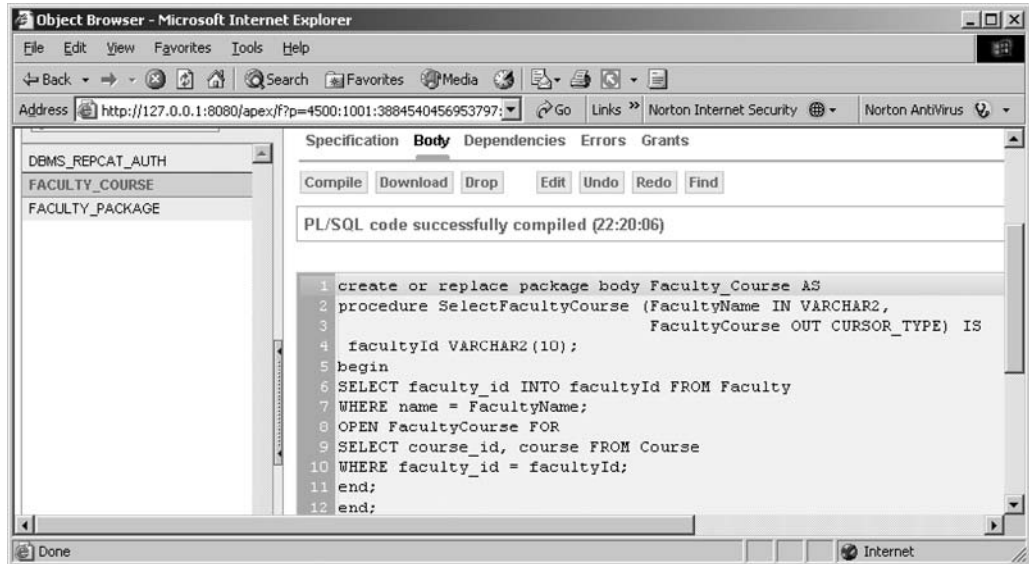


Figure 5.181 Compiled coding for the body part of the package.

The OPEN FacultyCourse FOR command is used to assign the returned data columns from the following query to the cursor variable FacultyCourse. Recall that we used a SET command to perform this assignment functionality in the SQL Server stored procedure in Section 5.19.2.7.4. Starting from lines 9 and 10, the second query is declared, and it is to get all course_id and courses taught by the selected faculty from the Course table based on the intermediate variable's value, faculty_id, which is obtained from the first query above. The queried results are assigned to the cursor variable FacultyCourse.

Now let's compile our package by clicking on the Compile button. A successful compiling information

PL/SQL code successfully compiled (22:20:06)

will be displayed if this package is bug free, which is shown in Figure 5.181.

The development of our Oracle package is complete, and now let's go to the Visual Studio.NET to call this package to perform our course query for our Course form.

5.20.3.9 Query Data Using Oracle Package for Course Form

Open the Course form window and double-click on the Select button to open its Click method and enter the codes shown in Figure 5.182 into this method.

Let's take a look at this piece of code to see how it works.

- A. The package query string is declared, and this string contains both the package's name (Faculty_Course) and the stored procedure's name (Select FacultyCourse). All query strings for the Oracle database package must follow this style. The package's name and the procedure's name defined in this string must be identical with those names we used when we created this package in the Object Browser in Oracle Database 10g XE. Otherwise your calling to this package would fail.

OracleSelectRTOject.CourseForm		cmdSelect_Click()	
		private void cmdSelect_Click(object sender, EventArgs e)	
		{	
A		string cmdString = "Faculty_Course.SelectFacultyCourse";	
B		OracleParameter paramFacultyName = new OracleParameter();	
		OracleParameter paramFacultyCourse = new OracleParameter();	
		OracleDataAdapter CourseDataAdapter = new OracleDataAdapter();	
		OracleCommand oraCommand = new OracleCommand();	
		DataTable oraDataTable = new DataTable();	
		OracleDataReader oraDataReader;	
		LogInForm logForm = new LogInForm();	
		logForm = logForm.getLogInForm();	
C		paramFacultyName.ParameterName = "FacultyName";	
		paramFacultyName.OracleType = OracleType.VarChar;	
		paramFacultyName.Value = ComboName.Text;	
D		paramFacultyCourse.ParameterName = "FacultyCourse";	
		paramFacultyCourse.OracleType = OracleType.Cursor;	
		paramFacultyCourse.Direction = ParameterDirection.Output;	
E		oraCommand.Connection = logForm.oraConnection;	
		oraCommand.CommandType = CommandType.StoredProcedure;	
F		oraCommand.CommandText = cmdString;	
		oraCommand.Parameters.Add(paramFacultyName);	
G		oraCommand.Parameters.Add(paramFacultyCourse);	
		if (ComboMethod.Text == "DataAdapter Method")	
		{	
		CourseDataAdapter.SelectCommand = oraCommand;	
H		CourseDataAdapter.Fill(oraDataTable);	
		if (oraDataTable.Rows.Count > 0)	
		FillCourseTable(oraDataTable);	
		else	
I		MessageBox.Show("No matched course found!");	
		oraDataTable.Dispose();	
		CourseDataAdapter.Dispose();	
		}	
J		else if (ComboMethod.Text == "DataReader Method")	
		{	
		oraDataReader = oraCommand.ExecuteReader();	
		if (oraDataReader.HasRows == true)	
		FillCourseReader(oraDataReader);	
		else	
K		MessageBox.Show("No matched course found!");	
		oraDataReader.Close();	
		}	
		else //LINQ to DataSet Method is selected...	
L		{	
		CourseDataAdapter.SelectCommand = oraCommand;	
M		CourseDataAdapter.Fill(ds, "Course");	
		var courseinfo = (from ci in ds.Tables["Course"].AsEnumerable()	
		select ci);	
N		CourseList.Items.Clear();	
O		foreach (var cRow in courseinfo)	
		{	
		CourseList.Items.Add(cRow.Field<string>("course_id"));	
		}	
P		ds.Clear();	
		}	
Q		CourseList.SelectedIndex = 0;	
		}	

Figure 5.182 Coding for the Select button Click method.

- B.** All data components used to perform this query are declared and created here. First, two Oracle Parameter objects are created: paramFacultyName and paramFacultyCourse. These two Parameter objects will be passed into the calling package, and they work as input and output parameters, respectively. Some other components, such as the DataAdapter, Command, DataTable, and Data Reader, are also created here.
- C.** The first Parameter object is initialized here. Both parameter name, FacultyName, and the data type, VARCHAR, must be identical with the name and the data type we used when we created this procedure in Oracle Database 10g XE R2. The parameter's value should be equal to the selected name from the faculty name combobox (ComboName.Text) in the Course form window in Visual C#.NET.
- D.** The second parameter is also initialized with the associated parameter name and data type. One important point is that the second parameter is an output parameter and its data type is cursor, and the transmission direction of this parameter is output. So the Direction property of this Parameter object must be clearly indicated by assigning the Output to it. Otherwise, the procedure calling may encounter some error and this error is hard to debug.
- E.** The Command object is initialized by assigning the associated property, such as the Connection, CommandType, and CommandText. The CommandType must be StoredProcedure and the CommandText should be the query string we declared at the beginning of this method (step **A**).
- F.** Two initialized Parameter objects are added into the Command object, that is, they are added into the Parameters collection property of the Command object.
- G.** If the user selected the DataAdapter method, the initialized Command object is assigned to the SelectCommand property of the DataAdapter, and the Fill() method is executed to fill the Course table. Basically, only at this moment, the Oracle package we developed is called and two queries are executed. The returned columns should be stored in the Course data table if this fill is successful.
- H.** If the Count property of the Course table is greater than 0, which means that at least one row is filled into the table, the user-defined method FillCourseTable() is called to fill the queried course_id into the CourseList box in the Course form window. Otherwise, an error message is displayed to indicate that this fill has failed.
- I.** Some cleaning jobs are performed to release some data objects used for this query.
- J.** If the user selected the DataReader method, the ExecuteReader() method is executed to invoke the DataReader to retrieve required columns and store the returned results into the DataReader. If the property HasRows is true, which means that DataReader did read back some rows, the user-defined method FillCourseReader() is called to fill the CourseList box in the Course form window with the read rows. Otherwise, an error message is displayed.
- K.** Another cleaning job is performed to release the DataReader used for this query.
- L.** If the user selected the LINQ to DataSet method, the initialized Command object is assigned to the SelectCommand property of the DataAdapter, and the Fill() method is executed to initialize the DataSet with the stored procedure. This operation will not only fill the Course table but also the Faculty table since there are two query operations in the package we built above.
- M.** A typical LINQ query structure is created and executed to retrieve back all course_id from the Course table. The courseinfo is a Visual C# 2008 implicitly typed local variable with a data type var. The Visual C# 2008 will be able to automatically convert this var to any suitable data type. In this case, it is a collection, when it sees it. An iteration variable ci is used to iterate over the result of this query from the Course table. Then a similar SQL

SELECT statement is executed without the WHERE clause. The reason for that is because we have built the Command object `oraCommand` by assigning both the input and output parameters to the Parameters collection. Therefore, we do not need to use any other query criterion here. Another key point for this structure is the operator `AsEnumerable()`. Since different database systems use different collections and query operators, those collections must be converted to a type of `IEnumerable<T>` in order to use the LINQ technique because all data operations in LINQ use Standard Query Operators that can perform complex data queries on an `IEnumerable<T>` sequence. A compiling error would be encountered without this operator.

- N. The `CourseList` box is cleaned up to make it ready to add and display all `course_id` for the selected faculty. This code is important and necessary. Without this line, duplicated `course_id` will be added and displayed in this `CourseList` box.
- O. A `foreach` loop is utilized to pick up each `course_id` from the selected result `cRow`, which is obtained from the `courseinfo` we get from the LINQ query. Then, add each `course_id` into the `CourseList` box in the `CourseForm` window to display them. Since we are using a nontyped `DataSet`, we must indicate the column clearly with the `field<string>` and the column's name as the position for each of them.
- P. The `DataSet` is cleaned up after this query. This operation is important and necessary. Without this line, multiple duplicated `course_ids` would be added and displayed in the `CourseList` box since multiple duplicated query results are filled into the `DataSet`, that is, into the `Course` table.
- Q. This statement is very important and it is used to select the first `course_id` in the `CourseList` box as the default course as the `Course` form is opened. More important, this command can work as a trigger event to trigger the `CourseList` box's `SelectedIndexChanged` method to display the detailed information related to that default `course_id`. The coding for that method is our next job.

The coding for the `FillCourseTable()` and the `Back` button `Click` method have nothing to do with any object used in this project. Thus, no coding modification is needed. The user-defined method `FillCourseReader()` needs only one small modification, which is to change the nominal argument's type to `OracleDataReader` since now we are using an Oracle data provider. The detailed explanations for methods `FillCourseTable()` and `FillCourseReader()` can be found in Figure 5.92. For your convenience, we list this piece of code with some explanations again, which is shown in Figure 5.183.

Let's see how this piece of code works.

- A. Before we can fill the `CourseList` box, a cleaning job is needed. This cleaning is very important. Otherwise multiple repeat `course_ids` would be added and displayed in this listbox if you forget to clean it up first.
- B. A `foreach` loop is used to scan all rows of the filled `Course` table. Recall that we filled 6 columns of data from the `Course` table in the database to this `Course` table in the `DataTable` object, starting with the first column—`course_id`—and the second column—`course`. Now we need to pick up the first column—`course_id` (column index = 0)—for each returned row or record of the `Course` table. Then the `Add()` method of the `ListBox` control is used to add each retrieved `course_id` into the `CourseList` box.
- C. For the `FillCourseReader()` method, the data type of the passed argument is `OracleDataReader` since we are using an Oracle database as our data source.
- D. A local string variable `strCourse` is created, and this variable can be considered as an intermediate variable used to temporarily hold the queried data from the `Course` table.

```

OracleSelectRTOject.CourseForm
FillCourseTable()
private void FillCourseTable(DataTable CourseTable)
{
A   CourseList.Items.Clear();
B   foreach (DataRow row in CourseTable.Rows)
    {
        CourseList.Items.Add(row[0]);    //the 1st column is course_id - cmdString
    }
}
C   private void FillCourseReader(OracleDataReader CourseReader)
    {
D   string strCourse = string.Empty;
E   CourseList.Items.Clear();
F   while (CourseReader.Read())
    {
        strCourse = CourseReader.GetString(0); //the 1st column is course_id
        CourseList.Items.Add(strCourse);
    }
}

```

Figure 5.183 Coding for the FillCourseTable and FillCourseReader methods.

- E.** Similarly, we need to clean up the CourseList box before it can be filled.
- F.** A while loop is utilized to retrieve each first column's data [GetString(0)] whose column index is 0 and the data value is the course_id. The queried data first is assigned to the intermediate variable strCourse, and then it is added into the CourseList box by using the Add() method.

Next we need to take care of the coding for the CourseList_SelectedIndexChanged() method. The functionality of the coding for this method is to display the detailed course information, such as the course title, course credit, classroom, course schedule, and enrollment, for the selected course_id by the user when the user clicks on a course_id from the CourseList box. Five textbox controls in the Course form are used to store and display the detailed course information.

Now let's begin to do our coding for this method. Open the Course form window and double-click on the CourseList box (any place inside that list box) to open this method. Enter the codes shown in Figure 5.184 into this method. Only the DataAdapter query method is used for this method. Let's take a look at this piece of code and see how it works.

- A.** The query string is first declared, and we need to retrieve six columns from the Course table. In fact, we have already gotten the course_id from the last query in the Select button method. However, in order to keep the code neat, we still retrieve this column from this query. A nominal parameter courseid that works as a dynamic parameter is assigned to the course_id column as our query criterion. You need to note that the assignment operator for the dynamic parameter in Oracle is an equal operator plus a colon.
- B.** All data components used to perform this query are declared here, such as the DataAdapter, Command object, and the DataTable objects. The keyword Oracle needs to be prefixed before these objects since we are using the Oracle data components to perform this query. The new instance of the LogInForm class is used to allow us to access and use the global connection object defined in that class.
- C.** The Command object is initialized by assigning it with associated properties such as the Connection, CommandType, and CommandText.

```

OracleSelectRTOject.CourseForm
CourseList_SelectedIndexChanged()

private void CourseList_SelectedIndexChanged(object sender, EventArgs e)
{
A   string cmdString = "SELECT course, credit, classroom, schedule, enrollment, course_id FROM Course ";
B   cmdString += "WHERE course_id =:courseid";
   OracleDataAdapter CourseDataAdapter = new OracleDataAdapter();
   OracleCommand oraCommand = new OracleCommand();
   DataTable oraDataTable = new DataTable();
   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();

C   oraCommand.Connection = logForm.oraConnection;
   oraCommand.CommandType = CommandType.Text;
   oraCommand.CommandText = cmdString;
D   oraCommand.Parameters.Add("courseid", OracleType.Char).Value = CourseList.SelectedItem;
E   CourseDataAdapter.SelectCommand = oraCommand;
   CourseDataAdapter.Fill(oraDataTable);
F   if (oraDataTable.Rows.Count > 0)
       FillCourseTextBox(oraDataTable);
   else
G   MessageBox.Show("No matched course information found!");
   oraDataTable.Dispose();
   oraCommand.Dispose();
   CourseDataAdapter.Dispose();
}

```

Figure 5.184 Coding for the SelectedIndexChanged method.

- D.** The dynamic parameter `courseid` is added into the `Parameters` collection, which is a property of the `Command` object using the `Add()` method. The real value of this parameter is the `course_id` that is selected by the user from the `CourseList` box.
- E.** The `SelectCommand` property of the `DataAdapter` is assigned with the initialized `Command` object, and the `Fill()` method is executed to fill the course table.
- F.** If the `Count` property of the returned data table is greater than 0, which means that at least one row is filled into the table, the `FillCourseTextBox()` method is called to fill five textbox controls with retrieved six columns. Otherwise, an error message is displayed if this fill has failed.
- G.** Some cleaning job is performed here to release all objects used for this fill table operation.

Two user-defined methods, `FillCourseTextBox()` and `MapCourseTable()`, have no relationship with any object used in this project. Therefore, no coding modification is needed for them. For the detailed line-by-line explanations of these two methods, refer to Figure 5.94 in section 5.18.4.

The coding for the `Back` button `Click` method is easy. Open this method and enter `this.Hide();` into this method. That's all!

Before we can run the project to test our codes, we need to copy all faculty image files to the folder in which our Visual C# executable file is located. In this application, it is the `Debug` folder of the project. In our case, this folder is located at `C:\Book 6\Chapter 5\OracleSelectRTOject\bin\Debug`.

Now let's start this project to test the codes we developed for the `Course` form. Click on the `Debug\Start Debugging` button to run the project. Enter the suitable username and password, such as **jhenry** and **test** for the `LogIn` form, and then select the `Course Information` item from the `Selection` form window to open the `Course` form. Select the desired faculty

Figure 5.185 Running status of the Course form.

name from the Faculty Name combobox and click on the Select button to list all courses taught by this faculty in the CourseList box. Then click each course_id item from the CourseList box, and the detailed course information related to the selected course_id will be displayed in five textbox controls in this form, which is shown in Figure 5.185.

You can try to perform this query by using different methods with different faculty members. Our coding is successful!

At this point we finished all coding for this project. As for the coding for the Student form, we prefer to leave this job to students as their homework. A complete project named XEOracleSelectRTOObject can be found in the folder DBProjects\Chapter 5 that is located at the accompanying ftp site (see Chapter 1).

5.21 CHAPTER SUMMARY

The main topic of this chapter is to develop professional data-driven applications in the Visual C#.NET 2008 environment by using different methods. The data query is the main topic of this chapter.

The first method is to utilize Design Tools and Wizards provided by Visual Studio.NET 2008 and ADO.NET to build simple but powerful data query projects. The second method is to use the runtime objects method to build the portable data query projects. The third method is to use LINQ to DataSet and LINQ to SQL to simplify the data query and improve query efficiency.

Comparably, the first method is simple, and it is easy to be understood and learned by those students who are beginners to Visual C# and databases. This method utilizes many powerful tools and wizards provided by Visual Studio.NET 2008 and ADO.NET to simplify the coding process, and most of codes are autogenerated by the .NET Framework 3.5 and Visual C#.NET 2008 as the user uses those tools and wizards to perform data operations such as adding new data source, making data binding, and con-

necting to the selected data source. The shortcoming of this method is that a lot of coding jobs are performed by the system behind the screen. Thus it is difficult to give users a clear picture of what is really happening behind those tools and wizards. Most codes are generated by the system automatically in the specific locations. Thus it is not easy to translate and execute those codes in other platforms.

The second method, the runtime objects, allows users to dynamically create all data-related objects and perform the associated data operations after the project runs. Because all objects are generated by the coding, it is very easy to translate and execute this kind of project in other platforms. This method provides a clear view for the users and enables users to have a global and detail picture of how to control the direction of the project with the coding according to the users' idea and feeling. The shortcoming of this method is that a lot of coding may make the project complicated and it is hard to be accepted by the beginners.

The third method, LINQ to DataSet and LINQ to SQL, is an up-to-date method, and this technique was released with the Microsoft Visual Studio.NET 2008. The coding process can be significantly simplified, and the query efficiency can be greatly improved by using this technique. These advantages can be found by comparing the codes we developed in some projects in this chapter.

Three kinds of databases are discussed in this chapter: Microsoft Access, SQL Server, and Oracle. Each database is explained in detail with a real sample project. Each project uses three different data query methods: DataAdapter method, runtime objects method, and LINQ to DataSet or LINQ to SQL. Line-by-line illustrations are provided for each sample project. The readers can obtain the solid knowledge and practical experience in how to develop a professional data query application after they finish this chapter.

By finishing Part I in this chapter, you should be able to:

- Use the tools and wizards provided by Visual Studio.NET 2008 and ADO.NET to develop simple but powerful data-driven applications to perform data queries on Microsoft Access 2007, SQL Server 2005, and Oracle databases.
- Use the OleDbConnection, SqlConnection, or OracleConnection class to connect to Microsoft Access 2007, SQL Server 2005 Express, and Oracle 10g XE databases.
- Perform data binding to a DataGridView using two methods.
- Use the OleDbCommand, SqlCommand, and OracleCommand class to execute the data query with dynamic parameters to three kinds of databases.
- Use the OleDbDataAdapter to fill a DataSet and a DataTable object with three kinds of databases.
- Use the OleDbDataReader class to query and process data with three kinds of databases.
- Use LINQ to DataSet to simplify the data query process and improve the data query efficiency.
- Set properties for the OleDbCommand objects to construct a desired query string for three kinds of databases.

By finishing Part II in this chapter, you should be able to:

- Use the runtime objects to develop the professional data-driven applications to perform data queries on Microsoft Access 2007, SQL Server 2005, and Oracle databases.

- Use the `OleDbConnection`, `SqlConnection`, and `OracleConnection` class to dynamically connect to Microsoft Access 2007, SQL Server 2005 Express, and Oracle 10g XE databases.
- Use the `OleDbCommand`, `SqlCommand`, and `OracleCommand` class to dynamically execute the data query with dynamic parameters to three kinds of databases.
- Use the `OleDbDataAdapter`, `SqlDataAdapter`, and `OracleDataAdapter` to dynamically fill a `DataSet` and a `DataTable` object with three kinds of databases.
- Use the `OleDbDataReader`, `SqlDataReader`, and `OracleDataReader` class to dynamically read and process data with three kinds of databases.
- Use LINQ to `DataSet` and LINQ to SQL to significantly simplify the query process and improve the query efficiency.
- Set properties for the `OleDbCommand`, `SqlCommand`, and `OracleCommand` objects dynamically to construct a desired query string for three kinds of databases.
- Use the Server Explorer to create, debug, and test stored procedures in Visual Studio.NET environment.
- Use the SQL stored procedure to perform the data query from Visual C#.NET.
- Use the SQL nested stored procedure to perform the data query from Visual C#.NET.
- Use the Object Browser in Oracle Database 10g XE to create, debug, and test stored procedures and packages.
- Use the Oracle stored procedures and packages to perform the data query from Visual C#.NET.

In Chapter 6, we will discuss the data-inserting technique with three kinds of databases. Three methods are introduced in two parts: Part I: Using the Design Tools and Wizards provided by Visual Studio.NET 2008 to develop data inserting query, and Part II: Using the runtime objects, LINQ to `DataSet` and LINQ to SQL to perform the data inserting job for three databases.

HOMEWORK

I. True/False Selections

- ___ 1. Data Provider-dependent objects are `Connection`, `Command`, `TableAdapter`, and `DataReader`.
- ___ 2. LINQ to `DataSet` can be used to access any kind of databases.
- ___ 3. To move data between the bound controls on a form window and the associated columns in the data source, a `BindingSource` is needed.
- ___ 4. To set up the connection between the bound controls on a form window and the associated columns in the data source, a `TableAdapter` is needed.
- ___ 5. All `TableAdapter` classes are located in the namespace `DataSetTableAdapters`.
- ___ 6. Running the `Fill()` method is equivalent to executing the `Command` object.
- ___ 7. The `DataSet` can be considered as a container that contains multiple data tables, but those tables are only a mapping of the real data tables in the database.
- ___ 8. To run the `Fill()` method to fill a table is like filling a data table that is located in the `DataSet`, not a real data table in the database.

- ___9. By checking the Count property of a data table, one can determine whether a fill-table operation is successful or not.
- ___10. The DataTable object is a Data Provider-independent object.
- ___11. If one needs to include the SELECT statements in an Oracle stored procedure, one can directly create a stored procedure and call it from Visual C#.NET.
- ___12. The Cursor must be used as an output variable if one wants to return multiple columns from a query developed in a Package in the Oracle database.
- ___13. You can directly create, edit, manipulate, and test stored procedures for the SQL Server database inside the Visual Studio.NET environment.
- ___14. To call an SQL Server stored procedure, one must set the CommandType property of the Command object to Procedure.
- ___15. To set up a dynamic parameter in a SELECT statement in the SQL Server database, an @ symbol must be prefixed before the nominal variable.
- ___16. To access different databases by using the LINQ technique, different LINQ APIs must be used. For example, to access the SQL Server database, LINQ to SQL is used, to access the Oracle database, LINQ to Oracle must be used.
- ___17. To assign a dynamic parameter in a SELECT statement in the SQL Server database, the keyword LIKE must be used as the assignment operator.
- ___18. Two popular tools to create Oracle Packages are the Object Browser page and the SQL Command page in Oracle Database 10g XE.
- ___19. Two popular ways to query data from any database are using the Fill() method, which belongs to the TableAdapter class, or calling the ExecuteReader method, which belongs to the Command class.
- ___20. A DataTable can be considered as a collection of DataRowCollection and DataColumnCollection, and the latter contains DataRow and DataColumn objects.

II. Multiple Choices

1. To connect a database dynamically, one needs to use the _____.
 - a. Data Source
 - b. TableAdapter
 - c. Runtime object
 - d. Tools and Wizards
2. Four popular data providers are _____.
 - a. ODBC, DB2, JDBC, and SQL
 - b. SQL, ODBC, DB2, and Oracle
 - c. ODBC, OLEDB, SQL, and Oracle
 - d. Oracle, OLEDB, SQL, and DB2
3. To modify the DataSet, one needs to use the _____ Wizard.
 - a. DataSet configuration
 - b. DataSet edit
 - c. TableAdapter configuration
 - d. Query Builder

4. To bind a label control with the associated column in a data table, one needs to use _____.
 - a. BindingNavigator
 - b. TableAdapter
 - c. DataSet
 - d. BindingSource

5. The _____ keyword should be used as an assignment operator for the WHERE clause with a dynamic parameter for a data query in SQL Server database.
 - a. =
 - b. LIKE
 - c. :=
 - d. @=

6. The _____ data provider can be used to execute the data query for _____ data providers.
 - a. SQL Server, OleDb and Oracle
 - b. OleDb, SQL Server and Oracle
 - c. Oracle, SQL Server and OleDb
 - d. SQL Server, Odbc and Oracle

7. To perform a Fill() method to fill a data table, it executes _____ object with suitable parameters.
 - a. DataAdapter
 - b. Connection
 - c. DataReader
 - d. Command

8. To fill a list box or combobox control, one must _____ by using the _____ method.
 - a. Remove all old items, Remove()
 - b. Remove all old items, ClearBeforeFill()
 - c. Clean up all old items, CleanAll()
 - d. Clear all old items, ClearAll()

9. A _____ accessing mode should be used to define a connection object if one wants to use that connection object _____ for the whole project.
 - a. Private, locally
 - b. Protected, globally
 - c. Public, locally
 - d. Public, globally

10. To _____ data between the DataSet and the database, the _____ object should be used.
 - a. Bind, BindingSource
 - b. Add, TableAdapter
 - c. Move, TableAdapter
 - d. Remove, DataReader

11. The keyword _____ will be displayed before the procedure's name if one modified an SQL Server stored procedure.
 - a. CREATE
 - b. CREATE OR REPLACE
 - c. REPLACE
 - d. ALTER
12. To perform a runtime data query to the Oracle database, one needs to use _____.
 - a. OleDb Data Provider
 - b. Oracle Data Provider
 - c. Both (a) and (b)
 - d. None of them
13. To query data from any database using the runtime object method, two popular methods are _____ and _____.
 - a. DataSet, TableAdapter
 - b. TableAdapter, Fill
 - c. DataReader, ExecuteReader
 - d. TableAdapter, DataReader
14. To use a stored procedure to retrieve data columns from an Oracle database, one needs to create a(n) _____.
 - a. Oracle Package
 - b. Oracle stored procedure
 - c. Oracle Trigger
 - d. Oracle Index
15. Two parts exist in an Oracle Package and they are _____ and _____.
 - a. Specification, body
 - b. Definition, specifications
 - c. Body, specification
 - d. Specification, execution

III. Exercises

1. Use the tools and wizards provided by Visual Studio.NET and ADO.NET to complete the data query for the Student form in the SelectWizard project. The project file can be found in the folder DBProjects\Chapter 5 located at the accompanying ftp site (see Chapter 1).
2. Use the runtime objects to complete the data query for the Student form using the DataReader query method in the AccessSelectRTOject project. The project file can be found in the folder DBProjects\Chapter 5 located at the accompanying ftp site (see Chapter 1).
3. Develop a method by adding some codes into the cmdLogIn_Click() method of the project OracleSelectRTOject to allow users to try the login process only three times. A warning message should be displayed and the project should be exited after three times of trying to login without success.
4. Use Procedural Language Extension for SQL (PL-SQL) to create a Package in the Object Browser page of Oracle Database 10g XE. The Package contains two stored procedures; one is used to query the student_id from the Student table based on the input student name, and the second is to query all course_ids taken by the selected student from the StudentCourse table

based on the `student_id` retrieved from the first stored procedure. Compile this package after it is created to confirm that it works.

5. Try to use the `OleDb` data provider to replace either the SQL Server or the Oracle data provider for the `SQLSelectRTOject` or the `OracleSelectRTOject` project to perform the similar data query jobs for the Faculty form.
6. Use LINQ to SQL to perform the data query to Student and StudentCourse tables for the Student form in `SQLSelectRTOject` project. For your reference, a sample project can be found in the folder `DBProjects\Chapter 5` located at the accompanying ftp site (see Chapter 1). In that project, the `DataReader` method is used to perform the data query for the Student form.
7. Develop the data query for the Student form to retrieve data from both Student and StudentCourse tables using Oracle Database 10g XE. Use LINQ to DataSet to perform this query. The desired way is to use an Oracle Package to build this query.

Chapter 6

Data Inserting with Visual C#.NET

We spent a lot of time discussing and explaining data query in the last chapter by using two different methods. In this chapter, we will concentrate on inserting data into the DataSet and the database. Inserting data into the DataSet, or inserting data into the data tables embedded in the DataSet, is totally different from inserting data into the database, or inserting data into the data tables in the database. The former only inserts data into the mapping of the data table in the DataSet, and this has nothing to do with the real data tables in the database. In other words, the data inserted into the mapped data tables in the DataSet are not inserted into the data tables in the real database. The latter inserts the data into the data tables in the real database.

As you know, ADO.NET provides a disconnected working mode for the database access applications. The so-called disconnected mode means that your data-driven applications will not always keep the connection with your database, and this connection may be disconnected after you set up your DataSet and load all data from the data tables in your database into those data table mappings in your DataSet. Most of the time you are just working on the data between your applications and your data table mappings in your DataSet. The main reason of using this mode is to reduce the overhead of a large number of connections to the database and improve the efficiency of data transferring and implementations between the users' applications and the data sources.

The two parts in this chapter will show readers how to insert data into the database: inserting data into the database using the Visual Studio.NET design tools and wizards and inserting data to the database using the runtime objects method.

When finished this chapter, you will:

- Understand the working principle and structure of inserting data to the database using the Visual Studio.NET design tools and wizards.
- Understand the procedures of how to configure the TableAdapter object by using the TableAdapter Query Configuration Wizard and build the query to insert data into the database.
- Design and develop special procedures to validate data before and after data insertion.
- Understand the working principle and structure of inserting data to the database using the runtime objects method.

- Insert data into the DataSet using LINQ to DataSet and insert data into the database using LINQ to SQL queries.
- Design and build stored procedures to perform the data insertion.

To successfully complete this chapter, you need to understand topics such as fundamentals of databases, which were introduced in Chapter 2, ADO.NET, which was discussed in Chapter 3, and introduction to LINQ, which was presented in Chapter 4. Also a sample database CSE_DEPT that was developed in Chapter 2 will be used throughout this chapter.

In order to save time and avoid repetition, we will use a sample project named SampleWizards developed in the last chapter to demonstrate data insertion. Recall that some command buttons on the different form windows in that project have not been coded, such as Insert, Update, and Delete, and those buttons, or the event methods related to those buttons, will be developed and coded in this chapter. We only concentrate on the coding for the Insert button in this chapter.

PART I DATA INSERTING WITH VISUAL STUDIO.NET DESIGN TOOLS AND WIZARDS

In this part, we discuss inserting data into the database using the Visual Studio.NET design tools and wizards. We develop two methods to perform this data insertion: First, we use the TableAdapter DBDirect method, TableAdapter.Insert(), to directly insert data into the database. Second, we show readers how to insert data into the database by first adding new records into the DataSet, and then updating new records from the DataSet to the database using the TableAdapter.Update() method. Both methods utilize the TableAdapter's direct and indirect methods to complete the data insertion. The database we use is the Microsoft Access 2007 database, CSE_DEPT.accdb, which was developed in Chapter 2 and is located in the folder **Database\Access** located at the accompanying site: ftp://ftp.wiley.com/public/sci_tech_med/practical_database. Of course, you can try to use any other databases such as Microsoft SQL Server 2005 or Oracle Database 10g XE. The only issue is that you need to select and connect to the correct database when you use the Data Source window to set up your data source for your Visual C#.NET data-driven applications.

6.1 INSERT NEW DATA INTO A DATABASE

Generally, there are many different ways to insert new data into the database in Visual C#.NET. Normally, however, three methods are widely utilized:

1. Use the TableAdapter's DBDirect methods, specifically the TableAdapter.Insert() method.
2. Use the TableAdapter's Update() method to insert new records that have already been added into the DataTable in the DataSet.
3. Use the Command object combined with the ExecuteNonQuery() method.

When using method 1, one can directly access the database and execute commands such as TableAdapter.Insert(), TableAdapter.Update(), and TableAdapter.Delete() to

manipulate data in the database without requiring `DataSet` or `DataTable` objects to reconcile changes in order to send updates to a database. As we mentioned at the beginning of this chapter, inserting data into a table in the `DataSet` is different with inserting data into a table in the database. If you are using the `DataSet` to store data in your applications, you need to use the `TableAdapter.Update()` method since the `Update()` method can trigger and send all changes (updates, inserts, and deletes) to the database. A good habit is to use the `TableAdapter.Insert()` method when your application uses objects to store data (e.g., you are using textboxes to store your data) or when you want finer control over creating new records in the database.

In addition to inserting data into the database, method 2 can be used for other data operations such as updating and deleting data from the database. You can build associated command objects and assign them to the appropriate `TableAdapter`'s properties such as `UpdateCommand` and `DeleteCommand`. The point is that when these properties are executed, the data manipulations only occur in the data table in the `DataSet`, not in the database. In order to make these data modifications occur in the real database, the `TableAdapter.Update()` method is needed to send these modifications to the database.

Actually, the terminal execution of inserting, updating, and deleting data of both methods 1 and 2 is performed by method 3. In other words, both methods 1 and 2 need method 3 to complete these data manipulations, which means that both methods need to execute the `Command` object; more precisely, the `ExecuteNonQuery()` method of the `Command` object finishes those data operations against the database.

Because methods 1 and 2 are relatively simple, in this part we will concentrate on inserting data into the database using the `TableAdapter` methods. First, we discuss how to insert new records directly into the database using the `TableAdapter.Insert()` method. Second, we show readers how to insert new records into the `DataSet` and then into a database using the `TableAdapter.Update()` method. Method 3 will be discussed in Part II since it contains more complicated coding related to the runtime objects method.

6.1.1 Insert New Records into a Database Using `TableAdapter.Insert` Method

When you use the `TableAdapter.DBDirect` method to perform data manipulations to a database, the main query must provide enough information for the `DBDirect` methods to be created correctly. The so-called main query is the default or original query method such as `Fill()` and `GetData()` when you first open any `TableAdapter` by using the `TableAdapter Configuration Wizard`. Enough information means that the data table must contain completed definitions. For example, if a `TableAdapter` is configured to query data from a table that does not have a primary key column defined, it does not generate `DBDirect` methods. Table 6.1 lists three `TableAdapter DBDirect` methods.

It can be found from Table 6.1 that the `TableAdapter.Update()` method has two functionalities: One is to directly make all changes in the database based on the parameters contained in the `Update()` method, and another job is to update all changes made in the `DataSet` to the database based on the associated properties of the `TableAdapter`, such as the `InsertCommand`, `UpdateCommand`, and `DeleteCommand`.

Table 6.1 TableAdapter DBDirect Methods

TableAdapter DBDirect Method	Description
TableAdapter.Insert	Adds new records into a database allowing you to pass in individual column values as method parameters.
TableAdapter.Update	Updates existing records in a database. The Update method takes original and new column values as method parameters. The original values are used to locate the original record, and the new values are used to update that record. The TableAdapter.Update method is also used to reconcile changes in a data set back to the database by taking a DataSet, DataTable, DataRow, or array of DataRows as method parameters.
TableAdapter.Delete	Deletes existing records from the database based on the original column values passed as method parameters.

In this chapter, we only take care of inserting data. Thus only the top two methods are discussed in this chapter. The third method will be discussed in Chapter 7.

6.1.2 Insert New Records into a Database Using TableAdapter.Update Method

To use this method to insert data into the database, one needs to perform the following steps:

1. Add new records to the desired DataTable by creating a new DataRow and adding it to the Rows collection.
2. After the new rows are added to the DataTable, call the TableAdapter.Update method. You can control the amount of data to be updated by passing an entire DataSet, a DataTable, an array of DataRows, or a single DataRow.

In order to provide a detailed discussion and explanation of how to use these two methods to insert new records into the database, a real example will be very helpful. Let's first create a new Visual Basic.NET project to handle these issues.

6.2 INSERT DATA INTO MICROSOFT ACCESS DATABASE USING SAMPLE PROJECT INSERTWIZARD

We provided a very detailed introduction on the design tools and wizards in Visual Studio.NET in Section 5.2, in the last chapter, such as DataSet, BindingSource, TableAdapter, Data Source window, Data Source Configuration window, and DataSet Designer. We need to use these to develop our data-inserting sample project based on the SampleWizards project developed in the last chapter. First, let's copy that project and do some modifications on that project to get our new project. The advantage of creating

our new project in this way is that you don't need to redo the data source connection and configuration since those jobs were performed in the last chapter.

6.2.1 Create New Project Based on SampleWizards Project

Open the Windows Explorer and create a new folder such as Chapter 6, and then browse to our project SampleWizards, which is located at the folder **DBProjects\Chapter 5** located at the accompanying ftp site (see Chapter 1). Copy this project to our new folder Chapter 6. Change the name of the solution and the project from SampleWizards to InsertWizard. Double-click on the InsertWizard Project.csproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to Project\InsertWizard Project Properties menu item to open the project's property window. Change the Assembly name from SampleWizards Project to InsertWizard Project, and the Default namespace from SampleWizards_Project to InsertWizard_Project, respectively.
- Click on the Assembly Information button to open the Assembly Information dialog box, change the Title and the Product to InsertWizard Project. Click on OK to close this dialog box.

Go to File\Save All to save those modifications. Now we are ready to develop the graphic user interfaces for our new created project InsertWizard.

6.2.2 Application User Interfaces

As you know from the last chapter, six form windows work as the user interfaces for the SampleWizards project: LogIn, Selection, Faculty, Course, Student, and Grid. Of all these six form windows, only three of them contain the Insert command button, and they are Faculty, Course, and Student. Therefore we need to work on these three forms to perform the data insertion to our database. First, let's concentrate on the Faculty form to perform the data insertion into our Faculty table in the database. To insert a new record into the Faculty table, a separate user interface or form is needed to allow us to enter the new faculty information.

Now let's create another new form, Insert Faculty form.

6.2.3 Create Insert Faculty Form Window

The functionality of this Insert Faculty form is: As the project runs, after the user has finished a correct login process and selected the Faculty Information from the Select form, the Faculty form window will be displayed. When the user clicks on the Insert button, the Insert Faculty form window will appear. This form allows users to use two different methods to insert data into the database: either the TableAdapter.Insert() method or the TableAdapter.Update() method. The form also allows users to enter all pieces of information into the appropriate textboxes for the new inserted faculty. By clicking on the Insert button, a new record about a faculty is inserted into the database.

Table 6.2 Objects for the Insert Faculty Form

Type	Name	Enabled	Text	TabIndex	DropDownStyle
CheckBox	chkPhoto	True	Faculty Photo	2	
GroupBox	GroupBox1	True	Faculty photo	3	
Label	Label1	True	Photo Name	3.0	
TextBox	txtPhotoName	False		3.1	
Label	Label2	True	Photo Location	3.2	
TextBox	txtPhotoLocation	False	Default Location	3.3	
Label	Label3	True	Method	4	
ComboBox	comboMethod	True		5	DropDownList
GroupBox	GroupBox2	True	Faculty ID and name	0	
Label	Label4	True	Faculty ID	0.0	
TextBox	TxtID	True		0.1	
Label	Label5	True	Faculty Name	0.2	
TextBox	txtName	True		0.3	
GroupBox	GroupBox3	True	Faculty information	1	
Label	Label6	True	Title	1.0	
TextBox	TxtTitle	True		1.1	
Label	Label7	True	Office	1.2	
TextBox	txtOffice	True		1.3	
Label	Label8	True	Phone	1.4	
TextBox	txtPhone	True		1.5	
Label	Label9	True	College	1.6	
TextBox	txtCollege	True		1.7	
Label	Label10	True	Email	1.8	
TextBox	txtEmail	True		1.9	
Button	cmdInsert	True	Insert	6	
Button	cmdSelect	True	Select	7	
Button	cmdCancel	True	Cancel	8	
Button	cmdBack	True	Back	9	
PictureBox	PhotoBox	True			
Form	InsertFacultyForm	True	CSE_DEPT Insert Faculty Form		

However, if the user wants to reenter those pieces of information before finishing this insertion, the Cancel button can be used and all information entered will be erased. The Select button is used to validate this data insertion by retrieving back the new inserted data. The Back button is used to allow users to return to the Selection form to perform other data operations.

Go to the Project\Add Windows Form menu item to open the Add New Item dialog box. Select the Windows Form from the Templates box, and enter the Insert Faculty Form.cs into the Name box as the name for this new form. Then click on the Add button to add this form into our project. Add the items shown in Table 6.2 into this form.

In addition to the Form's properties shown in Table 6.2, the following properties of the form should be set up:

- AcceptButton: **cmdInsert** (select the Insert button as the default button)
- StartPosition: **CenterScreen** (locate the form in the center of the screen when it runs)

The checkbox `chkPhoto` is used to allow users to choose the inserting faculty photo, including the name of the faculty photo and the location of that photo, if they like. By checking this checkbox, it indicates that a faculty photo will be included and displayed if this data insertion operation is successful. Also the user needs to provide the photo's

Figure 6.1 Finished Insert Faculty form window.

name (`txtPhotoName`) and the location (`txtPhotoLocation`) in which the photo is stored if this checkbox is checked, and those pieces of information will be used later to load and display that photo when the new inserted faculty is validated by clicking on the **Select** button. Note that the faculty's photo or image file should have been already stored in the location indicated by the `txtPhotoLocation` box before this project can be run. The finished Insert Faculty form window is shown in Figure 6.1.

From both Table 6.2 and Figure 6.1, note that the textbox **Photo Name** and the textbox **Photo Location** are disabled (`Enabled` properties are `False` in Table 6.2). The reason for this is that we want to use the checkbox **Faculty Photo** to control these two controls to make the project more professional. In other words, these two controls are disabled until the checkbox **Faculty Photo** is checked, and in that situation, it allows the user to enter the photo's name and the photo's location into these two controls. If the checkbox **Faculty Photo** is unchecked, which means that the user does not want to include any faculty photo for this data-inserting operation, no faculty photo will be involved and displayed for this data insertion action.

The combobox `comboMethod` allows users to select the different methods to insert data into the database, either the `TableAdapter.Insert()` or `TableAdapter.Update()`.

The order of the `TabIndex` values we created in Table 6.2 is to select the **Faculty Name** textbox as the default one and set a focus to it to allow users to directly enter the faculty name without needing to click that box first.

Now if you compile and build this new project, you would find a lot of compiling and building errors. Do not worry about them and we will fix them in the next section.

6.2.4 Duplicate Visual C#.NET Projects with Installed DataSet

Unlike Visual Basic.NET projects, in which one can create a new project by duplicating an existing project and then make some simple modifications on that duplicated project, in Visual C#.NET, this duplication is not as easy as that in Visual Basic.NET. The main issue is that in Visual C#.NET, the `namespace` is widely used to provide a better protection and identification to different classes and C# objects. These namespaces would not be modified as the project's name is changed, and therefore result in a lot of compiling

and building errors when the project is compiled due to the inconsistency that exists between the namespaces used in new projects and the duplicated project.

Another issue, which is more important, is that this inconsistency becomes a serious problem when a DataSet or database is installed and connected to that existing project. The duplication of project causes not only the conflicts between namespaces used in the new and the duplicated projects, but also the inconsistent namespaces used in the DataSet or database configuration file.

These two issues are involved in the creation of our new project InsertWizard by duplicating an existing project SampleWizards, as we did in the last section. The following steps must be followed to solve these conflict issues:

1. Change all namespaces from SampleWizards_Project to InsertWizard_Project for all C# source files used in this new project. This case includes the following six files: LogIn.cs, Faculty.cs, Course.cs, Selection.cs, Program.cs, and Student.cs.
2. Change all namespaces from SampleWizards_Project to InsertWizard_Project for all C# configuration files. This project includes the following seven files:
 - a. Resources.Designer.cs, under the folder Resources.resx
 - b. Settings.Designer.cs, under the folder Settings.settings
 - c. LogIn.Designer.cs, under the folder LogIn.cs
 - d. Faculty.Designer.cs, under the folder Faculty.cs
 - e. Selection.Designer.cs, under the folder Selection.cs
 - f. Course.Designer.cs, under the folder Course.cs
 - g. Student.Designer.cs, under the folder Student.cs

Both Resources.resx and Settings.settings folders are subfolders of the Properties folder located in the Solution Explorer window. Most namespaces are located at the top of each file and are easy to locate and modify. Some namespaces are located at the middle or bottom of some files. One can first compile the new project, click on each compiling error line from the Output window to locate them, and then make the modification.

3. Change all namespaces from SampleWizards_Project to InsertWizard_Project for all DataSet configuration files. In this project, the following files are involved:
 - a. CSE_DEPTDataSet.cs, under the folder CSE_DEPTDataSet.xsd
 - b. CSE_DEPTDataSet.Designer.cs, under the same folder

An easy way to locate the namespaces from the second file CSE_DEPTDataSet.Designer.cs is to first compile the new project and then click on each compiling error from the Output window to find each of them.

Now perform the steps listed above to open all related files, both C# source files and configuration files as well as the DataSet configuration files, to change all namespaces in those files to complete these modifications. No compiling problem would be encountered if you build this new project again after these modifications are performed.

6.2.5 Validate Data Before Data Insertion

It is important to validate data before they can be inserted into the database since we want to make sure that the data inserted into the database are correct. The most popular validation mode is to make sure that each data item is not NULL and it contains a certain value. Of course, one can insert some NULL values into the database, but here we want

to make sure that each piece of data has a value, either a real value or a NULL value, before they can be inserted into the database.

In this application, we try to validate that each piece of faculty information, which is stored in the associated textbox, is not an empty string unless the user intends to leave it as an empty datum. In that case, a NULL must be entered.

There are two methods to do this validation in Visual C#.NET: (1) using the C# object array and (2) using the control collections. To make this validation more professional, we adopt the second method, which is to develop a control collection and add all of those textboxes into this collection. With method 2 we don't need to check each textbox as we did for the C# object array if we used the first method, but instead we can use the `foreach` loop to scan the whole collection to find the empty textbox.

6.2.5.1 The .NET Framework Collection Classes

The .NET Framework provides a control collection class, and this class can be used to store objects in either the same type or different types in an order. One important issue is that the index value in the .NET Framework collection class is 0 based, which means that the index starts from 0, not 1. The namespace for the .NET Framework control collection class is `System.Collections.Generic`. A generic collection is useful when every item in the collection has the same data type.

To create a .NET Framework collection object `newCSCollection`, one can use one of the following declarations:

```
private Dictionary<string, string> newCSCollection =
    new System.Collections.Generic.Dictionary<string, string>();
```

or

```
private Dictionary<string, string> newCSCollection
    = new Dictionary<string,
    string>();
```

The **Dictionary(Key, Value)** generic class provides a mapping from a set of keys to a set of values. The first declaration uses the full name of the collection class, which means that both the class name and the namespace is included. The second declaration uses only the collection class name with the default namespace. The new created collection object contains two arguments, the `item key` and the `item content`, and both are in the string format. Besides `Dictionary()` class, there are some other classes available to store object collections, such as `Queue()`, `List()`, and `Stack()`. The `Dictionary()` class is more powerful and easier to implement compared with other classes.

Now let's begin to develop the coding for this data validation using this collection component for our application.

6.2.5.2 Validate Data Using Generic Collection

First, we need to create the generic collection object for our Insert faculty form. Since this collection will be used by the different methods in this form, a class-level object or a field object should be created. Open the Code Window of the Insert Faculty form by clicking the View Code button from the Solution Explorer window, and enter the codes that are shown in Figure 6.2 into the fields declaration section.


```

namespace InsertWizard_Project
{
    public partial class InsertFacultyForm : Form
    {
        //private Dictionary<string, string> FacultyCollection = new System.Collections.Generic.Dictionary<string, string>();
        private Dictionary<string, string> FacultyCollection = new Dictionary<string, string>();
    }
}

```

Figure 6.2. Declare the class-level collection object.

The so-called fields declaration section, which is located just under the class header, is used to create all class-level variables or objects. First, we create a field-level string variable `FacultyName`, and this variable will be used to temporarily store the faculty name entered by the user in the `txtName` textbox, and this faculty name will be used later by the `Select` button's `Click` method to validate the new inserted faculty data. Second, the generic collection object, `FacultyCollection`, is created with two arguments: item key and the item content. The light color code in which the default namespace is utilized also works fine. Here we comment it out to illustrate that we prefer to use the full class name to create this collection object.

In order to use the collection object to check all textboxes, one needs to add all textboxes into the collection object after the collection object `FacultyCollection` is created by using the `Add()` method. One point we need to emphasize is the order to perform this validation check. As the project starts, all textboxes are blank. The user needs to enter all faculty information into the appropriate textbox. Then the user clicks on the `Insert` button to perform this data insertion. The time to add all textboxes into the collection object should be after the user finishes entering all pieces of information into all textboxes, not before. Also each time when you finish data validation by checking all textboxes, all textboxes should be removed from that collection since the collection only allows those textboxes to be added once.

Another point to note is that in order to simplify this data validation, in this application we need all textboxes to be filled with certain information or a `NULL` needs to be entered if no information will be entered. In other words, we don't allow any textbox to be empty. The data insertion will not be performed until all textboxes are nonempty in this application. Based on these descriptions, we need to create three user-defined methods to perform these adding, checking, and removing textboxes from the collection object, respectively.

Open the graphical user interface window of the `Insert Faculty` form by clicking on the `View Designer` button from the `Solution Explorer` window, and then double-click on the `Insert` button to open its `Click` method. Enter the codes shown in [Figure 6.3](#) into this method.

Let's take a look at this piece of code to see how it works.

- A.** First, we need to create an integer variable `check` and initialize it to zero. The purpose of this variable is to delay checking the status of the `CheckFacultyCollection()` method until later to indicate whether the validation of all textboxes is successful.
- B.** Next, we need to call the `CreateFacultyCollection()` method to add all textboxes into the collection `FacultyCollection`. Refer to [Figure 6.4](#) for the detailed coding for this method.


```

InsertWizard_Project.InsertFacultyForm  cmdInsert_Click
private void cmdInsert_Click(object sender, EventArgs e)
{
    A     int check = 0;
    B     CreateFacultyCollection();
    C     check = CheckFacultyCollection();
    D     if (check == 0)
    {
    E         MessageBox.Show("OK, No problem and no TextBox is empty...");
        if (chkPhoto.Checked == true && (txtPhotoName.Text == "" || txtPhotoLocation.Text == ""))
            MessageBox.Show("Photo Name/Photo Location is empty");
        // continue to perform the data insertion.....
    }
    F     RemoveFacultyCollection();
}

```

Figure 6.3. Coding for the Insert button Click method.

```

InsertWizard_Project.InsertFacultyForm  CreateFacultyCollection()
private void CreateFacultyCollection()
{
    FacultyCollection.Add("Faculty ID", txtID.Text);
    FacultyCollection.Add("Faculty Name", txtName.Text);
    FacultyCollection.Add("Faculty Title", txtTitle.Text);
    FacultyCollection.Add("Faculty Office", txtOffice.Text);
    FacultyCollection.Add("Faculty Phone", txtPhone.Text);
    FacultyCollection.Add("Faculty College", txtCollege.Text);
    FacultyCollection.Add("Faculty Email", txtEmail.Text);
}

```

Figure 6.4 Coding for the CreateFacultyCollection() method.

```

InsertWizard_Project.InsertFacultyForm  CheckFacultyCollection()
private int CheckFacultyCollection()
{
    A     int check = 0;
    B     foreach (KeyValuePair<string, string> strCheck in FacultyCollection)
    {
    C         if (strCheck.Value == string.Empty)
        {
            MessageBox.Show(strCheck.Key + " is empty!");
            check++;
        }
    D     }
    return check;
}

```

Figure 6.5 Coding for the CheckFacultyCollection() method.

- C. Then we need to execute the CheckFacultyCollection() method to validate whether all textboxes are nonempty. Refer to Figure 6.5 for the detailed coding for this method.
- D. If the checking status returned from the CheckFacultyCollection() method is zero, which means that all textboxes are nonempty, the validation is successful. At this moment, we use a MessageBox() method to indicate this situation. Later on, we need to replace this method by developing the codes to perform the data insertion.

```

private void RemoveFacultyCollection()
{
    FacultyCollection.Remove("Faculty ID");
    FacultyCollection.Remove("Faculty Name");
    FacultyCollection.Remove("Faculty Title");
    FacultyCollection.Remove("Faculty Office");
    FacultyCollection.Remove("Faculty Phone");
    FacultyCollection.Remove("Faculty College");
    FacultyCollection.Remove("Faculty Email");
}

```

Figure 6.6 Coding for the RemoveFacultyCollection() method.

- E. If the Checked property of the chkPhoto is true, which means that the user checked the Faculty Photo checkbox and wants to include a faculty photo for this data insertion, then we need to check whether the related textboxes contain a valid photo name and a photo location. A MessageBox will be shown to remind users to enter both the photo name and the photo location if any of them is empty.
- F. Finally the RemoveFacultyCollection() method is called to remove all textboxes that have been added into the collection in the CreateFacultyCollection() method since the collection only allows those textboxes to be added once. The detailed code for the RemoveFacultyCollection() method is shown in Figure 6.6.

Now let's take care of the coding for the CreateFacultyCollection() method, which is shown in Figure 6.4.

The coding is very simple and straightforward. Each textbox is added into the collection by using the Add() method with two parameters, Key and Value. The Key parameter works as an identifier for the object, and the Value parameter contains the content of the object—textbox in this case. Both parameters are represented in a string format. In this way, each object, or textbox, can be identified by its Key. Of course, each textbox can also be identified by its index, but remember that the index starts from 0, not 1, since it is a .NET Framework collection.

To check and validate all textboxes from the collection, the CheckFacultyCollection() method is executed. The code for this method is shown in Figure 6.5.

Let's take a look at this piece of code to see how it works.

- A. First, we need to create an integer variable check and initialize it to zero. The purpose of this variable is to hold the returned checking status of execution of this method and return this status to the calling method.
- B. A foreach loop is executed to scan each textbox stored in the FacultyCollection collection. Note that the data type of the loop variable or the iteration variable strCheck must be an IEnumerable or IEnumerable<T> (refer to Sections 4.2 and 4.4.2 in Chapter 4). Also the foreach statement of the C# requires the type of the elements in the collection. Since each element of a collection based on Dictionary(Key, Value) is a Key/Value pair, the element type is not the type of the Key or the type of the Value. Instead, the element type is **KeyValuePair(Key, Value)**. Since both Key and Value are in String format, the string type is used for both of them for the element strCheck.
- C. An if block is used to check whether any textbox is empty or not by using the Value of each textbox. A message will be displayed if any textbox is empty with the

```

InsertWizard_Project.InsertFacultyForm
chkPhoto_CheckedChanged()

private void chkPhoto_CheckedChanged(object sender, EventArgs e)
{
    if (chkPhoto.Checked == true)
    {
        txtPhotoName.Enabled = true;
        txtPhotoLocation.Enabled = true;
        txtPhotoName.Focus();
    }
    else
    {
        txtPhotoName.Enabled = false;
        txtPhotoLocation.Enabled = false;
    }
}

```

Figure 6.7 Coding for the `chkPhoto_CheckedChanged()` method.

Key as the identifier. The check variable is increased by 1 if an empty textbox is detected.

D. Finally the checking status variable `check` is returned to the calling method.

To remove all textboxes from the collection, the `RemoveFacultyCollection()` method should be called, and the coding for this subroutine is shown in Figure 6.6.

The `Key` parameter of each textbox is used as the identifier for each textbox, and the `Remove()` method is called to remove all textboxes from the collection object.

The next coding job we need to handle is the coding for the checkbox `chkPhoto`. Recall that when the project begins to run, the Photo Name and the Photo Location textboxes are disabled. As the user checks this checkbox, it means that the user wants to include a faculty photo in the data insertion. Both the Photo Name and the Photo Location boxes should be enabled and focused to allow users to enter the associated information about the faculty photo. Open the Form Window by clicking on the View Designer button from the Solution Explorer window. Double-click on the Faculty Photo checkbox to open the `chkPhoto_CheckedChanged()` method. Enter the codes shown in Figure 6.7 into this method.

First, we need to check whether the `chkPhoto` checkbox has been selected or not. Because when the `chkPhoto` checkbox is clicked by the user, there are two possibilities existed: the `chkPhoto` is checked or unchecked. If it is checked, both `txtPhotoName` and `txtPhotoLocation` textboxes should be enabled, and a focus should be set for the Photo Name textbox to enable the user to directly enter the photo name into that textbox without clicking on it first. Otherwise both textboxes should be disabled if the `chkPhoto` is not checked.

At this point, we have completed the coding for the data validation. Next we need to handle some initialization and termination coding for the data insertion.

6.2.6 Initialization and Termination Coding for Data Insertion

In this section, we need to handle the coding for the following methods:

- Coding for the `Form_Load` method to initialize the combobox `comboMethod` to display two data insertion methods: `TableAdapter.Insert()` and `TableAdapter.Update()`

The screenshot shows a code editor window with the following content:

```

InsertWizard_Project.InsertFacultyForm ▼ InsertFacultyForm_Load() ▼
private void InsertFacultyForm_Load(object sender, EventArgs e)
{
    comboMethod.Items.Add("TableAdapter Insert");
    comboMethod.Items.Add("TableAdapter Update");
    comboMethod.SelectedIndex = 0;
}

```

Figure 6.8 Coding for the Form_Load method.

The screenshot shows a code editor window with the following content:

```

InsertWizard_Project.InsertFacultyForm ▼ cmdCancel_Click() ▼
private void cmdCancel_Click(object sender, EventArgs e)
{
    txtID.Text = string.Empty;
    txtName.Text = string.Empty;
    txtTitle.Text = string.Empty;
    txtOffice.Text = string.Empty;
    txtPhone.Text = string.Empty;
    txtCollege.Text = string.Empty;
    txtEmail.Text = string.Empty;
    txtPhotoName.Focus();
}

```

Figure 6.9 Coding for the Cancel button method.

- Coding for the Cancel button Click method to erase all contents from those textboxes that contained the faculty information
- Coding for the Back button Click method to return the program to the Selection form window to allow users to select other data operations.

The first coding is for the combobox `comboMethod`. As the project runs and the Insert Faculty form window appears, two different methods should be displayed in this box to allow users to select one to perform the data insertion: either the `TableAdapter DBDirect` method, `TableAdapter.Insert()`, or the `TableAdapter.Update()` method. The display of this selection should be made first as the form window is loaded and displayed; therefore, we need to put this coding into the `Form_Load()` method. Open the Code Window for the Insert Faculty form by clicking on the View Code button from the Solution Explorer window; then select the `InsertFacultyForm_Load` item from the Members combobox to open this method. Enter the code shown in Figure 6.8 into this method (remove the default codes from this method).

The coding is straightforward and easy to be understood. Two methods are added into the combobox by using the `Add()` method, and the first method is selected as the default one by setting up the `SelectedIndex` property to zero.

The coding for other command buttons such as the Cancel and the Back is simple. The function of the Back button is to return the program to the Selection form window, so its coding is only one line: `this.Hide()`. The function of the Cancel button is to clean up all textboxes' contents except the Photo Name and the Photo Location, and we want to keep their contents to be used later. The Default Location in the Photo Location textbox means that the photo is located at the folder in which the Visual C#. NET executable file is located. The coding for the Cancel button is shown in Figure 6.9.

To clean up the content of a textbox, either an empty string “ ” or the `Empty` field of the string class `string.Empty` can be used, and both need to be assigned to the `Text` property of the textbox to be cleaned. A focus is set to the Photo Name textbox that is convenient for the user.

Now we need to take care of the coding for the data inserting. Because we are using the design tools to perform this job, we need first to configure the `TableAdapter` and build the insert query using the `TableAdapter Query Configuration Wizard`.

6.2.7 Build Insert Query

As we mentioned, two methods will be discussed in this part: One is to insert new records using the `TableAdapter DBDirect` method `TableAdapter.Insert()` to insert data into the database, and the other one is to use the `TableAdapter.Update()` method to insert new records into the database. Let's now concentrate on the first method.

6.2.7.1 Configure TableAdapter and Build Data Insertion Query

In order to use the `TableAdapter.Insert()` `DBDirect` method to access the database, we need first to configure the `TableAdapter` and build the Insert query.

Open the `Data Source` window by going to the `Data>Show Data Sources` menu item. On the opened window, click on the `Edit the DataSet with Designer` button located at the second left on the toolbar in the `Data Source` window to open this `Designer`. Then right-click on the bottom item from the `Faculty` table and select the `Add Query` item from the pop-up menu to open the `TableAdapter Query Configuration Wizard`. Keep the default selection `Use SQL statements` unchanged and click on `Next` to go to the next window. Select and check the `INSERT` item from this window since we need to insert new records query, and then click on the `Next` button again to continue. Click on the `Query Builder` button to build our insert query. The opened `Query Builder` window is shown in Figure 6.10.

The default Insert query statement is identical with our requirement since we want to insert a new faculty record that contains new information about that inserted faculty, which includes the `faculty_id`, `faculty_name`, `office`, `phone`, `college`, `title`, and `email`. Click on the `OK` button to go to the next window. Click on the `Next` button to confirm this query and continue to the next step. Modify the query function name from the default one to the `InsertFaculty` and click on the `Next` button to go to the last window. Click on the `Finish` button to complete this query building and close the wizard. Immediately you can find that a new query function has been added into the `FacultyTableAdapter` as the last item.

Now that we have finished the configuration of the `TableAdapter` and building of the insert query, it is time for us to develop the codes to run the `TableAdapter` to complete this data-inserting query.

But wait a moment! You probably encountered two problems after you finished the building of this query:

1. No database-related object is available to this project. For example, in order to perform data manipulations, we need to use some design tools and wizard-related objects such as `DataSet` and `TableAdapter` to develop codes to access and insert data to our database.

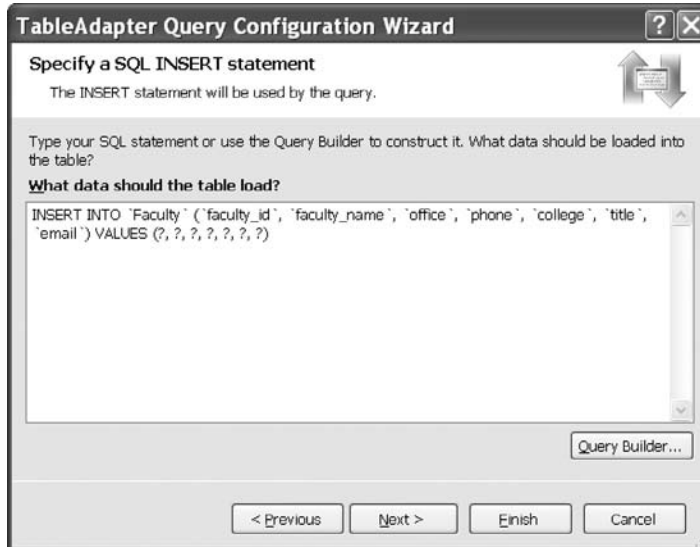


Figure 6.10 Opened Query Builder window.

2. You will encounter some compiling errors if you build your project now. The reason for that is because we modified the DataSet configuration file when we built the Insert query using the TableAdapter Query Configuration Wizard above. The DataSet we are using was created in an existing project, the SampleWizards Project we developed in Chapter 5, and we only copied this project and renamed it to our current project InsertWizard Project in Section 6.2.1. Therefore, the namespace used in the DataSet configuration file is still the old one, which is SampleWizards_Project.

To solve these two problems, we need to perform the following two steps:

1. To create and add database-related objects, we need to perform the data binding between some object in the InsertFacultyForm and the mapping item in the database. Of course, one can perform this binding between any object in the InsertFacultyForm and its associated data item in the database. In this application, we try to use the Faculty Name textbox and faculty_name item in the Faculty table in our sample database. To do this binding, follow the steps below:
 - a. On the opened InsertFacultyForm window, select the Faculty Name textbox.
 - b. Go to the Property window and expand the DataBindings item.
 - c. Go to Text property and click on the drop down arrow. Expand all items until the CSE_DEPTDataSet; then expand the Faculty table.
 - d. Select faculty_name from the Faculty table.

Immediately you will find that three database-related objects, cSE_DEPTDataSet, facultyTableAdapter, and facultyBindingSource, are created and added under the InsertFacultyForm window.

2. To correct all compiling errors that appeared in the DataSet configuration file, just replace all old namespaces, SampleWizards_Project, with the current namespace InsertWizard_Project. An easy way to do this is find each error by double-clicking on each error in the Output window, and perform that replacement one by one.

Now let's start our coding job. First, we need to develop the codes for the first method—using the TableAdapter DBDirect method, TableAdapter.Insert().

6.2.8 Develop Codes to Insert Data Using TableAdapter.Insert Method

Open the graphical user interface of the Insert Faculty Form by clicking the View Designer button from the Solution Explorer window, and then double-click the Insert button to open its Click method and add the codes that are shown in Figure 6.11 into this method.

Recall that we have created some codes for this method in Section 6.2.5.2 to perform the data validation. Thus, these codes are indicated by shading in the background.

Let's have a look at this piece of code to see how it works.

- A. A local integer variable `intInsert` is declared here, and it is used to hold the returned value from execution of the `TableAdapter.Insert()` method. The value of this returned integer, which indicates how many records have been successfully inserted or affected to the database, can be used to determine whether this data insertion is successful or not. A returned value of zero means that no record is added or affected to the database; in other words, this insertion has failed.
- B. If the user selected the TableAdapter Insert method to perform this data insertion, the query function `InsertFaculty()`, which we built in the last section by using the TableAdapter Query Configuration Wizard, will be called to complete this data insertion. Seven pieces of new information, which is about the new inserted faculty and entered by the user into seven textboxes, will be inserted to the Faculty table in the database.

```

InsertWizard_Project.InsertFacultyForm  cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
  int check = 0, intInsert = 0;
  CreateFacultyCollection();
  check = CheckFacultyCollection();
  if (check == 0)
  {
    MessageBox.Show("OK, No problem and no TextBox is empty...");
    if (chkPhoto.Checked == true && (txtPhotoName.Text == "" || txtPhotoLocation.Text == ""))
      MessageBox.Show("Photo Name/Photo Location is empty");
    if (comboBoxMethod.Text == "TableAdapter Insert")
      intInsert = facultyTableAdapter.InsertFaculty(txtID.Text, txtName.Text, txtOffice.Text,
        txtPhone.Text, txtCollege.Text, txtTitle.Text, txtEmail.Text);
    else
    {
      // coding for the second method TableAdapter Update().....
    }
    if (intInsert != 0) // data insertion is successful
    {
      cmdCancel.PerformClick(); // clean up all faculty information
      cmdInsert.Enabled = false; // disable the Insert button to avoid the duplicated insertion
    }
    else
    {
      MessageBox.Show("The data insertion is failed");
      cmdInsert.Enabled = true; // enable the Insert button to allow the insertion
    }
  }
  RemoveFacultyCollection(); // remove all textboxes from the collection
}

```


Figure 6.11 Modified coding for the Insert button Click method.

- C. If the user selected the `TableAdapter.Update()` method to perform this data insertion, the data insertion should be performed by calling that method. The coding for this method will be discussed in the next section.
- D. If this data insertion is successful, the returned integer value will reflect the number of records that have been inserted into the database correctly. As we mentioned in step A, a returned value of nonzero indicates that this insertion is successful. An equivalent clicking on the Cancel button is executed to trigger that button's Click method to clean up all inserted faculty information from the six textboxes to make them ready for the next data insertion.
- E. If the `intInsert` returns a zero, this data insertion has failed. A `MessageBox` will be displayed with a warning message to indicate this situation to the user. One point we need to emphasize is that when performing a data insertion, the same data can only be inserted into the database once, and the database does not allow multiple insertions of the same data item. To avoid multiple insertions, in this application (actually in most popular applications) we need to disable the Insert button once a record is inserted successfully (refer to step D). If the insertion has failed, we need to recover or reenble the Insert button to allow the user to try another insertion later.



Most databases, including Microsoft Access, SQL Server, and Oracle, do not allow multiple data insertions of the same data item into the databases. Each data item or record can only be added or inserted into the database once. In other words, no duplicating record can be added or exists in the database. Each record in the database must be unique. The popular way to avoid this situation is to disable the Insert button after one insertion is done.

From the above explanations, we know that it is a good way to avoid the multiple insertions of the same data item into the database by disabling the Insert button after that insertion is successfully completed. A question arises: When and how can this button be enabled again to allow us to insert some other different new records if we want to do that later? The answer to this question is to develop another method to handle this issue. Think about it. When we want to insert new different data items into the database, first we must enter each piece of new faculty information into each associated textbox such as `txtID`, `txtName`, `txtOffice`, `txtPhone`, `txtTitle`, `txtCollege`, and `txtEmail`. In other words, as long as the content of a textbox is changed, which means that a new different record has been inserted, we should enable the Insert button at that moment to allow users to perform this new insertion. Visual C#.NET did provide an event called `TextChanged` and an associated method for the textbox control. Therefore, we can use this event and method to enable the Insert button as long as a `TextChanged` event occurs. Well, another question arises: With which textbox's `TextChanged` event should we trigger the associated method to enable the Insert button to allow users to insert a new record? Can any textbox's `TextChanged` event do this? To answer this question, we need to review the data issue in the database. As you know, in our sampling database `CSE_DEPT` (actually in our Faculty data table), it identifies a record based on its primary key. In other words, only those records with different primary keys can be considered as different records. So the solution to our question is: If the content of the textbox that stores the primary key, in our case it is the `txtID` that stores the `faculty_id`, is changed, it means that a new record will be inserted. As this happens, that textbox's `TextChanged` method should be triggered to enable the Insert button.



```

InsertWizard_Project.InsertFacultyForm
FacultyInfoChanged()

private void txtID_TextChanged(object sender, EventArgs e)
{
    cmdInsert.Enabled = true;
}

```

Figure 6.12 Coding for the TextChanged method.

To open the TextChanged method for the textbox txtID, open the graphical user interface of the Insert Faculty form window by clicking on the View Designer button from the Solution Explorer window, and then double-click on the txtID (Faculty ID) textbox to open its TextChanged method. Enter the codes into this method shown in Figure 6.12.

The coding for this method is simple: Enable the Insert button by setting the Enabled property of that button to true as the txtID TextChanged event occurs.

Now that we have finished the coding for the first data insertion method, let's continue to do the coding for the second method, TableAdapter.Update method.

6.2.9 Develop Codes to Insert Data Using TableAdapter.Update Method

When a data-driven application uses DataSet to store data, as we did for this application by using the CSE_DEPTDataSet, one can use the TableAdapter.Update() method to insert or add a new record into the database.

To insert a new record into the database using this method, two steps are needed:

1. Add new records to the desired data table in the DataSet; for example, in this application, the Faculty table in the DataSet CSE_DEPTDataSet.
2. Call the TableAdapter.Update() method to update new added records from the data table in the DataSet to the data table in the database. The amount of data to be updated can be controlled by passing the different argument in the Update() method, either an entire DataSet, a DataTable, an array of DataRow, or a single DataRow.

Now let's develop our codes based on these two steps to insert data using this method. Reopen the graphical user interface of the Insert Faculty form window and double-click on the Insert button to open its Click method. We have already developed most codes for this method in the last section, and now we need to add the codes to perform the second data insertion method. Browse to the else block (step E in Figure 6.11), and enter the codes shown in Figure 6.13 into this block.

In order to distinguish between the new codes and the old codes that have been added before, all old codes are indicated with shading.

Let's take a close look at this piece of new inserted code to see how it works.

- A. First, we need to declare a new object of the DataRow class. Each DataRow object can be mapped to a real row in a data table. Since we are using the DataSet to manage all data tables in this project, the DataSet must be prefixed before the DataRow object. Also we need to create a row in the Faculty data table, the FacultyRow is selected as the DataRow class.

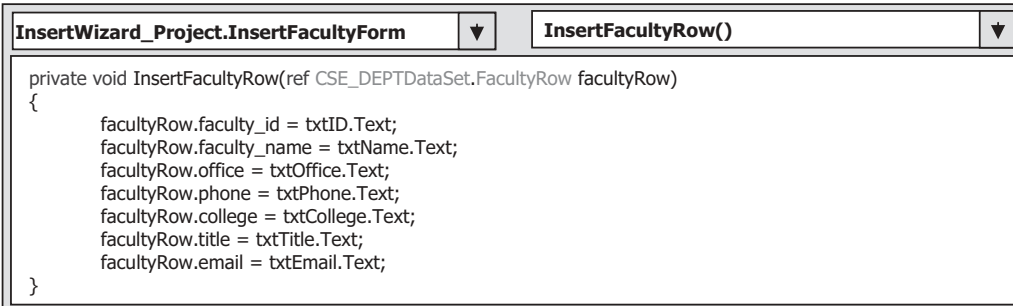
```

InsertWizard_Project.InsertFacultyForm  cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
    int check = 0, intInsert = 0;
    CSE_DEPTDataSet.FacultyRow newFacultyRow;
    CreateFacultyCollection();
    check = CheckFacultyCollection();
    if (check == 0)
    {
        MessageBox.Show("OK, No problem and no TextBox is empty...");
        if (chkPhoto.Checked == true && (txtPhotoName.Text == "" || txtPhotoLocation.Text == ""))
            MessageBox.Show("Photo Name/Photo Location is empty");
        if (comboMethod.Text == "TableAdapter Insert")
            intInsert = facultyTableAdapter.InsertFaculty(txtID.Text, txtName.Text, txtOffice.Text,
                txtPhone.Text, txtCollege.Text, txtTitle.Text, txtEmail.Text);
        else // TableAdapter.Update() method is selected...
        {
            newFacultyRow = this.cSE_DEPTDataSet.Faculty.NewFacultyRow();
            InsertFacultyRow(ref newFacultyRow);
            cSE_DEPTDataSet.Faculty.Rows.Add(newFacultyRow);
            intInsert = facultyTableAdapter.Update(cSE_DEPTDataSet.Faculty);
        }
        if (intInsert != 0) // data insertion is successful
        {
            cmdCancel.PerformClick(); // clean up all faculty information
            cmdInsert.Enabled = false; // disable the Insert button
        }
        else
        {
            MessageBox.Show("The data insertion is failed");
            cmdInsert.Enabled = true;
        }
    }
    RemoveFacultyCollection();
}

```

Figure 6.13 Coding for the second data insertion method.

- B.** Next, we need to create a new object of the `NewFacultyRow` class.
- C.** A user-defined method `InsertFacultyRow()` is called to add all pieces of faculty information about the new inserting faculty, which is stored in seven textboxes, into this new created `DataRow` object. The coding and the functionality of this user-defined method are explained below. The method returns a completed `DataRow` that contains all seven pieces of faculty information about the new record. Note that a reference parameter, `ref newFacultyRow`, is passed into the method, which means that the address (not the value) of the object `newFacultyRow` is passed into the method, and any modification to this object is permanent. We just need this permanent property to keep our returned `newFacultyRow` object updated.
- D.** The completed `DataRow` is added into the `Faculty` table in our `DataSet` object. Note that adding a new record into a data table in the `DataSet` has nothing to do with adding a new record into a data table in the database. The data tables in the `DataSet` are only mappings of those real data tables in the database. To add this new record into the database, one needs to perform the next step.
- E.** The `TableAdapter`'s method `Update()` is executed to have this new record added into the real database. As we mentioned before, you can control the amount of data to be added into the database by passing the different arguments. Here we only want to add one new



```

InsertWizard_Project.InsertFacultyForm
InsertFacultyRow()

private void InsertFacultyRow(ref CSE_DEPTDataSet.FacultyRow facultyRow)
{
    facultyRow.faculty_id = txtID.Text;
    facultyRow.faculty_name = txtName.Text;
    facultyRow.office = txtOffice.Text;
    facultyRow.phone = txtPhone.Text;
    facultyRow.college = txtCollege.Text;
    facultyRow.title = txtTitle.Text;
    facultyRow.email = txtEmail.Text;
}

```

Figure 6.14 Coding for the user-defined `InsertFacultyRow()` method.

record into the Faculty table, so a data table is passed as the argument. This `Update()` method supposes to return an integer value to indicate whether this update is successful or not. The value of this returned integer is equal to the number of rows that have been successfully added into the database. A returned value of zero means that this update has failed since no new row has been added into the database.

Now let's develop the coding for the user-defined method `InsertFacultyRow()`. Open the code window and enter the code shown in Figure 6.14 into this method.

The functionality of this piece of coding is straightforward and easy to understand. Seven pieces of new faculty information stored in the associated textboxes is added into the new `DataRow` object, actually added into a new row of the faculty table in the `DataSet`. Since the argument `facultyRow` is passed into this method using a reference value, any modification to this object `facultyRow` is permanent, which means that this modification can be returned to the calling method and kept valid until the project is terminated or exited.

At this point, we completed the coding for our data insertion by using two methods. Now let's test our coding by running our project. You have two methods to test the project. One method is to run the project in a formal way, which means that you run the project starting from the `LogIn` form, `Selection` form, and then the `Faculty` form. Then you can click on the `Insert` button in the `Faculty` form to open the `Insert Faculty` form window to test the data insertion. The second method, which is more flexible, is to directly start from the `Insert Faculty` form. To run the project with the first method, you need to add some codes and methods into the `Faculty Form` window.

Perform the following operations to finish adding these codes:

1. Create a class-level or fields-level object of the `Insert Faculty Form` class since we need to open the `Insert Faculty Form` window by calling this object as the users click on the `Insert` button from the `Faculty Form` window. Refer to step **A** in Figure 6.15 to create this object.
2. Calling the object created in step 1 from the `Insert` button `Click` method in the `Faculty Form` directs the program to open the `Insert Faculty` form window as the `Insert` button is clicked. To do that, go to the `Faculty Form` window and open the `Insert` button `Click` method by double-clicking on the `Insert` button. Enter the codes shown in step **B** in Figure 6.15 into this method.
3. You need to close the `Insert Faculty Form` window when the `Faculty Form` window is terminated. Add one instruction into the `Back` button's `Click` method in the `Faculty Form` window, which is shown in step **C** in Figure 6.15.

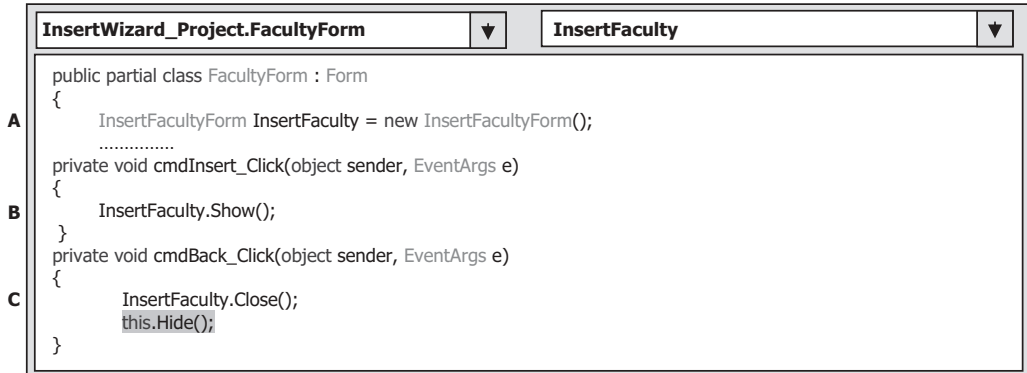


Figure 6.15 Added coding for the Faculty Form.

To run the project by the second method, one must modify the startup object from inside the `Main()` method of this project. Open the `Main()` method by double-clicking on the **Program.cs** file from the Solution Explorer window. Change the third instruction from

```
Application.Run(new LogInForm());
```

to

```
Application.Run(new InsertFacultyForm());
```

Now you can run the project by either method. However, we still prefer to run it by the first method. Recover the startup object to `LogInForm` in the `Main()` method, and then click on the Start Debugging button to run the project. Enter the correct username and password to the `LogIn` form, and select the Faculty Information from the Selection Form window. Click on the Insert button from the opened Faculty Form window to open the Insert Faculty Form window, which is shown in Figure 6.16.

Now let's test the first method, `TableAdapter.Insert()`, to add a new faculty record into the database. Keep the default value in the combobox unchanged. Enter the information for this new faculty member into the associated textbox, such as the `txtID`, `txtName`, `txtOffice`, `txtPhone`, `txtCollege`, `txtTitle`, and `txtEmail`. After all seven pieces of new information have been entered into all associated textboxes, click on the Insert button to execute this data insertion using the first method. If this insertion is successful, a successful message box with a message: "OK, No problem and no TextBox is empty ..." will be displayed, and the contents of all textboxes are cleaned up. The Insert button is disabled to avoid the same data being added into the database more than one time (you can delete that successful message if you want to speed up the project).

Click on the Back-Back-Exit buttons for Insert Faculty Form, Faculty Form, and Selection Form windows to terminate the project. To confirm this data insertion, we can open the database to check whether this new record has been added into the Faculty table. To do that, go to the Debug folder of the current project; in our case, it is `C:\Chapter6\InsertWizard Solution\InsertWizard Project\bin\Debug`, and double-click on the embedded database `CSE_DEPT.accdb` to open it. Then open the Faculty table by double-clicking on it, too. The opened Faculty table is shown in Figure 6.17.

Figure 6.16 Running status of the Insert Faculty Form window.

Figure 6.17 Opened Faculty table.

It can be found from this table that the second row with `faculty_id` as A56789 is the new record we just added into this Faculty table. Recall that there are in total eight faculty members in this Faculty table when we designed this database in Chapter 2. Count how many records in this table right now. It is nine. So this is why our data insertion is very successful!

It is highly recommended to remove this new inserted faculty from the Faculty table since we want to keep our database clean and neat. Remove this new inserted faculty from the Faculty table by selecting the second record, right-clicking on it, and selecting `Delete Record` from the pop-up menu and close the database. You can try to insert this new faculty record using the second method, and it will also be successful.

Next, we want to develop a more professional way to verify our data insertion operation. This method retrieves the inserted records from the database and displays them in the Faculty Form window to confirm our data insertion. We need to use the `Select` button in the Faculty Form window and its `Click` method to complete this job.

6.2.10 Validate Data After Data Insertion

To use the Select button and its Click method in the Faculty Form window to confirm our data insertion, we need to perform the following modifications to both Insert Faculty Form and Faculty Form code windows. First, let's do the coding modifications for the Faculty Form code window:

1. Modify the codes inside the FacultyForm_Load() method to retrieve the current faculty members from the Faculty table in the database. This can be realized by adding a new user-defined method UpdateFaculty().
2. Modify the codes inside the Insert button's Click method by adding an instruction to set up an instance of the FacultyForm class in the InsertFacultyForm window. The reason is because we need to use some object and method located in the FacultyForm window from the InsertFacultyForm, such as the ComboName and the UpdateFaculty(), to update the faculty members stored in the ComboName box in the FacultyForm window. By setting up this instance, we can easily call and use the FacultyForm from the InsertFacultyForm object.
3. Modify the codes inside the FindName() method to check and display the photo of the new inserted faculty. A default photo will be displayed if no photo is used with that faculty insertion.

Now let's begin to do these modifications for the FacultyForm window.

6.2.10.1 Modifications to Faculty Form Window

Open the FacultyForm_Load() method in the Faculty Form window and perform the following modifications as shown in Figure 6.18.

Let's take a look at this piece of code to see how it works.

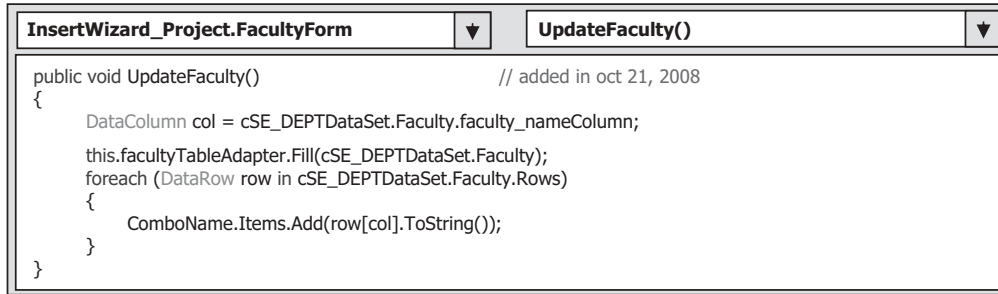
- A. Remove eight Add() methods from the ComboName object, which have been highlighted with shading. Recall that we always load these eight faculty members and add them into the ComboName box using this Add() method in the previous projects we built in the last

```

InsertWizard_Project.FacultyForm FacultyForm_Load()
private void FacultyForm_Load(object sender, EventArgs e)
{
    //ComboName.Items.Add("Ying Bai");
    //ComboName.Items.Add("Satish Bhalla");
    //ComboName.Items.Add("Black Anderson");
    //ComboName.Items.Add("Steve Johnson");
    //ComboName.Items.Add("Jenney King");
    //ComboName.Items.Add("Alice Brown");
    //ComboName.Items.Add("Debby Angles");
    //ComboName.Items.Add("Jeff Henry");
    UpdateFaculty(); // added in oct 23 2008
    ComboName.SelectedIndex = 0;
    this.cmdSelect_Click(this.cmdSelect, null);
    ComboMethod.Items.Add("TableAdapter Method");
    ComboMethod.Items.Add("LINQ & DataSet Method");
    this.ComboMethod.SelectedIndex = 0;
}

```

Figure 6.18 Modifications to the FacultyForm_Load() method.



```

InsertWizard_Project.FacultyForm UpdateFaculty()
public void UpdateFaculty() // added in oct 21, 2008
{
    DataColumn col = cSE_DEPTDataSet.Faculty.faculty_nameColumn;
    this.facultyTableAdapter.Fill(cSE_DEPTDataSet.Faculty);
    foreach (DataRow row in cSE_DEPTDataSet.Faculty.Rows)
    {
        ComboName.Items.Add(row[col].ToString());
    }
}

```

Figure 6.19 Detailed codes for the UpdateFaculty() method.

chapter. The reason is because in those projects we only needed to perform the data retrieving operations without changing the data in our database, and we assumed that the faculty members stored in the Faculty table were identical with those we added into the ComboName box. However, the situation is changed in this project since we need to insert some new faculty members into our database; therefore, we need to pick up and display the updated or the current faculty members each time the FacultyForm window is opened.

- B.** Add a new user-defined method, UpdateFaculty(), to this Form Load method to pick up and display the updated faculty members in the ComboName box. The new added method has been highlighted with the bold style. The detailed codes for this method is shown in Figure 6.19. One point to be noted is that the accessing mode for this method is `Public` since we need to call this method from the Insert Faculty Form window later; therefore, we need to allow this method to be accessed globally.

All of the rest of the codes in this method are kept unchanged. The detailed codes of the UpdateFaculty() method are shown in Figure 6.19.

Let's take a look at this piece of code to see how it works.

- A.** Create a new instance of the DataColumn class based on the faculty_nameColumn class that can be mapped to the faculty_name column in the Faculty table in our DataSet since we need to access this column to retrieve the current faculty members (names).
- B.** The default Fill() method of the FacultyTableAdapter is executed to load the updated faculty members from the Faculty table in our database and fill them into the Faculty table in our DataSet.
- C.** A foreach loop is executed with each row in the Faculty table as the loop counter, and each faculty name stored in the faculty_name column is retrieved and added into the ComboName box using the Add() method. A ToString() method is utilized to convert each row that is a DataRow object to a string before it can be added into the ComboName box.

Now let's modify the codes inside the Insert button's Click method by adding an instruction to set up an instance of the FacultyForm class in the InsertFacultyForm window. Figure 6.20 shows this added instruction, which is highlighted in bold.

As we mentioned, we need to update the faculty members stored in the ComboName box in the Faculty Form window from the Insert Faculty Form window after a new faculty has been inserted to our database. Therefore, we need to access the Faculty Form window to perform this updating from the Insert Faculty Form window. In order to do that, we need to set up an instance of the Faculty Form class to the Insert Faculty Form window


```

InsertWizard_Project.FacultyForm  cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
    InsertFaculty.setFacultyForm(this);           // added in oct 23, 2008
    InsertFaculty.Show();
}

```

Figure 6.20 Modifications to the Insert button's Click method.

before we can call and run the Insert Faculty Form window by using a new method `setFacultyForm()`, which will be developed later in the Insert Faculty Form window. We do not want to create a new instance of the Faculty Form class. Instead, we want to use the current instance of the Faculty Form class; therefore, the current instance name **this** is passed as the argument for that method.

Next, let's modify the codes inside the `FindName()` method to check and display the photo of the new inserted faculty. The purpose of this method is to identify and display the correct photo for the selected faculty member in the PhotoBox. Three possible situations exist for this faculty photo when a new faculty is inserted from the Insert faculty Form window:

1. The Faculty Photo checkbox, `chkPhoto`, in the Insert Faculty Form window is not checked, which means that the user inserted a new faculty without any photo accompanying that insertion.
2. The Faculty Photo checkbox is checked and the Photo Location is the Default Location.
3. The Faculty Photo checkbox is checked and the Photo Location textbox contains a valid photo location.

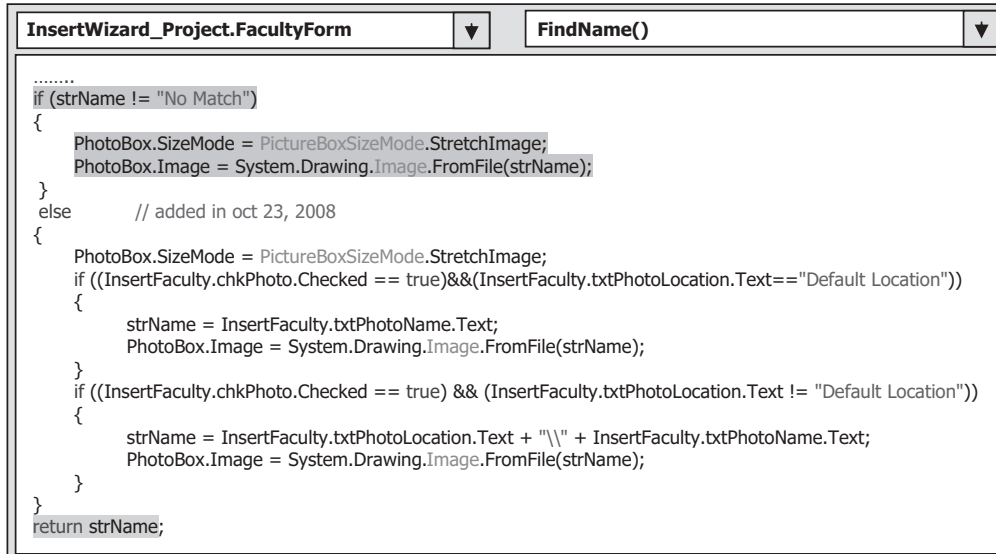
For the first situation, a `MessageBox` will be displayed to indicate that no faculty photo is found for the selected faculty when the Select button is clicked and the validation process is executed. In the second case, the faculty photo file is located at the folder in which the executable file of our current project, `InsertWizard Project.exe`, is stored. In this application, it is `C:\Chapter 6\InsertWizard Solution\InsertWizard Project\bin\Debug`. The third situation indicates that the faculty photo is located at a specified location.

Based on the analysis above, we only need to modify the codes in the selection branch part of this method, which means that no matched faculty photo file can be found for the selected faculty. This makes sense since the photo file of the new inserted faculty has not been added into the `Switch ... Case` block.

Open this method and browse to the selection branch part. Perform the following modifications to this part, which are shown in Figure 6.21.

Let's take a look at this piece of modified code to see how it works.

- A. An `else` branch is added under the `if` block to handle the situation in which no matched faculty photo can be found.
- B. The stretch mode, `StretchImage`, is assigned to the `SizeMode` property of the PhotoBox to adjust the size of the faculty photo and make it match the size of the PhotoBox.
- C. If the second situation we discussed above occurs, the faculty photo file is located at the default location and the name of the faculty photo file, which is stored in the `txtPhoto`



```

InsertWizard_Project.FacultyForm FindName()
.....
if (strName != "No Match")
{
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage;
    PhotoBox.Image = System.Drawing.Image.FromFile(strName);
}
else // added in oct 23, 2008
{
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage;
    if ((InsertFaculty.chkPhoto.Checked == true) && (InsertFaculty.txtPhotoLocation.Text == "Default Location"))
    {
        strName = InsertFaculty.txtPhotoName.Text;
        PhotoBox.Image = System.Drawing.Image.FromFile(strName);
    }
    if ((InsertFaculty.chkPhoto.Checked == true) && (InsertFaculty.txtPhotoLocation.Text != "Default Location"))
    {
        strName = InsertFaculty.txtPhotoLocation.Text + "\\\" + InsertFaculty.txtPhotoName.Text;
        PhotoBox.Image = System.Drawing.Image.FromFile(strName);
    }
}
return strName;

```

Figure 6.21 Modifications to the FindName method.

Name textbox on the Insert Faculty Form window, is assigned to the string variable strName and the FromFile() method is executed to display this photo.

- D.** In the third case we discussed above happened, the faculty photo file is located at a specified location that is stored in the txtPhotoLocation textbox on the Insert Faculty Form window. The concatenating operator + is used to combine that location with the name of the faculty photo file and assign this combination result to the string variable strName. Similarly, the FromFile() method is executed to display this photo.

Now we have completed the modifications to the Faculty Form. Next let's perform the modifications to the coding of the Insert Faculty Form window.

6.2.10.2 Modifications to Insert Faculty Form Window

The modifications to this part contain the following steps:

1. Declare a field-level nominal instance of the Faculty Form class, Faculty_Form. The so-called nominal instance is that we only declare this instance without assigning any memory space to this instance. This is different from creating a new instance with the keyword new. We need to assign the passed instance we did in Figure 6.20 to this Faculty_Form later. Note that the accessing mode of this instance should be Public since we need to access and use it via the Insert Faculty Form object during the project running.
2. Create a new method setFacultyForm() as we discussed in Figure 6.20 to reserve the current instance of the Faculty Form class, and assign it to the nominal instance Faculty_Form we declared in step 1.
3. Modify the codes inside the Back button's Click method to call the UpdateFaculty() method shown in Figure 6.19 to update the faculty members stored in the ComboName box on the Faculty Form window. This means that when the Back button is clicked, the data insertion is complete and the next job, data insertion validation, starts.

```

InsertWizard_Project.InsertFacultyForm  InsertFacultyForm()
public partial class InsertFacultyForm : Form
{
    private Dictionary<string, string> FacultyCollection = new Dictionary<string, string>();
    public FacultyForm Faculty_Form; // added in oct 23, 2008
    .....
    public void setFacultyForm(FacultyForm facultyForm) // added in oct 23, 2008
    {
        Faculty_Form = facultyForm;
    }
    .....
    private void cmdBack_Click(object sender, EventArgs e)
    {
        Faculty_Form.ComboName.Items.Clear(); // added in oct 23, 2008
        Faculty_Form.UpdateFaculty(); // added in oct 23, 2008
        Faculty_Form.ComboName.SelectedIndex = 0; // added in oct 23, 2008
        this.Hide();
    }
}

```

Figure 6.22 Modifications to the Insert Faculty Form code window.

Now open the code window of the Insert Faculty Form to perform those modifications listed above. The finished codes are shown in Figure 6.22.

All of the modified codes have been highlighted in bold. Let's have a close look at this piece of modified code to see how it works.

- A.** A field-level nominal instance of the Faculty Form class, `Faculty_Form`, is declared (Step 1 above).
- B.** A new method `setFacultyForm()` is created with the accessing mode as Public.
- C.** The passed current instance of the Faculty Form class, `facultyForm`, is assigned to that nominal instance `Faculty_Form`. Now the `Faculty_Form` is the current instance of the Faculty Form class.
- D.** In the Back button's Click method, the `Clear()` method is called to clean up all faculty members from the `ComboName` box in the Faculty Form window. This step is necessary because we will load and add the updated faculty members to this box, and some duplicated faculty members would be displayed in this box without this cleaning job executing first.
- E.** The `UpdateFaculty()` method in the Faculty Form window is called to load the updated faculty members from the database and add them into the `ComboName` box to update it.
- F.** To set up the `SelectedIndex` property to 0, the first faculty member stored in the `ComboName` box can be selected and displayed.

At this point, we have completed all coding modifications for the data insertion and the data validation tasks. Before we can run the project to test those tasks, first we must store the faculty photo file to the desired location, either to the default folder or to the user-selected folder, and this location must be identical with the location to be entered into the Photo Location textbox as the project runs.

In this application, we want to test the following two functionalities:

1. Insert a new faculty record with the faculty photo file located at the default location. You need to save the photo file of this faculty, `Mhamed.jpg`, into the default location: `C:\`

Chapter 6\InsertWizard Solution\InsertWizardProject\bin\Debug before the project runs.

2. Insert a new faculty record with the faculty photo file located at a specified location. You need to save the photo file of this faculty, Mhamed.jpg, into a specified location before the project runs. In this application, this specified location is C:\Chapter 6. Perform this photo file saving operation now.

When these are done, we can start our test. Let's do this test starting from the first one. We assume that the faculty photo file Mhamed.jpg has been saved in the default folder. Now click the Start Debugging button to run our project to test our data insertion and validation functionality.

After completing the login and selection process, click on the Insert button to open the Insert Faculty Form window; then check the Faculty Photo checkbox and enter the following information into the associated textboxes:

- Mhamed.jpg Photo Name textbox
- A56789 Faculty ID textbox
- Ali Mhamed Faculty Name textbox
- Associate Professor Title textbox
- MTC-222 Office textbox
- 750-330-1662 Phone textbox
- University of Main College textbox
- amhamed@college.edu Email textbox

Then click on the Insert button to insert this new faculty record into the database. Now the Insert button is disabled. To perform the data validation with the faculty photo displaying ability, click on the Back button to close the Insert Faculty Form window and open the Faculty Form window.

Now if you drag down the ComboName box, you can find that our new inserted faculty, Ali Mhamed, has been added into this box. To validate this data insertion, select this faculty and click on the Select button to try to retrieve back the inserted information. You need to note that we want to use the default location to store the faculty photo file, so we need to keep the content of the Photo Location textbox unchanged.

Immediately, the new inserted faculty information is retrieved back from the database and displayed in the associated textboxes after the Select button is clicked. The faculty photo is also displayed in the PhotoBox, which is shown in Figure 6.23.

Click on the Back and then the Exit buttons from the Faculty Form and the Selection Form windows to terminate the project.

Before we can continue to perform the next test, we must first delete that inserted new faculty record from our database since the Access does not allow duplicated records to be inserted into the database. Open our sample database CSE_DEPT.accdb and the Faculty table from our default folder to perform that deletion.

Next let's test to load the faculty photo from the specified folder, such as from the folder C:\Chapter 6. Before we can start the project, make sure that the faculty photo file, Mhamed.jpg, has been stored in C:\Chapter 6. Now start the project and open the



Figure 6.23 Testing status of the data validation with the faculty photo.

Insert Faculty Form window, check the Faculty Photo checkbox, and enter this path into the Photo Location textbox. Complete all other textboxes with the suitable new faculty information listed above, and then click on the Insert button to insert this new record into the database. Now click the Back button to close the Insert Faculty Form window. Select the new inserted faculty member, ALi Mhamed, from the ComboName box and click on the Select button from the Faculty Form window to validate this data insertion. Your running result should match the one shown in Figure 6.23.

Click on the Back and then the Exit buttons from the Faculty Form and the Selection Form windows to terminate the project. Our data insertion project is very successful! You can test to insert a new faculty record without the faculty photo if you like to do that. A MessageBox would be displayed to indicate that no faculty photo can be found for that situation.

A completed project, InsertWizard Project, is located at the folder `DBProjects\Chapter 6` located at the accompanying ftp site (see Chapter 1). In the next section, we will discuss how to insert new data into the database using the SQL Server database.

6.3 INSERT DATA INTO SQL SERVER DATABASE USING SAMPLE PROJECT SQLINSERTWIZARD

In this section, we discuss how to insert data into the SQL Server database. Basically there is no significant difference between inserting data into the Microsoft Access or SQL Server databases. For both kinds of databases, one can use either method provided by the TableAdapter class we discussed in the last section, such as the `TableAdapter.Insert()` or the `TableAdapter.Update()`, to insert new records into the database. A small difference between these two databases is that you cannot call the stored procedures to insert new records into the Microsoft Access database by using the TableAdapter [you can do that using the runtime objects method by executing the `ExecuteNonQuery()` method]; however, you can do that for the SQL Server database by using the TableAdapter. In this part, we first concentrate on data insertion using the two TableAdapter methods, and then we discuss how to insert data using the stored procedure method in the next part.

Because of the similarity between inserting data into the Microsoft Access database and inserting data into the SQL Server database, we only provide a quick review for inserting data into the latter. Only the differences of inserting data between two databases will be emphasized in this project.

As we did in the last section, to save time and space, we want to modify an existing project to perform this data insertion job. To do that, open an existing project named `SelectWizard Project` we developed in Chapter 5 (you can find this project from the folder `DBProjects\Chapter 5` located at the accompanying ftp site, see Chapter 1). In that project, an SQL Server database was used. An SQL Server 2005 Express SP2 database file `CSE_DEPT.mdf` has been set and connected to that project. Therefore, we don't need to perform any database connection and configuration job if we modify that project to create our new project to perform the data insertion in the next section.

6.3.1 Modify Existing Project to Get New Data Insertion Project

Open the Windows Explorer and create a new folder `Chapter 6` if you have not done that. Then browse to our project `SelectWizard Project` from the folder `DBProjects\Chapter 5` located at the accompanying ftp site (see Chapter 1) and copy this project to our new folder `Chapter 6`. Change the folder names of the solution and the project from `SelectWizard` to `SQLInsertWizard`; also change the project name from `SelectWizard.csproj` to `SQLInsertWizard.csproj` and then double-click on the `SQLInsertWizard.csproj` to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to `Project | SQLInsertWizard Project Properties` menu item to open the project's property window. Change the `Assembly name` from `SelectWizard` to `SQLInsertWizard` and the `Default namespace` from `SelectWizard` to `SQLInsertWizard`, respectively.
- Click on the `Assembly Information` button to open the `Assembly Information` dialog box, change the `Title` and the `Product` to `SQLInsertWizard`. Click on `OK` to close this dialog box.

Go to the `File | Save All` to save those modifications. Now we need to add a new graphic user interface named `InsertCourseForm` to our new project to perform the new course insertion for a selected faculty. However, before we can do that, first we need to modify the namespaces for some project-related configuration files in this project since we duplicated an existing project without changing the namespaces for those files. The possible errors you may encounter are that either you cannot open any graphical user interface such as `LogIn Form`, `Faculty Form`, or `Course Form` windows or you will have some compiling errors when you compile the project now. All of these errors are caused by the nonmatched namespaces between the existing project and our new project since we changed the name of the project. In C#, each project should be located at a different namespace, with the name of the namespace equaling the project's name. Since we modified the project's name but without touching the namespace inside those files, some nonmatched namespace compiling errors appear.

To fix this nonmatched namespace error, we need to perform the following modifications:

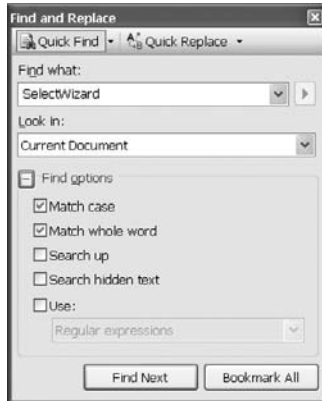


Figure 6.24 Find and Replace dialog box.

1. Change the namespace from `SelectWizard` to `SQLInsertWizard` for each source file. To do that, open the source file for each class such as `LogIn.cs`, `Grid.cs`, `Faculty.cs`, `Course.cs`, `Selection.cs`, and `Student.cs` by opening the Code Window of each form; then change the namespace to `SQLInsertWizard`.
2. Change the namespace from `SelectWizard` to `SQLInsertWizard` for each form's Designer.cs file. To do that, try to open the following Form window such as `LogIn.cs`, `Grid.cs`, `Faculty.cs`, and `Course.cs` by opening the Designer Window (graphical user interface). Then click on the link `Go to code` from the Error Message Window to open the Code Window for the Designer Window. To identify those nonmatched namespaces, go to `Edit|Quick Find` menu item to open the Find and Replace dialog box, which is shown in Figure 6.24.

Enter `SelectWizard` into the Find what box and select `Current Document` from the Look in box. Expand the Find options checkbox to check both `Match case` and `Match whole word` checkboxes. Click on the `Find Next` button to try to find those nonmatched namespaces. Click on the `Quick Replace` tab and then enter `SQLInsertWizard` into the Replace with box when a nonmatched namespace is found. Then you can click on either `Replace` button or `Replace All` button to do these replacements. Perform this replacement for all of those forms listed above. Now if you try to open each Designer window, it can be opened without a problem.

3. Change the namespace from `SelectWizard` to `SQLInsertWizard` for the project file `Program.cs`. To do that, open that file by double-clicking on it, and then replace the `SelectWizard` that is located at the top line of this program file by `SQLInsertWizard`. Now if you try to build this new project, everything is fine and all compiling errors are gone.

Now we are ready to add one more graphic user interface, `InsertCourseForm`, to this new project to perform the new course insertion functionality for a selected faculty.

6.3.2 Create New Form Window to Insert Data for Course Form

In our new project `SQLInsertWizard`, the Course Form window contains a button named `Insert`. We want to use that button to insert a new course record into the `Course` table in the database. To do that, we need to create a new form window named `InsertCourseForm`

into this new project, and this new form should be triggered by the clicking of the Insert button from the Course Form window as the project runs.

The functionality of this Insert Course Form is: As the project runs, after the user has correctly finished the login process and selected the Course Information from the Selection Form, the Course Form window will be displayed. When the user clicks on the Insert button, the Insert Course Form window will appear. This form allows users to use two different methods to insert data into the database: either the `TableAdapter.Insert()` method or the `TableAdapter.Update()` method. The form also allows users to enter all pieces of new course information into the appropriate textboxes for the selected faculty based on the faculty name. By clicking on the Insert button, a new course record related to the selected faculty is inserted into the database. However, if the user wants to reenter those pieces of information before finishing this insertion, the Cancel button can be used and all information entered will be erased. The Select button is used to validate this data insertion by retrieving the new inserted data. The Back button is used to allow users to return to the Selection Form to perform other data operations.

Go to the **Project\Add Windows Form** menu item to open the **Add New Item** dialog box. Select the item **Windows Forms** from the **Categories** box and **Windows Form** from the **Templates** box. Enter `Insert Course Form.cs` into the **Name** box as the name for this new form. Then click the **Add** button to add this form into our project.

Add the items shown in Table 6.3 into this form.

In addition to the Form's properties shown in Table 6.3, the following properties of the form should also be set up:

Table 6.3 Objects in the Insert Course Form window

Type	Name	Text	TabIndex	DropDownStyle
GroupBox	GroupBox1	Method and Faculty Information	0	
Label	Label1	Insert Method	0.0	
ComboBox	ComboMethod		0.1	DropDownList
Label	Label2	Faculty Name	0.2	
ComboBox	ComboName		0.3	DropDownList
GroupBox	GroupBox2	New Course Information	1	
Label	Label3	Faculty ID	1.0	
TextBox	txtFacultyID		1.1	
Label	Label4	Course ID	1.2	
TextBox	txtCourseID		1.3	
Label	Label5	Course Title	1.4	
TextBox	txtCourse		1.5	
Label	Label6	Schedule	1.6	
TextBox	TxtSchedule		1.7	
Label	Label7	Classroom	1.8	
TextBox	txtClassroom		1.9	
Label	Label8	Credits	1.10	
TextBox	txtCredits		1.11	
Label	Label9	Enrollment	1.12	
TextBox	txtEnrollment		1.13	
Button	cmdInsert	Insert	2	
Button	cmdSelect	Select	3	
Button	cmdCancel	Cancel	4	
Button	cmdBack	Back	5	
Form	InsertCourseForm	CSE DEPT Insert Course Form		

Figure 6.25 Completed Insert Course Form window.

- AcceptButton: **cmdInsert** (select the Insert button as the default button)
- StartPosition: **CenterScreen** (locate the form in the center of the screen when it runs)

Your finished Insert Course Form should match the one shown in Figure 6.25.

The order of the TabIndex values we created in Table 6.3 makes the Faculty Name combobox the default one, and it allows users to directly select the faculty name without click on that box first.

Next we need to trigger and connect the Design Tools provided by Visual Studio, and we will use them to facilitate our new course insertion query in this project. Two methods can be used to perform this Design Tools' trigger and connection.

6.3.3 Trigger and Connect to Visual Studio Design Tools

The first method is to perform a data binding between objects in the Insert Course Form window and data columns in the associated data tables in the database. As long as any object has been bound to a column in a selected table in our database, all Design Tools, such as DataSet, BindingSource, and TableAdapter, can be triggered and connected to our current form window. In this study, two textboxes, Faculty ID and Course ID, in the Insert Course Form window and two associated columns, faculty_id in the Faculty table and course_id in the Course table, will be bound together to trigger and connect the Design Tools to help us perform this data insertion.

As we know, there is no faculty_name column in the Course table, and the only relationship between the faculty and course in the Course table is the faculty_id. Therefore we need to perform two queries to do this course insertion: (1) get the faculty_id from the Faculty table based on selected faculty name, and (2) insert a new course record based on the faculty_id obtained from query (1).

Now let's perform the data binding for these two textboxes in the Insert Course Form window with two related columns in the two tables to trigger Design Tools. Click on the Faculty ID textbox from the Insert Course Form window to select it and go to the

DataBindings property, expand it to the Text subproperty, and perform the following steps to complete this binding:

1. Expand the Other Data Sources item.
2. Expand the Project Data Sources item.
3. Expand the CSE_DEPTDataSet item.
4. Expand the Faculty table.
5. Select the faculty_id.

Then click on the Course ID textbox to perform the same five steps to perform another data binding. The only difference between this binding and the last one is that after step 3, you need to expand the Course table and select the course_id from that table.

Immediately you can find that five Design Tools have been added into this form, which are cSE_DEPTDataSet, facultyTableAdapter, facultyBindingSource, courseTableAdapter, and courseBindingSource. One point you need to note is that all of them are objects, not classes, and we can directly use them without needing to instantiate them. We need to use them to complete this new course insertion query.

The second method used to trigger and connect those Design Tools in our current form window is to create instances based on the classes provided by Design Tools. To do that, open the Insert button's Click method from the Insert Course Form window, and enter the following codes to create three instances:

```
CSE_DEPTDataSet cSE_DEPTDataSet = new CSE_DEPTDataSet();

CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
facultyTableApt =
    new CSE_DEPTDataSetTableAdapters.FacultyTableAdapter();

CSE_DEPTDataSetTableAdapters.CourseTableAdapter courseTableApt =
    new CSE_DEPTDataSetTableAdapters.CourseTableAdapter();
```

Now we are ready to build our new course insertion query. However, before we can do that using the TableAdapter Query Configuration Wizard, let's first do some coding for the project initialization and data validation.

6.3.4 Project Initialization and Validate Data Before Data Insertion

Just as we did for the last sample project, a data validation must be performed before the course information can be inserted into the database. To save time and space, we use a string array to store all course information. In total we have seven pieces of course information: faculty_id, course_id, course title, course schedule, course classroom, course credits, and course enrollment, and all of these pieces of information should be entered by the user into seven textboxes as the project runs. Also the combobox Faculty Name should be initialized by adding all faculty members to allow users to make the selection from this box as the project runs, too.

To do these jobs, open the constructor of the Insert Course Form, and then enter the codes into this constructor shown in Figure 6.26.

```

SQLInsertWizard.InsertCourseForm | InsertCourseForm()
public partial class InsertCourseForm : Form
{
A   string[] CourseInfo = new string[7];
    public InsertCourseForm()
    {
B       InitializeComponent();
        ComboName.Items.Add("Ying Bai");
        ComboName.Items.Add("Satish Bhalla");
        ComboName.Items.Add("Black Anderson");
        ComboName.Items.Add("Steve Johnson");
        ComboName.Items.Add("Jenney King");
        ComboName.Items.Add("Alice Brown");
        ComboName.Items.Add("Debby Angles");
        ComboName.Items.Add("Jeff Henry");
        ComboName.SelectedIndex = 0;
C       ComboMethod.Items.Add("TableAdapter Insert");
        ComboMethod.Items.Add("TableAdapter Update");
        ComboMethod.SelectedIndex = 0;
D       txtCourseID.DataBindings.Clear();
        txtFacultyID.DataBindings.Clear();
    }
}

```

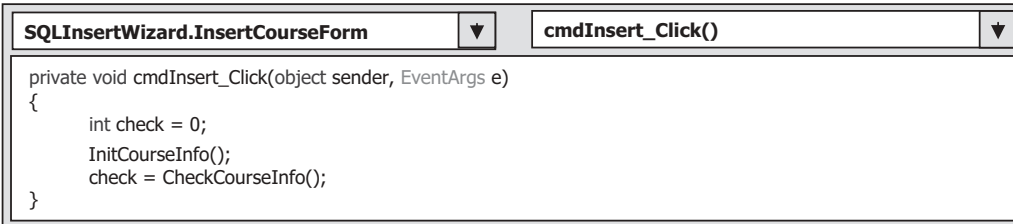
Figure 6.26 Coding for the constructor of the InsertCourseForm.

Let's take a look at this piece of code to see how it works.

- A.** First, we create a string array `CourseInfo` with the upper bound of 6, which means that this array contains 7 items with the index starting from 0. This array is used to store seven pieces of course information entered by the user to the seven textboxes.
- B.** All faculty members are added into the combobox `ComboName` by executing the `Add()` method, and the first item is selected as the default faculty member.
- C.** The first method, `TableAdapter Insert`, in the `Combo Method` box is selected as the default by setting the `SelectedIndex` property to 0 (the index of the combobox also starts from 0).
- D.** These two instructions are used to clean up the binding contents in two textboxes, `Faculty ID` and `Course ID`. As you know, in order to use `Design Tools` to perform this new course insertion, we performed the data binding for them. One problem for those data bindings is that the bound columns would be displayed in these two textboxes as the project runs. In order to make them blank to allow users to enter new course information to these two textboxes, we prefer to clean them up. This cleaning will remove the bound relationships without affecting the `Design Tools`.

Now let's develop the codes for the data validation before performing the data insertion. This data validation makes sure that all textboxes that contain the course information are nonempty. This means that you need to fill out all textboxes with a certain piece of course-related information and the project does not allow an empty textbox or a blank piece of information to be inserted into the database. If you try to intentionally keep some columns in the `Course` table empty, you need to place a `NULL` into the associated textbox.

Open the `Insert` button `Click` method by double-clicking on the `Insert` button from the `Insert Course Form` window. Enter the codes shown in [Figure 6.27](#) into this method.

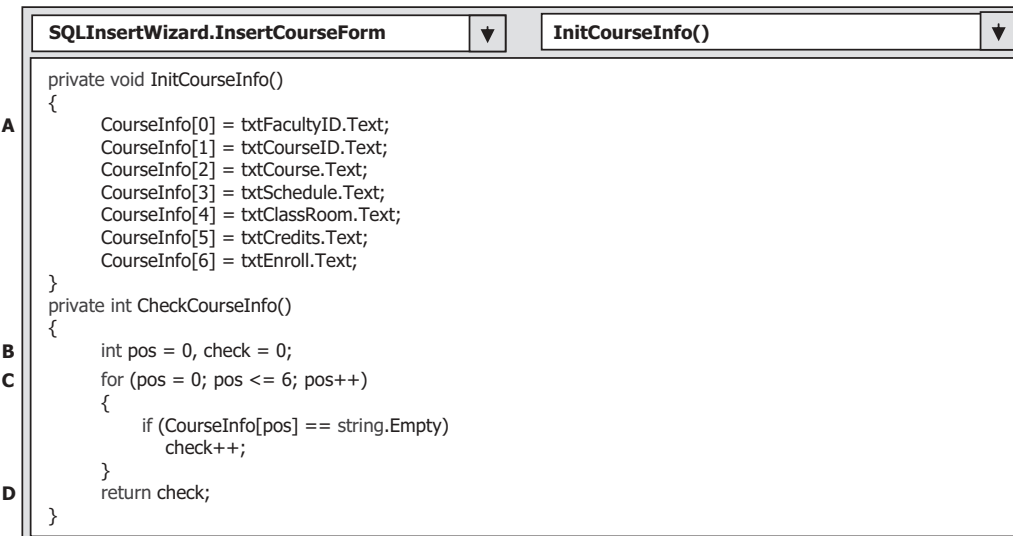


```

SQLInsertWizard.InsertCourseForm  cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
    int check = 0;
    InitCourseInfo();
    check = CheckCourseInfo();
}

```

Figure 6.27 Coding for the Insert button Click method.



```

SQLInsertWizard.InsertCourseForm  InitCourseInfo()
private void InitCourseInfo()
{
A   CourseInfo[0] = txtFacultyID.Text;
   CourseInfo[1] = txtCourseID.Text;
   CourseInfo[2] = txtCourse.Text;
   CourseInfo[3] = txtSchedule.Text;
   CourseInfo[4] = txtClassRoom.Text;
   CourseInfo[5] = txtCredits.Text;
   CourseInfo[6] = txtEnroll.Text;
}
private int CheckCourseInfo()
{
B   int pos = 0, check = 0;
C   for (pos = 0; pos <= 6; pos++)
   {
       if (CourseInfo[pos] == string.Empty)
           check++;
   }
D   return check;
}

```

Figure 6.28 Coding for two user-defined methods.

The functionality of this piece of code is very simple. First, a user-defined method `InitCourseInfo()` is called to assign each textbox's content to the associated string item in the string array, and then another user-defined method `CheckCourseInfo()` is executed to make sure that all textboxes are filled with a piece of course-related information and no empty textbox exists. The code for these two methods is shown in Figure 6.28.

The functionality of these two methods is:

- A.** The so-called initialization of the course information assigns each textbox's content to the associated string item in the string array `CourseInfo`. In this way, it is easier for us to check those textboxes to make sure that no textbox is empty later.
- B.** Two local variables `pos` and `check` are created. The first one is used as a loop counter for the `for` loop, and the second is used to store the checking result for this method.
- C.** A `for` loop is utilized to scan all elements in the string array, which is equivalent to scan all textboxes, to check whether any of them is empty. The checking result `check` will be increased by one if this situation happened.
- D.** Finally, the checking result is returned to the calling method. A nonzero value means that there are some empty textboxes in this Course Form window, and the value of this variable equals the number of empty textboxes checked.

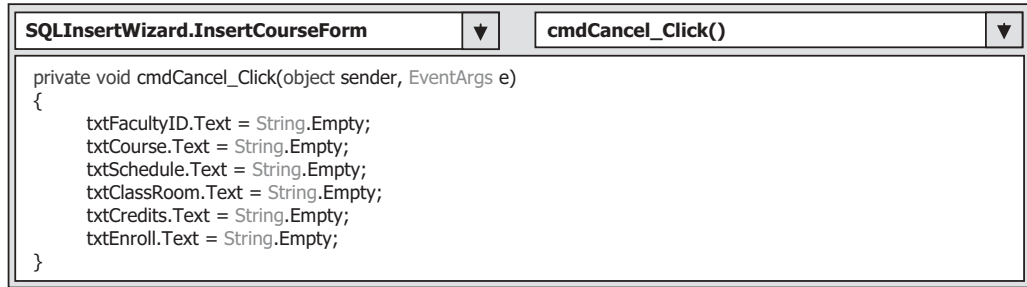


Figure 6.29 Coding for the Cancel button's Click method.

The other coding jobs for the project initialization are the coding for the Cancel and the Back command buttons. When the Back button is clicked, the current form should be closed and the project needs to be returned to the Course Form to perform the validation for that new course insertion. The coding for this button is easy. Open this method and enter `this.Close();`. Yes, that's it!

The functionality of the Cancel button is to clean up most textboxes' contents to allow users to reenter all course-related information into those textboxes. Open the Cancel button's Click method by double-clicking on that button and enter the codes shown in Figure 6.29 into this method.

The coding for this method is straightforward and easy to understand. All textboxes' contents, except the Course ID, are cleaned up to allow users to reenter new course information. One reason we keep the Course ID textbox's contents unchanged is that this piece of information will be used later for data validation purposes since we need to retrieve the new inserted course record from the database based on the Course ID. Another reason is that we try to stop the TextChanged event from occurring to reenble the Insert button (refer to Section 6.2.8 and Figure 6.36).

Now we can start to configure the TableAdapters to build our inserting data query.

6.3.5 Configure TableAdapter and Build Data Insertion Query

Recall that when we built our sample database CSE_DEPT in Chapter 2, there is no faculty name column in the Course table, and the only relationship that existed between the Faculty and the Course tables is the `faculty_id`, which is a primary key in the Faculty table but a foreign key in the Course table. As the project runs and the Insert Course Form window appears, the user needs to insert a new course record based on the faculty name, not the faculty ID. So for this new course record insertion, we need to perform two queries with two tables: First, we need to make a query to the Faculty table to get the `faculty_id` based on the faculty name selected by the user, and second we can insert a new course record based on the `faculty_id` we obtained from our first query. These two queries belong to two TableAdapters: the FacultyTableAdapter and the CourseTableAdapter. Now let's create these two query functions under two TableAdapters.

Recall that in Section 5.14 we built a query function under the FacultyTableAdapter, `FindFacultyIDByName()`. In this section, we don't need to redo this operation and we can use that query function directly from this project. We do need to create the second query function to insert new course records based on the selected `faculty_id` we obtained from the first query function `FindFacultyIDByName()`.

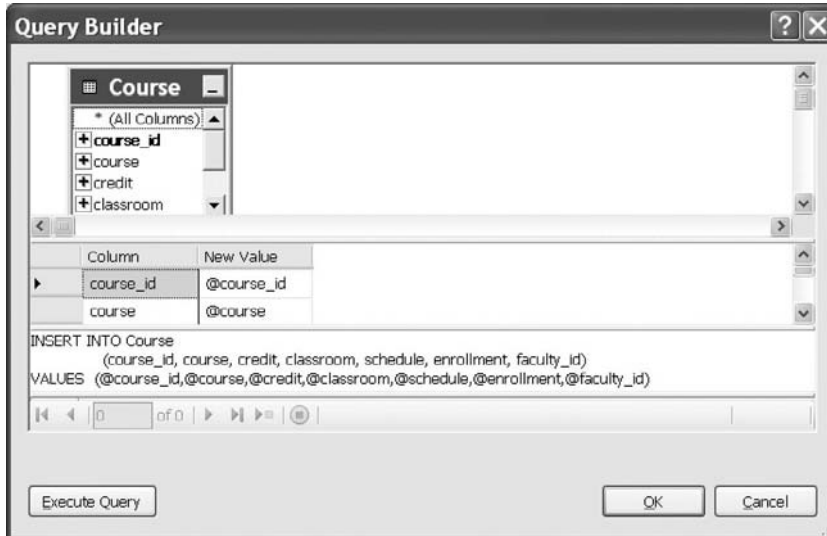


Figure 6.30 Query Builder dialog box.

Open the Data Source window by clicking on the **Data>Show Data Sources** menu item from the menu bar. Then right-click on any place inside this window and select the **Edit DataSet with Designer** item to open the **DataSet Designer Wizard**. Right-click on the last line of the **Course** table and choose the **Add!Query** item to open the **TableAdapter Query Configuration Wizard**. Keep the default item **Use SQL statements** selected and click on the **Next** button to go to the next window. Select the **INSERT** item by checking this radio button and click on **Next** again to continue. Click on the **Query Builder** button on the next window to open the **Query Builder** dialog box, which is shown in Figure 6.30.

The default **INSERT** statement is matched to our data insertion command, so we'd like to keep it as our query command and click on the **OK** button to go to the next window.

On the next window, the **Query Builder** needs us to confirm this query function. Two queries are displayed in this window. The first one is an **INSERT** statement and it is what we need for our data insertion. However, the second one is the **SELECT** statement and we do not need this one for this data insertion. Therefore, highlight the second statement and delete it from this window. Click on the **Next** button to continue.

In the next window, we need to enter the name for our query function. Enter **InsertCourse** into the name box and click on the **Next** button to go to the next window, and then click on the **Finish** button to complete this **Query Builder** dialog box and exit this process.

Now we have finished the configuration and the query building for the **Course TableAdapter**. Next, we need to develop the codes to execute this data insertion function.

6.3.6 Develop Codes to Insert Data Using **TableAdapter.Insert Method**

Open the **Insert Course Form** window and double-click on the **Insert** button to open its **Click** method, and then enter the codes shown in Figure 6.31 into this method. The

```

SQLInsertWizard.InsertCourseForm  cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
A   int check = 0, intInsert = 0;
    InitCourseInfo();
B   check = CheckCourseInfo();
    if (check == 0)
    {
C       facultyTableAdapter.ClearBeforeFill = true;
D       string strFacultyID = facultyTableAdapter.FindFacultyIDByName(ComboName.Text);
E       txtFacultyID.Text = strFacultyID;
        if (ComboMethod.Text == "TableAdapter Insert")
        {
            intInsert = courseTableAdapter.InsertCourse(txtCourseID.Text, txtCourse.Text, txtClassRoom.Text,
                txtSchedule.Text, int.Parse(txtEnroll.Text), txtFacultyID.Text, double.Parse(txtCredits.Text));
        }
F       else
        {
            // insert new course using the TableAdapter.Update() method...
        }
G       if (intInsert != 0) // data insertion is successful
        {
            cmdCancel.PerformClick(); // clean up all faculty information
            cmdInsert.Enabled = false; // disable the Insert button
        }
H       else
        {
            MessageBox.Show("The course insertion is failed");
            cmdInsert.Enabled = true;
        }
I       }
    }
    else
        MessageBox.Show("Fill all Course Information box, enter a NULL for blank column");
}

```

Figure 6.31 Coding for the Insert button Click method.

codes we developed in the previous section for this method have been highlighted with shading.

Let's take a look at this piece of code to see how it works.

- A.** A local integer variable `intInsert` is created, and it is used to hold the returning status of calling the second query function `InsertCourse()`.
- B.** After the methods `InitCourseInfo()` and `CheckCourseInfo()` are executed, the validation result for all textboxes is returned and stored in the local variable `check`. A returned value of zero means that this validation is successful and no textbox is empty. Then the `ClearBeforeFill` property of the `TableAdapter` is set to `true` to clean up the `TableAdapter`'s associated data table, the `Faculty` table in the `DataSet`, to make it ready to be filled with new records.
- C.** The first query function `FindFacultyIDByName()`, which we built in Section 5.14, in Chapter 5, is used to return a single data value—`faculty_id`—from the `Faculty` table based on the faculty name selected by the user from the combobox `ComboName` as the project runs. A string variable `strFacultyID` is created, and it will be used to hold the returned `faculty_id` obtained from the first query function.
- D.** The returned `faculty_id` value is assigned to the `Faculty ID` textbox. This step is necessary. Later you will find that as the project runs, only the faculty name appears and can

be selected by the user to perform this data insertion since the user has no knowledge about the `faculty_id`. Therefore, when this new course insertion is performed, a `NULL` must be filled into the `faculty_id` textbox in order to make this data insertion operate properly.

- E. If the user selected the `TableAdapter.Insert()` method to perform this data insertion, the second query function `InsertCourse()`, which we just built in the last section, is executed to insert new course record into the `Course` table in the database. Note that not only the order of inserted parameters in this `InsertCourse()` query function, but also the data type of each parameter, must be identical with those of the inserted parameters in the same query function `InsertCourse()` we built in the last section. Refer to Figure 6.30 to confirm the order and data types for those parameters if you are not sure. Here, the `Parse()` method is used to convert the `txtEnroll` and the `txtCredit` from string to the integer and the double data types, respectively.
- F. If the user selected the `TableAdapter.Update()` method to perform this data insertion, the associated codes that will be developed in the next section will be executed to first add the new course record into the `Course` table in the `DataSet`; and then it will be updated into the `Course` table in the database.
- G. The second query function `InsertCourse()` will return an integer to indicate the execution status of calling this function. Actually, the returned integer's value equals the number of records that have been successfully added or inserted into the database. A returned nonzero means that this insertion is successful and a new record has been inserted into the database. If that occurred, we need to clean up the contents of all textboxes (except the Faculty Name) by executing the `PerformClick()` method. Also in order to avoid multiple insertions for the same record, the `Insert` button is disabled by resetting its `Enabled` property to `false`.
- H. A returned zero means that no record has been inserted into the database, and this data insertion has failed. If that case occurs, a message box with a warning message is displayed. The `Insert` button is enabled again to allow users to perform another insertion.
- I. If the `CheckCourseInfo()` method returns a nonzero value, it means that the validation for all textboxes has failed and there are some unfilled textboxes. A message box with a warning message is displayed to ask users to fill all textboxes.

Now let's test this data insertion by running our project. We have two ways to do it. One way is to trigger the `Insert Course Form` window by clicking on the `Insert` button from the `Course Form` window. To start the project in this way, we must start from the `LogIn` form, and then continue to the `Selection Form`, and to the `Course Form` by selecting the `Course Information` item from the `Selection Form` window. Another way is simple. We can start the project directly from the `Insert Course Form` window. To start the project in this way, one needs to modify the startup object located inside the `Main()` method. To do that, double-click on the `Program.cs` item from the `Solution Explorer` window, and then change the startup object by replacing

```
Application.Run(new LogInForm());
```

with

```
Application.Run(new InsertCourseForm());
```

To start the project in the first way, we need to do a little modification to the `Course Form`. First, we need to create a field-level instance of the `InsertCourseForm` class,

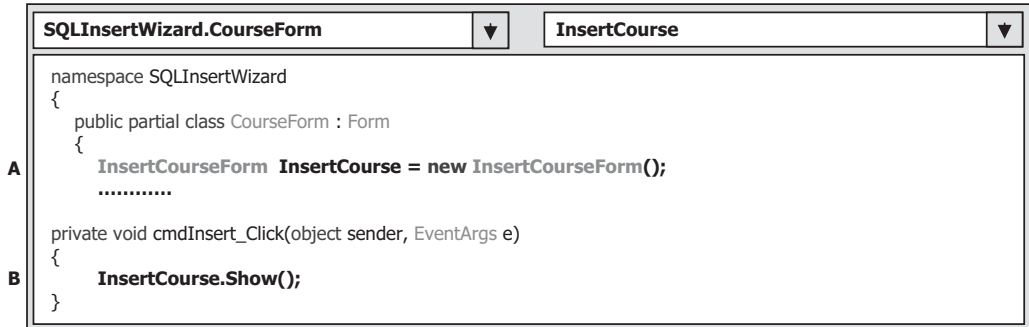


Figure 6.32 Coding for the Insert button's Click method in the Course Form.

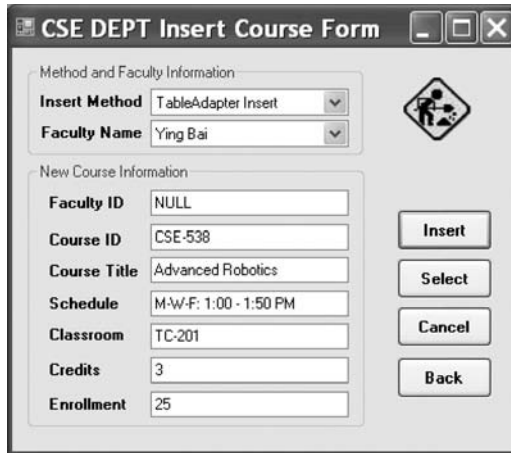


Figure 6.33 Running status of the project.

InsertCourse, and then add some codes to the Insert button's Click method to display the InsertCourseForm window as the user clicks on this Insert button from the Course Form window. Open the Course Form window, create a field-level instance of the class InsertCourseForm, which is shown in step A in Figure 6.32, and then double-click on the Insert button to open its Click method. Enter the code shown in step B in Figure 6.32 into this method.

As the Insert button is clicked, a new InsertCourseForm object, InsertCourse, is created and the Show() method is called to display this form window.

Now let's run the project using the first method to test our data insertion by clicking on the Start Debugging button. The running status of the project is shown in Figure 6.33.

Enter the suitable username and password, such as jhenry and test, for the LogIn form, and select the Course Information item from the Selection Form window to open the Course Form window. Then click on the Insert button to open the Insert Course Form window.

At this time, we can only test the first data insertion method—TableAdapter.Insert()—since we have not developed the codes for the second method. Select a faculty

from the Faculty Name combobox, and enter the new course information shown in Figure 6.33 into the associated textbox. We may not remember the Faculty ID for the selected faculty; therefore, a NULL is entered at this moment (of course, you can enter the selected `faculty_id` to the Faculty ID textbox if you remember it). Click on the Insert button; all textboxes that contain the new course information are cleaned up except the Course ID textbox. We try to keep the content of this Course ID textbox unchanged to avoid a `TextChanged` event occurring, and because we want to avoid mistakenly making multiple insertions of the same data. Click on the **Back**, **Back**, and **Exit** buttons from the Insert Course Form, Course Form, and Selection Form windows to terminate the project. We will validate this data insertion in the next section.

Next let's develop the codes to insert new course data using the second method, `TableAdapter.Update()`.

6.3.7 Develop Codes to Insert Data Using `TableAdapter.Update` Method

Open the Insert button's Click method from the Insert Course Form window and add the codes shown in Figure 6.34 into this method. Since we have already developed some codes for the first data insertion method in the previous section, all of those developed codes are highlighted with shading.

Let's take a look at this piece of code to see how it works.

- A. First, we need to declare a new object of the `DataRow` class. Each `DataRow` object can be mapped to a real row in a data table. Since we are using the `DataSet` to manage all data tables in this project, the `DataSet` must be prefixed before the `DataRow` object. Also we need to create a row in the Course data table. The `CourseRow` is selected as the `DataRow` class.
- B. Next, we need to create a new object of the `NewCourseRow` class.
- C. A user-defined method `InsertCourseRow()` is called to add all information about the new inserting course, which is stored in seven textboxes, into this new created `DataRow` object. The functionality and the codes for this user-defined method is explained below. This method returns a completed `DataRow` in which all information about the new inserted course record has been added.
- D. The completed `DataRow` is added into the Course table in our `DataSet`. Note that adding a new record into the data table in the `DataSet` is nothing to do with adding a new record into the data table in the database. The data tables in the `DataSet` are only mappings of those real data tables in the database. To add this new record into the database, one needs to perform the next step.
- E. The `TableAdapter's Update()` method is executed so that this new record is added into the real database. As we mentioned before, you can control the amount of data to be added into the database by passing the different arguments. Here we only want to add one new record into the Course table; therefore, a data table is passed as the argument. This `Update()` method supposes returning an integer value to indicate whether this updating has successful or not. The value of this returned integer is equal to the number of rows that have been successfully added into the database. A returned value of zero means that this updating has failed since no new row has been added into the database.

```

private void cmdInsert_Click(object sender, EventArgs e)
{
    int check = 0, intInsert = 0;
    CSE_DEPTDataSet.CourseRow newCourseRow;
    InitCourseInfo();
    check = CheckCourseInfo();
    if (check == 0)
    {
        facultyTableAdapter.ClearBeforeFill = true;
        string strFacultyID = facultyTableAdapter.FindFacultyIDByName(ComboName.Text);
        txtFacultyID.Text = strFacultyID;
        if (ComboMethod.Text == "TableAdapter Insert")
        {
            intInsert = courseTableAdapter.InsertCourse(txtCourseID.Text, txtCourse.Text, txtClassRoom.Text,
                txtSchedule.Text, int.Parse(txtEnroll.Text), txtFacultyID.Text, double.Parse(txtCredits.Text));
        }
        else
        {
            newCourseRow = cSE_DEPTDataSet.Course.NewCourseRow();
            newCourseRow = InsertCourseRow(ref newCourseRow);
            cSE_DEPTDataSet.Course.Rows.Add(newCourseRow);
            intInsert = courseTableAdapter.Update(cSE_DEPTDataSet.Course);
        }
        if (intInsert != 0) // data insertion is successful
        {
            cmdCancel.PerformClick(); // clean up all faculty information
            cmdInsert.Enabled = false; // disable the Insert button
        }
        else
        {
            MessageBox.Show("The course insertion is failed");
            cmdInsert.Enabled = true;
        }
    }
    else
        MessageBox.Show("Fill all Course Information box, enter a NULL for blank column");
}

```

Figure 6.34 Modified coding for the Insert button's Click method.

The codes for the user-defined method `InsertCourseRow()` are shown in Figure 6.35. Open the code window of the Insert Course Form and enter the codes shown in Figure 6.35 into this method.

Let's see how this piece of codes works.

- A.** The method `InsertCourseRow()` needs to return a completed `DataRow` object, and the returned data type is indicated at the beginning of the method header after the accessing mode `private`. The argument is also a `DataRow` object, but it is a new created blank `DataRow` object. The data type of the argument is very important. Here we used a reference mode for this argument. The advantage of using this mode is that the passed variable is an address of the `DataRow` object. Any modification to this object, such as adding new elements to this `DataRow`, is permanent, and the modified object can be completely returned to the calling method.
- B.** Seven pieces of new course information stored in the associated textboxes are added into this new `DataRow` object, that is, added into a new row of the Course table in the `DataSet`. Note that the data type of each inserted item must be identical with the data type of each

```

SQLInsertWizard.InsertCourseForm | InsertCourseRow()
A private CSE_DEPTDataSet.CourseRow InsertCourseRow(ref CSE_DEPTDataSet.CourseRow courseRow)
B {
  courseRow.faculty_id = txtFacultyID.Text;
  courseRow.course_id = txtCourseID.Text;
  courseRow.course = txtCourse.Text;
  courseRow.schedule = txtSchedule.Text;
  courseRow.classroom = txtClassRoom.Text;
  courseRow.credit = double.Parse(txtCredits.Text);
  courseRow.enrollment = int.Parse(txtEnroll.Text);
C   return courseRow;
  }

```

Figure 6.35 Coding for the user-defined InsertCourseRow method.

```

SQLInsertWizard.InsertCourseForm | txtCourseID_TextChanged
private void txtCourseID_TextChanged(object sender, EventArgs e)
{
  cmdInsert.Enabled = true;
}

```

Figure 6.36 Coding for the TextChanged method of the Course ID textbox.

associated column in the Course table. The Parse() method is utilized to perform the conversion between the string and different other built-in data types for columns `credit` and `enrollment`.

- C. Finally, this completed DataRow object is returned to the calling method. Another advantage of using this reference value is that we do not need to create another local variable as the returned variable; instead we can directly use this passed argument as the returned data.

We have one more step to go to finish this data insertion process. As we mentioned, after one new data is inserted into the database, the Insert button will be disabled to avoid the multiple insertions of the same data. In order to reenale this button, you have to change the content of the Course ID textbox, and this means that you want to insert a new course record. To make this happen, we need to use the TextChanged event of the Course ID textbox to trigger its TextChanged method to enable the Insert button. To do that, open the Insert Course Form window and double-click on the Course ID textbox to open its TextChanged method. Enter one line of code into this method, as shown in Figure 6.36.

Now we have completed developing two methods to perform the data insertion operation. You can run the project to test the second data insertion method with the following new course information (the selected Faculty Name is Ying Bai):

- Faculty ID NULL
- Course ID CSE-566
- Course Title Introduction to Fuzzy Logic
- Schedule TH: 1:30–2:45 PM

- Classroom TC-309
- Credits 3
- Enrollment 20

Next, we need to discuss how to validate these data insertions to confirm that these new course record insertions are successful.

6.3.8 Use Select Button in Course Form to Perform Data Validation

Using this method to validate the new inserted data is very simple and efficient. The basic idea of this method is to use the query function `FillByFacultyID()`, which we developed in Figure 5.59, to retrieve new inserted records from the Course table and display them in a CourseList box in the Course Form window. Since that query function has been developed, we can directly use it. The Select button's Click method contains two query functions: `FindFacultyIDByName()` and `FillByFacultyID()`. We need both functions to validate our new inserted record.

To use this method to validate the new inserted data, run the project and go to the Course Form window. To validate new inserted records, select the desired faculty from the Faculty Name combobox (such as Ying Bai) and click on the Select button to retrieve all course records and display them in the CourseList box. You can find that two new inserted courses, CSE-538 and CSE-566, have been successfully inserted into our database and displayed in the Course Listbox, which is shown in Figure 6.37. By clicking on each of these new records, all pieces of course-related information are displayed in the associated textboxes, as shown in Figure 6.37.

At this point, we completed inserting new data into the SQL Server database using the two methods. Next, we want to discuss how to insert new records into the database

The screenshot shows a Windows application window titled "CSE DEPT Course Form". The window is divided into several sections:

- Name and Method:** A section containing two dropdown menus. "Faculty Name" is set to "Ying Bai" and "Query Method" is set to "TableAdapter Method".
- Buttons:** Three buttons are located to the right of the dropdowns: "Select", "Insert", and "Back".
- Course List:** A list box containing the following course IDs: CSC-132B, CSC-234A, CSE-434, CSE-438, CSE-538, and CSE-566. The course "CSE-566" is currently selected and highlighted.
- Course Information:** A section with five text boxes displaying details for the selected course:
 - Course:** Introduction to Fuzzy Logic
 - Schedule:** TH: 1:30 - 2:45 PM
 - Classroom:** TC-309
 - Credits:** 3
 - Enrollment:** 20

Figure 6.37 Data validation using the Course Form window.

using the stored procedures. Recall that we did not discuss this for the Microsoft Access database in Section 6.2 since the Microsoft Access database does not allow users to use stored procedures to access it using the Visual Studio.NET design tools and wizards such as the TableAdapter Query Configuration Wizard and Query Builder. However, for the SQL Server database, it does allow users to use stored procedures to access it to perform different data actions.

6.3.9 Insert Data into Database Using Stored Procedures

In this section, we want to discuss how to insert new records into the database using the stored procedures. To make it simple, we still use the Course and the Insert Course Form windows to show readers how to insert a new course record into the Course table in the database using the stored procedure. To do that, first we need to create a stored procedure named `InsertCourseSP` under the Course table using the TableAdapter Query Configuration Wizard, and then we need to make little modifications to the coding of the Insert button's Click method and the coding for the constructor of the Insert Course Form. The modifications include the following steps:

1. Open the constructor of the Insert Course Form window and add one more item "Stored Procedure" into the combobox `ComboMethod` to allow users to select this method to perform the data insertion.
2. In the Insert button's Click method, change the `else` clause to the `else if` clause for the `if` block to allow users to select the second method, `TableAdapter.Update()`, and add one more `else` clause for the third method, "Stored Procedure", to allow users to select this method.
3. In the Insert button's Click method, add the associated codes to call the built stored procedure to perform the data insertion.

Let's first create a stored procedure under the Course table using the TableAdapter Query Configuration Wizard.

6.3.9.1 Create Stored Procedure Using TableAdapter Query Configuration Wizard

Open the Data Source window by clicking the `Data>Show Data Sources` menu item, and then right-click on any location inside the Data Source window and select the `Edit the DataSet with Designer` item from the pop-up menu to open this window. Right-click on the last item from the Course table and select the `Add Query` item from the pop-up menu to open the TableAdapter Query Configuration Wizard. Check the `Create new stored procedure` radio button since we want to create a new stored procedure to do the data insertion, and click on the `Next` button to go to the next window. Check the `INSERT` radio button and click on `Next` to continue. Click on the `Query Builder` button on the opened window since we need to build a new query. The opened Query Builder dialog box is shown in Figure 6.38.

You may note that the order of the inserted columns in this stored procedure is different from the order of columns in the `InsertCourse()` query function we built in Figure 6.30. Of course, you can modify this order to make it identical with the order in the

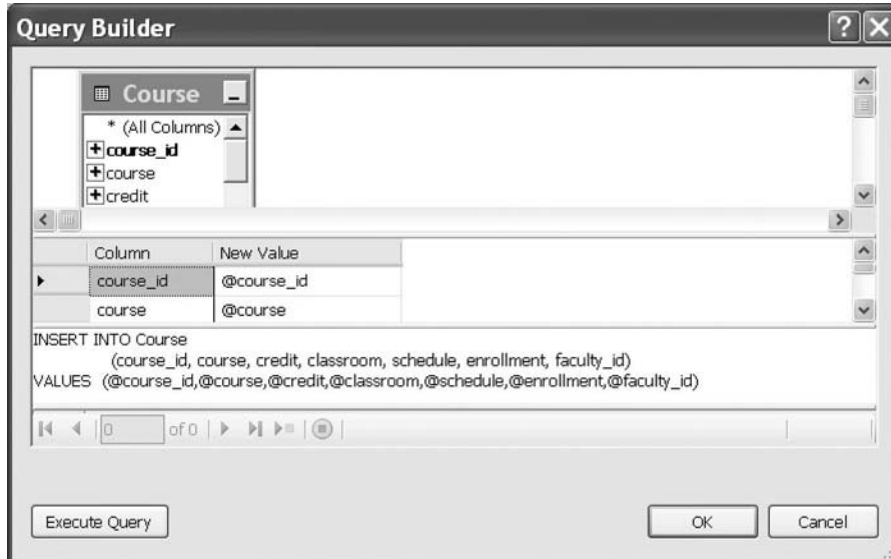


Figure 6.38 Opened Query Builder dialog box.

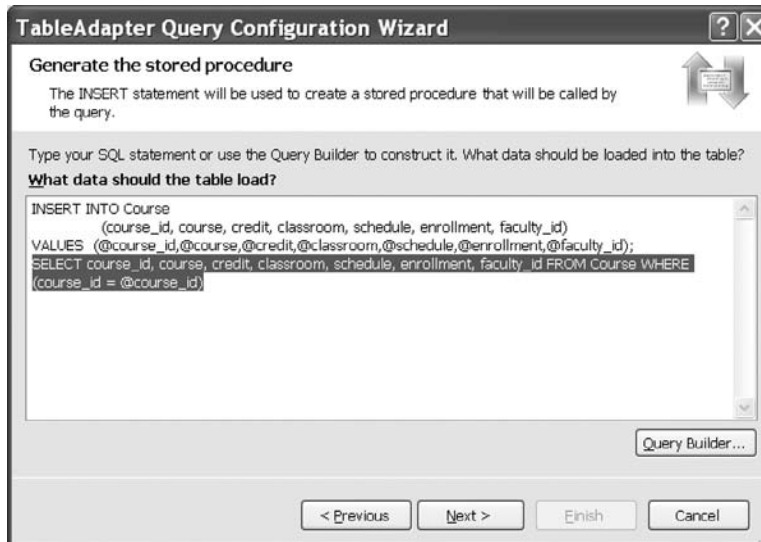


Figure 6.39 Confirmation dialog box.

InsertCourse() query function if you like. However, the important issue is that it doesn't matter if these orders are identical or not, the only key issue is that you must make sure that the order in this query function is identical with the order in the function that will be called from the Insert button's Click method later in your program. Click on the OK button to the next dialog box to confirm our built query function, which is shown in Figure 6.39.

```

SQLInsertWizard.InsertCourseForm | InsertCourseForm()
public InsertCourseForm()
{
    InitializeComponent();
    ComboName.Items.Add("Ying Bai");
    ComboName.Items.Add("Satish Bhalla");
    ComboName.Items.Add("Black Anderson");
    ComboName.Items.Add("Steve Johnson");
    ComboName.Items.Add("Jenney King");
    ComboName.Items.Add("Alice Brown");
    ComboName.Items.Add("Debby Angles");
    ComboName.Items.Add("Jeff Henry");
    ComboName.SelectedIndex = 0;
    ComboMethod.Items.Add("TableAdapter Insert");
    ComboMethod.Items.Add("TableAdapter Update");
    ComboMethod.Items.Add("Stored Procedure");
    ComboMethod.SelectedIndex = 0;
    txtCourseID.DataBindings.Clear();
    txtFacultyID.DataBindings.Clear();
}

```

Figure 6.40 Modification to the codes in the constructor.

Since we only need to insert a record into the database, highlight the second **SELECT** statement and delete it by pressing the Delete key on your keyboard. Click on the Next button again, and enter `InsertCourseSP` as the name of this query stored procedure into the name box. Click on the Next and the Finish buttons to close this process.

6.3.9.2 Modify Codes to Perform Data Insertion Using Stored Procedure

The first modification is to add one more item “Stored Procedure” into the combobox `ComboMethod` in the constructor of the Insert Course Form to allow users to select it to perform the data insertion. Open the constructor and add one more line of the code, which is shown in Figure 6.40, into this constructor. The codes we developed in the previous sections are indicated with shading.

Both the second and the third modifications are performed inside the Insert button’s Click method in the Insert Course Form window. Both modifications include adding an `else if` clause in the `if` block and adding related codes to perform the inserting new course records using the stored procedure method.

Open the Insert button’s Click method from the Insert Course Form window and add the codes shown in Figure 6.41 into this method. The new added codes, which are highlighted in bold, are indicated in steps **A** and **B**, respectively. In step **B**, the stored procedure we developed in the last section using the TableAdapter Query Configuration Wizard is called to insert a new course record into the Course table in our database. Note that the order and the data types of inserted items must be identical with those we defined in the stored procedure in the last section.

Now you can run the project to test inserting data using the stored procedure. It seems there is no difference between calling a query function and calling a stored procedure to perform this data insertion. Yes, that is true for this data action because the stored procedure is a function or a collection of functions that performs some special functional-

```

SQLInsertWizard.InsertCourseForm cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
    int check = 0, intInsert = 0;
    CSE_DEPTDataSet.CourseRow newCourseRow;
    InitCourseInfo();
    check = CheckCourseInfo();
    if (check == 0)
    {
        facultyTableAdapter.ClearBeforeFill = true;
        string strFacultyID = facultyTableAdapter.FindFacultyIDByName(ComboName.Text);
        txtFacultyID.Text = strFacultyID;
        if (ComboMethod.Text == "TableAdapter Insert")
        {
            intInsert = courseTableAdapter.InsertCourse(txtCourseID.Text, txtCourse.Text, txtClassRoom.Text,
                txtSchedule.Text, int.Parse(txtEnroll.Text), txtFacultyID.Text, double.Parse(txtCredits.Text));
        }
        A else if (ComboMethod.Text == "TableAdapter Update")
        {
            newCourseRow = cSE_DEPTDataSet.Course.NewCourseRow();
            newCourseRow = InsertCourseRow(ref newCourseRow);
            cSE_DEPTDataSet.Course.Rows.Add(newCourseRow);
            intInsert = courseTableAdapter.Update(cSE_DEPTDataSet.Course);
        }
        B else
            intInsert = courseTableAdapter.InsertCourseSP(txtCourseID.Text, txtCourse.Text,
double.Parse(txtCredits.Text), txtClassRoom.Text, txtSchedule.Text,
int.Parse(txtEnroll.Text), txtFacultyID.Text);
        if (intInsert != 0) // data insertion is successful
        {
            cmdCancel.PerformClick(); // clean up all faculty information
            cmdInsert.Enabled = false; // disable the Insert button
        }
        else
        {
            MessageBox.Show("The course insertion is failed");
            cmdInsert.Enabled = true;
        }
    }
    else
        MessageBox.Show("Fill all Course Information box, enter a NULL for blank column");
}

```

Figure 6.41 Code modifications for the Insert button's Click method.

ity or functionalities. However, we cannot create a stored procedure that can be used to perform multiple data actions to the multiple different data tables by using the TableAdapter Query Configuration Wizard since each TableAdapter can only access the associated data table. To insert data into the database using the runtime objects method, which we will discuss it in Part II, one stored procedure can access multiple different data tables and fulfill multiple different data manipulation operations.

At this point, we finished developing our sample project to insert data into the SQL Server database. A completed project SQLInsertWizard is located at the folder DBProjects\Chapter 6 located at the accompanying ftp site (See Chapter 1).

Next we will discuss how to insert data into the Oracle database using the Visual Studio.NET design tools and wizards.

6.4 INSERT DATA INTO ORACLE DATABASE USING SAMPLE PROJECT ORACLEINSERTWIZARD

Because of the similarity in data insertion between the SQL Server database and the Oracle database, all the codes we developed in the last section can be used to access the Oracle database to perform data insertion. The only difference between both databases is the connection string when the Oracle database is connected to the Visual C#.NET applications. To save space and time, we will not duplicate those codes in this section. Refer to Section 5.2.2.2 in Chapter 5 and Appendix E to get more detailed information on how to add and connect an Oracle database with your Visual C#.NET applications using the Design Tools and Wizards as well as the associated codes. Refer to Appendix F to get a clear picture on how to use the sample Oracle 10g XE database CSE_DEPT in C# projects. As long as this connection is set up, all coding jobs are identical with those we did for the SQL Server database in the last section, and you can directly use those codes to access the Oracle database to perform the different data actions. A complete data insertion project named OracleInsertWizard Project is located at the folder DBProjects\Chapter 6 located at the accompanying ftp site (See Chapter 1).

PART II DATA INSERTION WITH RUNTIME OBJECTS

Inserting data into the database using the runtime objects method is a flexible and professional way to perform the data insertion operation in the Visual C#.NET environment. Compared with those methods discussed in Part I in which Visual Studio.NET design tools and wizards were utilized to insert data into the database, the runtime objects method provides more sophisticated techniques to do this operation more efficiently and conveniently. Relatively speaking, the methods we discussed in the first part are easy to learn, but some limitations existed with those methods. First, each TableAdapter can only access the associated data table to perform data actions such as inserting data to that table only. Second, each query function built by using the TableAdapter Query Configuration Wizard can only perform a single query such as data insertion. Third, after the query function is built, no modifications can be made to that function dynamically, which means that the only time you can modify that query function is either before the project runs or after the project terminates. In other words, you cannot modify that query function as the project runs.

To overcome these shortcomings, we will discuss how to insert data using the runtime objects method in this part. Three sections are covered in this part: inserting data using the general runtime objects method is discussed. Inserting data into the database using the LINQ to DataSet and LINQ to SQL queries is introduced in the second Section. Inserting data using the stored procedures is presented in the third Section.

Generally, you need to use the TableAdapter to perform data actions in the database if you developed your applications using the Visual Studio.NET design tools and wizards in the design time. However, you should use the DataAdapter to make these data manipulations if you developed your projects using the runtime objects method.

6.5 GENERAL RUNTIME OBJECTS METHOD

We provided a very detailed introduction and discussion about the runtime objects method in Section 5.17 in Chapter 5. Refer to that section to get more detailed information about this method. For your convenience, we highlight some important points and general methodology of this method as well as some key points in using this method to perform the data actions again the databases.

As you know, ADO.NET provides different classes to help users develop professional data-driven applications by using different methods to perform specific data actions such as inserting data, updating data, and deleting data. For data insertion, two popular methods are widely applied:

1. Add new records into the desired data table in the DataSet, and then call the DataAdapter.Update() method to update the new added records from the table in the DataSet to the table in the database.
2. Build the INSERT command using the Command object, and then call the command's ExecuteNonQuery() method to insert new records into the database. Or you can assign the built command object to the InsertCommand property of the DataAdapter and call the ExecuteNonQuery() method from the InsertCommand property.

The first method is to use the so-called DataSet-DataAdapter method to build a data-driven application. DataSet and DataTable classes can have different roles when they are implemented in a real application. Multiple DataTables can be embedded into a DataSet, and each table can be filled, inserted, updated, and deleted by using the different properties of a DataAdapter such as the SelectCommand, InsertCommand, UpdateCommand, or DeleteCommand when the DataAdapter's Update() method is executed. The DataAdapter will perform the associated operations based on the modifications you made for each table in the DataSet. For example, if you add new rows into a table in the DataSet, then you call this DataAdapter's Update() method. This method will perform an InsertCommand based on your modifications. The DeleteCommand will be executed if you delete rows from the table in the DataSet and call this Update() method. This method is relatively simple since you do not need to call some specific methods such as the ExecuteNonQuery() to complete these data queries. But this simplicity brings some limitations for your applications. For instance, you cannot access different data tables individually to perform multiple specific data operations. This method is very similar to the second method we discussed in Part I, so we will not discuss it further here.

The second method allows you to use each object individually, which means that you do not have to use the DataAdapter to access the Command object, or use the DataTable with DataSet together. This provides more flexibility. In this method, no DataAdapter or DataSet is needed, and you only need to create a new Command object with a new Connection object. Then build a query statement and attach some useful parameter into that query for the new created Command object. You can insert data into any data table by calling the ExecuteNonQuery() method that belongs to the Command class. We will concentrate on this method in this part.

In this section, we provide three sample projects named SQLInsertRTOObject, AccInsertRTOObject, and OracleInsertRTOObject to illustrate how to insert new records into three different databases using the runtime objects method. Because of the coding similarity between these three databases, we will concentrate on inserting data to the SQL Server database using the SQLInsertRTOObject project first. Then we will illustrate

the coding differences between these databases by using the real codes for the two other sample projects.

Now let's first develop the sample project `SQLInsertRuntimeObject` to insert data into the SQL Server database using the runtime objects method. Recall that in sections 5.18.3 to 5.18.5 in Chapter 5, we discussed how to select data for the Faculty, Course, and Student Form windows using the runtime objects method. For the Faculty Form, a regular runtime selecting query is performed, and for the Course Form, a runtime joined-table selecting query is developed. For the Student table, the stored procedures are used to perform the runtime data query.

We will concentrate on inserting data to the Faculty table from the Faculty Form window using the runtime object method in this part. To avoid duplication of the coding, we will modify an existing project named `SQLSelectRuntimeObject` we developed in Chapter 5 to create our new project `SQLInsertRuntimeObject` used in this section.

6.6 INSERT DATA INTO SQL SERVER DATABASE USING RUNTIME OBJECTS METHOD

Open Windows Explorer and create a new folder Chapter 6 if you have not done that, and then browse to the folder `DBProjects\Chapter 5` located at the accompanying ftp site (See Chapter 1). Then copy the project `SQLSelectRuntimeObject` to our new folder `C:\Chapter 6`. Change the names of the solution and the project from `SQLSelectRuntimeObject` to `SQLInsertRuntimeObject`. Change the project file name from `SQLSelectRuntimeObject.csproj` to `SQLInsertRuntimeObject.csproj`. Double-click on the `SQLInsertRuntimeObject.csproj` to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to `Project|SQLInsertRuntimeObject Properties` menu item to open the project's property window. Change the `Assembly name` from `SQLSelectRuntimeObject` to `SQLInsertRuntimeObject` and the `Default namespace` from `SQLSelectRuntimeObject` to `SQLInsertRuntimeObject`, respectively.
- Click on the `Assembly Information` button to open the `Assembly Information` dialog box and change the `Title` and the `Product` to `SQLInsertRuntimeObject`. Click on `OK` to close this dialog box.

Go to the `File|Save All` to save those modifications. Now we are ready to develop our graphic user interfaces based on our new project `SQLInsertRuntimeObject`. To insert data into the Faculty data table, we need to add one more Windows Form as the user interface into this new project.

6.6.1 Add Inserting Data Form Window: Insert Faculty Form

The function of the `Insert Faculty Form` is as follows: As the project runs, after the user has successfully finished the login process and selected the Faculty Information from the Selection Form, the Faculty Form window will be displayed. When the user clicks on the Insert button, the `Insert Faculty Form` window will appear. This form allows users to insert data into the Faculty data table in the database using the runtime objects method. The form also allows users to enter all pieces of information into the appropriate text-boxes for the new inserted faculty. By clicking the Insert button, a new faculty record is

inserted into the database. However, if the user wants to reenter those pieces of information before finishing this insertion, the Cancel button can be used and all pieces of information entered will be erased. The Back button is used to allow users to return to the Faculty Form to perform the validation to confirm that the data insertion is successful.

Go to the Project/Add Windows Form menu item to open the Add New Item dialog box. Keep the default Templates, Windows Form, selected and enter the Insert Faculty Form.cs into the Name box as the name for this new form. Then click the Add button to add this form into our project.

To save time and space, we can copy all controls on this Faculty Form from the project InsertWizard Project we developed in this chapter. Open that project from the folder DBProjects\Chapter 6 located at the accompanying ftp site (See Chapter 1), and open the Insert Faculty Form window. Then go to the Edit/Select All menu item to select all controls on that form window. Go to the Edit/Copy menu item to copy those items. Now open our newly created project SQLInsertRTOBJECT and our new form Insert Faculty Form window, enlarge it to an appropriate size, and go to the Edit/Paste menu item to paste those controls into this form. Note that the project InsertWizard is developed using the Visual Studio.NET design tools and wizards, so some objects related to those design tools and wizards such as the Data BindingSource will be added into this form as you paste those controls. Because we don't need those objects in this runtime objects method, just delete all of them from our new Insert Faculty Form window. To do that, right-click on FacultyBindingSource from the bottom of this form window and select the Delete item from the pop-up menu to remove it.

In addition to removing the components related to the design tools and wizards, you also need to perform the following modifications to this form:

- Remove the combobox control ComboMethod from this form since we only use one method, the ExecuteNonQuery method of the Command class, to perform this runtime data insertion.
- Remove the Select button from this form since we will not perform the data validation until we click the Back button to return to the Faculty form window. In other words, the data validation is performed in the Faculty Form.
- Make sure the following properties of the form are set up:
 - Name: **InsertFacultyForm**
 - Text: **CSE DEPT Insert Faculty Form**
 - AcceptButton: **cmdInsert** (select the Insert button as the default button)
 - StartPosition: **CenterScreen** (locate the form in the center of the screen)

Your finished form window, Insert Faculty Form, is shown in Figure 6.42.

The detailed descriptions of the function of each control on this form can be found in Section 6.2.3 in this chapter. Simply speaking, the Faculty Photo checkbox is used to control both the Photo Name and the Photo Location textboxes to allow users to select the newly inserted faculty's photo. The Faculty ID textbox is a key textbox since the Insert button will be enabled if this textbox's content is changed, which means that a new faculty will be inserted. The Cancel button enables users to clean up the contents of all textboxes (except the Faculty ID) to reenter the faculty information. A new faculty record will be inserted into the database as the Insert button is clicked.

Our graphical user interface design is done. Next let's modify the codes for the copied project to make them match our new project.

Figure 6.42 Modified Insert Faculty Form window.

6.6.2 Modify Codes to Copied Project

The main modification is to change the namespace, `SQLSelectRTOObject`, which is used by the old project `SQLSelectRTOObject Project`, to the new one, `SQLInsertRTOObject`, which should be used for the new project for all code windows. Open the following code windows and perform this namespace modification:

1. LogIn Form.cs
2. Selection.cs
3. Faculty Form.cs
4. Course Form.cs
5. Program.cs
6. SP Form.cs
7. Student Form.cs

Open the following Form designer.cs files to perform the same modification:

1. LogIn Form designer.cs
2. Selection.designer.cs
3. Faculty Form.designer.cs
4. Course Form.designer.cs
5. SP Form.designer.cs
6. Student Form.designer.cs

The coding for this data insertion is divided into three steps: the data validation before the data insertion, data insertion using the runtime objects method, and the data validation after the data insertion. The purpose of the first step is to confirm that all inserted data stored in each associated textbox should be complete and valid. In other words, all textboxes should be nonempty. The third step is used to confirm that the data insertion is successful. In other words, the newly inserted data should be in the desired

table in the database and should be read back and displayed in the form window. Let's begin with the coding for the first step.

6.6.3 Startup Coding and Data Validation Before Data Insertion

First, let's take care of the startup coding. The so-called startup coding includes the coding for the constructor of the class `InsertFacultyForm` and field-level variables declarations, as well as coding for the `Cancel` and the `Back` buttons' `Click` methods. Open the code window of the `Insert Faculty Form` window and enter the codes shown in Figure 6.43 into this code window.

Let's take a close look at this piece of coding to see how it works.

- A. Some field-level variables are declared and created here. The field-level instance `logForm` is used to access the `Connection()` method in the `LogIn Form` window to check the database connection status before the data insertion can start. The string variable `FacultyInfo` is a string array that is mapped to all seven textboxes that hold the seven data columns of the `Faculty` data table, and it is used for the data validation before the data insertion.
- B. When the project runs and the `Insert Faculty Form` window is opened, the constructor of the `Insert Faculty Form` is executed. First, we need to check whether a valid connection between our Visual C#.NET project and the database exists. All nonreturned data queries such as `INSERT`, `UPDATE`, and `DELETE` are different with the data query `SELECT` that will return data. To perform a returned-data query such as the `SELECT`, one can use

```

public partial class InsertFacultyForm : Form
{
    A   LogInForm logForm = new LogInForm();
        string[] FacultyInfo = new string[7];

        public InsertFacultyForm()
        {
            B   InitializeComponent();
            C   logForm = logForm.getLogInForm();
                if (logForm.sqlConnection.State != ConnectionState.Open)
                    logForm.sqlConnection.Open();
        }

        private void cmdBack_Click(object sender, EventArgs e)
        {
            D   this.Hide();
        }

        private void cmdCancel_Click(object sender, EventArgs e)
        {
            E   txtName.Text = string.Empty;
                txtTitle.Text = string.Empty;
                txtOffice.Text = string.Empty;
                txtPhone.Text = string.Empty;
                txtCollege.Text = string.Empty;
                txtEmail.Text = string.Empty;
                txtPhotoName.Focus();
        }
}

```

Figure 6.43 Startup coding.

the `Fill()` method to populate a data table. The point is that when the `Fill()` is executed, first it checks whether a valid connection between the application and the database exists. If it does, this method executes the data query using that connection. However, if no valid connection exists, the `Fill()` method can first open a connection and then perform the data query. The key point is that all nonreturned data queries such as `INSERT`, `DELETE`, and `UPDATE` do not have this function to open a new database connection if no valid connection exist for those queries. We need to create a new connection if no valid connection is available for our data insertion to make sure that the data insertion job can be continued. The `getLogInForm()` method is executed to get the current `LogInForm`'s instance and assign it to the `logForm` instance we created in this form since we do not want to create any new instance of the `LogInForm` class and we want to use the existing one.

- C. Then we check the connection status of our database `Connection` object. The `Open()` method will be executed to open a new connection if no valid connection exists for this application.
- D. The coding for the `Back` button's `Click` method is easy. The method `this.Hide()` is executed to hide the `Insert Faculty Form` window. Hiding a window is different from closing a window; after the `Hide()` method is executed, the form window disappears from the screen, but it is still in the memory. However, after the `Close()` method is executed, the form window is removed from the memory and from the project. After the `Insert Faculty Form` window is hidden, the `Faculty Form` will be displayed to allow us to perform data validation.
- E. The coding for the `Cancel` button's `Click` method is also easy. The class method `string.Empty` is assigned to each textbox's `Text` property to clean up each of them. Note that this cleaning operation does not include the `Faculty ID` textbox. The reason for that is because after one new record is inserted into the database, the `Insert` button will be disabled to avoid multiple insertions of the same data. Each data is identified by the `Faculty ID`, which is a primary key in the `Faculty` table. Later on, in order to insert different new data that is indicated with a different `Faculty ID`, we need to enable the `Insert` button to allow users to do that. In other words, as long as the content of the `Faculty ID` textbox is changed, which means that a new different record needs to be inserted, the `Insert` button should be enabled. Cleaning the `Faculty ID` textbox is equivalent to changing its content. Therefore in order to avoid enabling the `Insert` button mistakenly, we will not clean up its content in this `Cancel` button's `Click` method.

Now let's do our coding for the data validation before the data insertion.

Data validation can be performed by calling two methods. The first method is named `InitFacultyInfo()`, and it is used to set up a mapping relationship between each item in the string array `FacultyInfo` and each textbox stored the faculty information. The second method is named `CheckFacultyInfo()`, and it is used to scan and check all textboxes to make sure that none them is empty.



Unlike the returned-data query, such as `SELECT`, in which a system query method `Fill()` is executed to try to open a new database connection if no valid connection exists between the database and the application, all nonreturned data queries such as `INSERT`, `UPDATE`, and `DELETE` do not have this function to open a new connection between the database and the application if no valid connection exists. Therefore the user must develop codes to provide this kind of functionality to those nonreturned data queries.


```

private void InitFacultyInfo()
{
    FacultyInfo[0] = txtID.Text;
    FacultyInfo[1] = txtName.Text;
    FacultyInfo[2] = txtTitle.Text;
    FacultyInfo[3] = txtOffice.Text;
    FacultyInfo[4] = txtPhone.Text;
    FacultyInfo[5] = txtCollege.Text;
    FacultyInfo[6] = txtEmail.Text;
}

private int CheckFacultyInfo()
{
    int pos = 0, check = 0;
    for (pos = 0; pos <= 6; pos++)
    {
        if (FacultyInfo[pos] == string.Empty)
            check++;
    }
    return check;
}

```

Figure 6.44 Coding for the InitFacultyInfo subroutine.

Open the code window of the Insert Faculty Form, and enter the codes to create two user-defined methods, InitFacultyInfo() and CheckFacultyInfo(), which are shown in Figure 6.44.

Let's take a look at this piece of code to see how it works.

- A.** The FacultyInfo is a zero-based string array that starts its index from 0. All seven textboxes related to faculty information are assigned to this array. In this way, it is easier for us to scan and check each of textbox to make sure that none is empty later.
- B.** A for loop is used to scan each textbox in the FacultyInfo string array to check whether any of them is empty. A message will be displayed if this situation happened and the checking status is returned to allow user to fill all textboxes.

Now let's develop the code for the Insert button's Click method to call those two methods to perform data validation. Open the Insert button's Click method by double-clicking on the Insert button from the form window of the Insert Faculty Form, and enter the codes shown in Figure 6.45 into this method.

The function of this code is straightforward and easy to understand. First, the user-defined method InitFacultyInfo() is called to set up the mapping relationship between each item in the string array FacultyInfo and each textbox. Then another user-defined method CheckFacultyInfo() is executed to check and make sure that no textbox is empty. If any of textboxes are empty, this method returns a nonzero value to allow users to refill any missed information to the associated textboxes until all of them are nonempty.

Otherwise, if this method returned a zero, which means that all textboxes are filled with certain faculty-related information, we need to check whether the Faculty Photo checkbox has been checked. The name of a valid faculty photo file and a valid location for that photo should have been filled into the Photo Name and the Photo Location


```

SQLInsertRTOject.InsertFacultyForm cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
    int check = 0;
    InitFacultyInfo();
    check = CheckFacultyInfo();
    if (check == 0) // all textboxes have been filled.
    {
        if (chkPhoto.Checked == true && (txtPhotoName.Text == "" || txtPhotoLocation.Text == ""))
            MessageBox.Show("Photo Name/Photo Location is empty");
        // continue to perform the data insertion.....
    }
    else
        MessageBox.Show("Fill all Faculty Information box, enter a NULL for blank column");
}

```

Figure 6.45 Initial coding for the Insert button's Click method.

```

SQLInsertRTOject.InsertFacultyForm chkPhoto_CheckedChanged()
private void chkPhoto_CheckedChanged(object sender, EventArgs e)
{
    if (chkPhoto.Checked == true)
    {
        txtPhotoName.Enabled = true;
        txtPhotoLocation.Enabled = true;
        txtPhotoName.Focus();
    }
    else
    {
        txtPhotoName.Enabled = false;
        txtPhotoLocation.Enabled = false;
    }
}

```

Figure 6.46 Coding for the chkPhoto_CheckedChanged() method.

textboxes by the user if this checkbox is checked. If not, a MessageBox with a warning note is displayed to remind users to fill those textboxes accordingly.

Next, let's write the coding to initialize the Faculty Photo (chkPhoto) checkbox. When the chkPhoto checkbox is clicked on by the user, two possibilities exist: The chkPhoto is checked or unchecked. If it is checked, both the Photo Name (txtPhotoName) and the Photo Location (txtPhotoLocation) textboxes should be enabled, and a focus should be set in the Photo Name textbox to enable the user to directly enter the photo name into that textbox without clicking on it first. Otherwise both textboxes should be disabled if the chkPhoto is not checked.

To do this coding, double-click on the Faculty Photo checkbox from the Insert Faculty Form window to open its CheckedChanged method and enter the codes shown in Figure 6.46 into this method.

At this point, we have completed the coding for the data validation before the data insertion and the startup coding. Now let's write the coding for the data insertion.

6.6.4 Insert Data into Faculty Table

Since we will develop codes to perform data insertion to the SQL Server database, we need to use some classes, components, and methods related to the SQL Server database, which are defined in the **System.Data.SqlClient** namespace in the ADO.NET. The first coding we need to write is to declare this namespace in our code window to allow those classes, components, and methods to be recognized. Open the code window of the Insert Faculty Form window and enter this declaration, which has been highlighted in bold at the top of this window, which is shown in Figure 6.47.

The main coding job is performed inside the Insert button's Click method in the Insert Faculty Form window. We already developed some codes in the last section. Now let's continue to complete this coding.

Open the method and enter the codes shown in Figure 6.48 into the Insert button's Click method. The codes developed in the previous section are shaded.

Let's take a look at this piece of code to see how it works.

- A. A local integer variable `intInsert` is created first, and this variable is used to hold the returned status of the execution of the data insertion query later. Then the SQL INSERT statement is declared, and this insertion query contains seven parameters following the statement VALUES. Each parameter is prefixed by an @ symbol since this is the requirement for the SQL Server database.
- B. The data components used to perform the data insertion are declared here, which include the `SqlDataAdapter` and `SqlCommand`.
- C. Since the database Connection object is created in the LogIn Form, in order to use that object, first we need to get the instance of the `LogInForm` class by calling the `getLogInForm()` method, which is defined in the `LogInForm` class.
- D. The Command object is initialized with the Connection, CommandType, and CommandText properties of the Command class.
- E. A user-defined method `InsertParameters()` is called to fill parameters to the Parameters collection of the Command object. Figure 6.49 shows the detailed coding for this method.

The screenshot shows a code window titled "SQLInsertRTOject.InsertFacultyForm" with a sub-window titled "InsertFacultyForm()". The code inside is as follows:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace SQLInsertRTOject
{
    public partial class InsertFacultyForm : Form
    {
        .....
    }
}

```

Figure 6.47 Coding of declaration of the namespace.

```

SQLInsertRTOject.InsertFacultyForm cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
  A int check = 0, intInsert = 0;
  B string cmdString = "INSERT INTO Faculty (faculty_id, faculty_name, office, phone, college, title, email) " +
    "VALUES (@faculty_id,@faculty_name,@office,@phone,@college,@title,@email)";
  C SqlDataAdapter FacultyDataAdapter = new SqlDataAdapter();
  D SqlCommand sqlCommand = new SqlCommand();
  E InitFacultyInfo();
  F check = CheckFacultyInfo();
  G if (check == 0) // all textboxes have been filled.
  {
    H logForm = logForm.getLogInForm();
    if (chkPhoto.Checked == true && (txtPhotoName.Text == "" || txtPhotoLocation.Text == ""))
      MessageBox.Show("Photo Name/Photo Location is empty");
    sqlCommand.Connection = logForm.sqlConnection;
    sqlCommand.CommandType = CommandType.Text;
    sqlCommand.CommandText = cmdString;
    InsertParameters(sqlCommand);
    //FacultyDataAdapter.InsertCommand = sqlCommand
    //intInsert = FacultyDataAdapter.InsertCommand.ExecuteNonQuery()
    intInsert = sqlCommand.ExecuteNonQuery();
    if (intInsert == 0)
      MessageBox.Show("The data insertion is failed");
    else
    {
      cmdCancel.PerformClick(); // clean up all faculty information
      cmdInsert.Enabled = false;
    }
  }
  else
    MessageBox.Show("Fill all Faculty Information box, enter a NULL for blank column");
}

```

Figure 6.48 Coding for the Insert button's Click method.

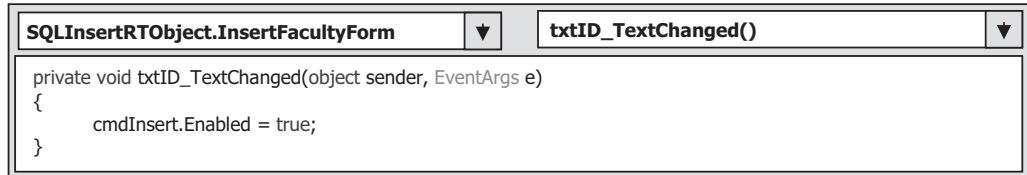
```

SQLInsertRTOject.InsertFacultyForm InsertParameters()
private void InsertParameters(SqlCommand cmd)
{
  cmd.Parameters.Add("@faculty_id", SqlDbType.Char).Value = txtID.Text;
  cmd.Parameters.Add("@faculty_name", SqlDbType.Char).Value = txtName.Text;
  cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text;
  cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text;
  cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text;
  cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text;
  cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text;
}

```

Figure 6.49 Coding for the user-defined InsertParameters() method.

- F.** After the Command object is initialized, the ExecuteNonQuery() method of the Command class is called to insert the new record into the Faculty table in the database. The commented out codes with the light color provide another data insertion method, and they can perform the same insertion function as this one.
- G.** The ExecuteNonQuery() method returns an integer as feedback to indicate whether this insertion is successful or not. The value of this returned integer is exactly equal to the



```

SQLInsertRTOject.InsertFacultyForm  txtID_TextChanged()
private void txtID_TextChanged(object sender, EventArgs e)
{
    cmdInsert.Enabled = true;
}

```

Figure 6.50 Coding for the TextChanged method.

number of new inserted records into the Faculty data table. A returned zero means that no new record have been inserted into the Faculty table and this insertion has failed. A warning message would be displayed to indicate this situation to the user if this occurs.

- H.** If the returned value of the variable `intInsert` is nonzero, this data insertion is successful. A cleaning action is taken by executing a system method `PerformClick()` for the Cancel button, which is equivalent to clicking on the Cancel button, to clean up the contents of all textboxes that contain the newly inserted faculty information, except the Faculty ID. Also the Insert button is disabled after this data insertion to avoid multiple insertions of the same data. This button will be enabled again when the content of the Faculty ID textbox is changed, which means that a new different record is ready to be inserted into the Faculty table.

The detailed coding for the user-defined method `InsertParameters()` is shown in Figure 6.49.

This piece of coding is easy; each piece of faculty-related information stored in the associated textbox is assigned to each matched parameter by using the `Add()` method. Note that the `@` symbol must be prefixed before each parameter since this is the requirement of the SQL Server database operations.

Another coding is for the Faculty ID textbox, actually for the `TextChanged` event and its method of the Faculty ID textbox. As we mentioned, in order to avoid multiple insertions of the same data, the Insert button will be disabled after one data insertion is completed. This Insert button will be enabled again when the content of the Faculty ID textbox is changed, which means when a different new record is ready to be inserted into the database. Double-click on the Faculty ID textbox from the Insert Faculty Form window to open this method and enter the codes shown in Figure 6.50 into this method.

Now let's first run the project to test the coding we developed to try to insert a new record into the Faculty data table in the database. However, before we can run the project, we need to add some codes into the Faculty Form to trigger the Insert Faculty Form window as the user clicks on the Insert button from the Faculty Form window.

Two pieces of codes need to be added into the Faculty Form window:

1. Display the Insert Faculty Form window as the user clicks on the Insert button from the Faculty Form window.
2. Close the Insert Faculty Form window and hide the Faculty Form window as the user clicks on the Back button from the Faculty Form window.

To do the coding for operations 1 and 2, we need first to create a new field-level instance of the `InsertFacultyForm` class, `InsertFaculty`, which is shown in step **A** in Figure 6.51. Then, open the Faculty Form window and double-click on the Insert button to open its `Click` method. Enter the codes shown in step **B** in Figure 6.51 into this method to

```

SQLInsertRTOject.FacultyForm  FacultyForm()
namespace SQLInsertRTOject
{
    public partial class FacultyForm : Form
    {
        private Label[] FacultyLabel = new Label[7];
        InsertFacultyForm InsertFaculty = new InsertFacultyForm();
        .....
        private void cmdInsert_Click(object sender, EventArgs e)
        {
            InsertFaculty.Show();
        }
        private void cmdBack_Click(object sender, EventArgs e)
        {
            InsertFaculty.Close();
            this.Hide();
        }
    }
}

```

Figure 6.51 Added codes to the Faculty Form window.

Figure 6.52 Running status of the Insert Faculty Form window.

display the Insert Faculty Form window. Finally double-click on the Back button on the Faculty Form window to open its Click method and add the code shown in step C in Figure 6.51 into that method. The codes we developed in the previous section have been indicated with shading.

Now we are ready to run our project to test the data insertion function. Start the project by clicking on the Start Debugging button, and enter the correct username and password to the LogIn form, and then select the Faculty Information from the Selection Form window to open the Faculty Form. Click on the Insert button from the Faculty Form to open the Insert Faculty Form window, which is shown in Figure 6.52.

Enter the following information to the associated textbox as the new information for a new faculty:

- P99875 Faculty ID textbox
- Peter Jones Faculty Name textbox
- Professor Title textbox
- MTC-225 Office textbox
- 750-330-5587 Phone textbox
- University of Chicago College textbox
- pjones@college.edu Email textbox

Keep the Faculty Photo checkbox unchecked, and your finished information should match Figure 6.52.

Click on the Insert button to insert this new record into the Faculty data table in the database. Immediately all textboxes, except the Faculty ID, become empty and the Insert button is disabled. Click on the Back button to return to the Faculty Form window, and then click on the Back and Exit buttons for the Faculty and the Selection forms to terminate the project. Our data insertion is successful!

Wait a moment. How do you know this data insertion is successful? We should find a way to confirm this. Well, an easy way is that the new inserted data should be able to be read back from the database and can be displayed if this insertion is successful. We have a very good candidate form to perform this data validation, the Faculty Form, because we already developed the codes for this form to perform the data query. Now let's add some codes to this Faculty Form to perform our data validation job for this new inserted data.

6.6.5 Validate Data After Data Insertion

To validate the new inserted faculty record, the Faculty Form window is used. The function of this validation is to read back the new inserted faculty record from the database and display it on the Faculty Form to confirm that the data insertion is successful. We need to use the codes we developed for the Select button's Click method in Chapter 5 to perform this data query.

To use the Select button and its Click method in the Faculty Form window to confirm our data insertion, we need to perform the following modifications to both the Insert Faculty Form and the Faculty Form code windows. First, let's do the coding modifications for the Faculty Form code window:

1. Modify the codes inside the constructor of the Faculty Form to retrieve the current faculty members from the Faculty table in the database. This can be realized by adding a new user-defined method UpdateFaculty().
2. Modify the codes inside the Insert button's Click method by adding an instruction to set up an instance of the FacultyForm class in the InsertFacultyForm window. The reason for that is because we need to use some object and method located in the FacultyForm window from the InsertFacultyForm, such as the ComboName and the UpdateFaculty(), to update the faculty members stored in the ComboName box in the FacultyForm window. By setting up this instance, we can easily call and use the FacultyForm from the InsertFacultyForm object.

3. Modify the codes inside the FindName() method to check and display the photo of the new inserted faculty. A default photo will be displayed if no photo is used with that faculty insertion.

Now let's begin to do these modifications for the FacultyForm window.

6.6.5.1 Modifications to Faculty Form Window

Open the constructor in the Faculty Form window and perform the modifications shown in Figure 6.53.

Let's take a look at this piece of code to see how it works.

- A. Remove eight Add() methods from the ComboName object, which have been highlighted in gray. Recall that we always load these eight faculty members and add them into the ComboName box using this Add() method in the previous projects we built in the last chapter. The reason for that is because in those projects, we only need to perform the data retrieving operations without changing the data in our database, and we assume that the faculty members stored in the Faculty table are identical with those we added into the ComboName box. However, the situation is changed in this project since we need to insert some new faculty members into our database. Therefore we need to pick up and display the updated or the current faculty members each time when the FacultyForm window is opened.
- B. Add a new user-defined method, UpdateFaculty(), to this constructor to pick up and display the updated faculty members in the ComboName box. The new added method has been highlighted in bold. The detailed codes for this method are shown in Figure 6.54. Note that the accessing mode for this method is Public since we need to call this method from the Insert Faculty Form window later. Therefore, we need to allow this method to be accessed globally.

The rest of the codes in this method are unchanged. The detailed codes for the UpdateFaculty() method are shown in Figure 6.54.

```

SQLInsertRTOject.FacultyForm | FacultyForm()
public FacultyForm()
{
    InitializeComponent();
    //ComboName.Items.Add("Ying Bai");
    //ComboName.Items.Add("Satish Bhalla");
    //ComboName.Items.Add("Black Anderson");
    //ComboName.Items.Add("Steve Johnson");
    //ComboName.Items.Add("Jenney King");
    //ComboName.Items.Add("Alice Brown");
    //ComboName.Items.Add("Debby Angles");
    //ComboName.Items.Add("Jeff Henry");
    UpdateFaculty(); // added in nov 15 2008
    ComboName.SelectedIndex = 0;
    ComboMethod.Items.Add("DataAdapter Method");
    ComboMethod.Items.Add("DataReader Method");
    this.ComboMethod.SelectedIndex = 0;
}

```

Figure 6.53 Modifications to the constructor of the FacultyForm class.

```

SQLInsertRTObject.FacultyForm UpdateFaculty()
public void UpdateFaculty()
{
A   LogInForm logForm = new LogInForm();
B   logForm = logForm.getLogInForm();
C   SqlCommand sqlCommand = new SqlCommand("SELECT faculty_name FROM Faculty", logForm.sqlConnection);
D   SqlDataReader sqlReader = sqlCommand.ExecuteReader();
   while (sqlReader.Read())
   {
       ComboName.Items.Add(sqlReader[0].ToString());
   }
}

```

Figure 6.54 Detailed codes for the UpdateFaculty() method.

```

SQLInsertRTObject.FacultyForm cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
   InsertFaculty.setFacultyForm(this);
   InsertFaculty.Show();
}

```

Figure 6.55 Modifications to the Insert button's Click method.

Let's take a look at this piece of code to see how it works.

- A.** A new instance of the LogIn Form class, logForm, is created since we need to use the Connection object created in that LogIn Form object.
- B.** The method getLogInForm() that is defined inside the LogIn Form class is called to get the current instance of that class and assigned to the new instance we just created because we do not want to use any new instance for the LogIn Form class.
- C.** A new instance of the Command class is created with two arguments: query string and Connection object. The ExecuteReader() method of the Command class is called to perform a read-only operation to retrieve the faculty names for all faculty members in the Faculty table, and assign them to the DataReader.
- D.** A while loop is executed to serially read out all returned faculty names and add them to the ComboName box.

In this way, we can load and update the contents of the ComboName box by retrieving all current faculty names from the Faculty table.

Now let's modify the codes inside the Insert button's Click method by adding an instruction to set up an instance of the FacultyForm class in the InsertFacultyForm window. Figure 6.55 shows this added instruction that is highlighted with the bold word.

As we mentioned, we need to update the faculty members stored in the ComboName box in the Faculty Form window from the Insert Faculty Form window after a new faculty has been inserted to our database. Therefore, we need to access the Faculty Form window to perform this updating from the Insert Faculty Form window. In order to do that, we need to set up an instance of the Faculty Form class to the Insert Faculty Form window before we can call and run the Insert Faculty Form window by using a new method setFacultyForm(), which will be developed later in the Insert Faculty Form window. Because

we do not want to create a new instance of the Faculty Form class, instead, we want to use the current instance of the Faculty Form class. Therefore, the current instance name **this** is passed as the argument for that method.

The final modification to the Faculty Form is the FindName() method. This method is used to select and display a valid faculty image based on the selected faculty name.

6.6.5.2 Insert New Faculty Photo

The function of the FindName() method is to identify and display the photo for both existing and new inserted faculty based on the selected faculty member in the Faculty Name textbox. Three possible situations exist for this faculty photo when a new faculty is inserted from the Insert Faculty Form window:

1. The Faculty Photo checkbox, chkPhoto, in the Insert Faculty Form window is not checked, which means that the user inserted a new faculty without any photo accompanying with that insertion.
2. The Faculty Photo checkbox is checked, and the Photo Location is the Default Location.
3. The Faculty Photo checkbox is checked, and the Photo Location textbox contains a valid photo location.

For the first situation, a MessageBox will be displayed to indicate that no faculty photo is found for the selected faculty when the Select button is clicked on and the validation process is executed. In the second case, the faculty photo file is located at the folder in which the executable file of our current project, SQLInsertRTObject.exe, is stored. In this application, it is C:\Chapter 6\SQLInsertRTObject\SQLInsertRTObject\bin\Debug. The third situation indicates that the faculty photo is located at a specified location.

Based on the analysis above, we only need to modify the codes in the selection branch part of this method, which means that no matched faculty photo file can be found for the selected faculty. This makes sense since the photo file of the new inserted faculty has not been added into the Switch ... Case block.

Open this method and browse to the selection branch part, perform the following modifications to this part, which are shown in Figure 6.56.

Let's take a look at this piece of modified code to see how it works.

- A. An `else` branch is added under the `if` block to handle the situation in which no matched faculty photo can be found.
- B. The stretch mode, `StretchImage`, is assigned to the `SizeMode` property of the `PhotoBox` to adjust the size of the faculty photo and make it match to the size of the `PhotoBox`.
- C. If the second situation discussed above occurs, the faculty photo file is located at the default location, and the name of the faculty photo file, which is stored in the `txtPhotoName` textbox on the Insert Faculty Form window, is assigned to the string variable `strName`, and the `FromFile()` method is executed to display this photo.
- D. If the third case discussed above happened, the faculty photo file is located at a specified location that is stored in the `txtPhotoLocation` textbox on the Insert Faculty Form window. The concatenating operator `+` is used to combine that location with the name of the faculty photo file and assign this combination result to the string variable `strName`. Similarly, the `FromFile()` method is executed to display this photo.

```

SQLInsertRTOject.FacultyForm FindName()
.....
if (strName != "No Match")
{
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage;
    PhotoBox.Image = System.Drawing.Image.FromFile(strName);
}
A else // added in oct 23, 2008
{
B   PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage;
C   if ((InsertFaculty.chkPhoto.Checked == true)&&(InsertFaculty.txtPhotoLocation.Text=="Default Location"))
    {
        strName = InsertFaculty.txtPhotoName.Text;
        PhotoBox.Image = System.Drawing.Image.FromFile(strName);
    }
D   if ((InsertFaculty.chkPhoto.Checked == true) && (InsertFaculty.txtPhotoLocation.Text != "Default Location"))
    {
        strName = InsertFaculty.txtPhotoLocation.Text + "\\\" + InsertFaculty.txtPhotoName.Text;
        PhotoBox.Image = System.Drawing.Image.FromFile(strName);
    }
}
return strName;

```

Figure 6.56 Modifications to the FindName method.

Now we have completed the modifications to the Faculty Form. Next let's perform the modifications to the coding of the Insert Faculty Form window.

6.6.5.3 Modifications to Insert Faculty Form Window

The modifications to this part contain the following steps.

1. Declare a field-level nominal instance of the Faculty Form class, Faculty_Form. The so-called nominal instance is that we only declare this instance without assigning any memory space to it. This is different from creating a new instance with the keyword new. We need to assign the passed instance we did in Figure 6.55 to this Faculty_Form later. Note that the accessing mode of this instance should be Public since we need to access and use it via the Insert Faculty Form object when the project is running.
2. Create a new method setFacultyForm() as we discussed in Figure 6.55 to reserve the current instance of the Faculty Form class, and assign it to the nominal instance Faculty_Form we declared in step 1.
3. Modify the codes inside the Back button's Click method to call that UpdateFaculty() method shown in Figure 6.54 to update the faculty members stored in the ComboName box on the Faculty Form window. This means that when one clicks on the Back button, the data insertion is complete and the next job, data insertion validation, starts.

Now open the code window of the Insert Faculty Form to perform those modifications listed above. The finished code is shown in Figure 6.57.

All of the modified codes have been highlighted in bold. Let's have a close look at this piece of modified code to see how it works.

- A. A field-level nominal instance of the Faculty Form class, Faculty_Form, is declared (step 1 above).
- B. A new method setFacultyForm() is created with the accessing mode as Public.

```

public partial class InsertFacultyForm : Form
{
    LogInForm logForm = new LogInForm();
    string[] FacultyInfo = new string[7];
    public FacultyForm Faculty_Form; // added in nov 16, 2008
    .....

    public void setFacultyForm(FacultyForm facultyForm) // added in nov 16, 2008
    {
        Faculty_Form = facultyForm;
    }
    .....

    private void cmdBack_Click(object sender, EventArgs e)
    {
        Faculty_Form.ComboName.Items.Clear(); // added in nov 16, 2008
        Faculty_Form.UpdateFaculty(); // added in nov 16, 2008
        Faculty_Form.ComboName.SelectedIndex = 0; // added in nov 16, 2008
        this.Hide();
    }
}

```

Figure 6.57 Modifications to the Insert Faculty Form code window.

- C. The passed current instance of the Faculty Form class, facultyForm, is assigned to that nominal instance Faculty_Form. Now the Faculty_Form is the current instance of the Faculty Form class.
- D. In the Back button's Click method, the Clear() method is called to clean up all faculty members from the ComboName box in the Faculty Form window. This step is necessary because we will load and add the updated faculty members to this box, and some duplicated faculty members would be displayed in this box without executing this cleaning job first.
- E. The UpdateFaculty() method in the Faculty Form window is called to load the updated faculty members from the database and add them into the ComboName box to update it.
- F. To set up the SelectedIndex property to 0, the first faculty member stored in the ComboName box can be selected and displayed.

At this point, we have completed all coding modifications for the data insertion and the data validation tasks. However, before we can run the project to test those tasks, we must store the faculty photo file into the desired location, either into the default folder or the user-selected folder if you want to display a faculty photo with that data insertion. Furthermore, this location must be identical to the location to be entered into the Photo Location textbox as the project runs.

Now let's run the project to test the functionalities of the coding we did above. Since we want to add a faculty photo for this data insertion, make sure that your desired faculty photo file has been already saved into the desired location. For this test, we want to display a faculty photo, and its file is named Mhamed . jpg, and we have stored this file in our default folder, C:\Chapter 6\SQLInsertRTOject\SQLInsertRTOject\bin\Debug.

Now start the project. After the project begins to run, enter the suitable username and password, such as jhenry and test, to the LogIn form, and then select the Faculty Information item from the Selection Form to open the Faculty Form window. Click the Insert button to open the Insert Faculty Form window, and enter the following information into this form as a piece of new faculty information:

Figure 6.58 Running status of the Insert Faculty Form window.

- A99875 Faculty ID textbox
- Ali Mhamed Faculty Name textbox
- Associate Professor Title textbox
- MTC-235 Office textbox
- 750-330-3387 Phone textbox
- University of Main College textbox
- amhamed@college.edu Email textbox

Then check the Faculty Photo checkbox and enter Mhamed.jpg into the Photo Name textbox as the photo file name. Keep the Default Location unchanged in the Photo Location textbox. Your finished information for this new faculty is shown in Figure 6.58.

Now click on the Insert button to insert this new faculty record into the database. Immediately the Insert button is disabled after this insertion. Click on the Back button to return to the Faculty Form to perform the validation for this new data insertion.

On the opened Faculty Form window, click on the drop-down arrow of the Faculty Name combobox, and you can find that not only our new inserted faculty member, Ali Mhamed, but also another new faculty member, Peter Jones, which we inserted in Section 6.6.4 (refer to Figure 6.52), has been added into this Faculty Name box. Click on Ali Mhamed to select this faculty, and then click on the Select button to try to read back new inserted information for that faculty from the database and display it in this Faculty Form window.

Immediately you can find that all information about that new inserted faculty record, including the faculty photo, is displayed in the associated labels, which is shown in Figure 6.59. Our data insertion is successful because the new inserted data have been retrieved from the database and displayed in this Faculty Form successfully.

One important issue for this data validation is that each time you insert a new piece of faculty information into the database using this Insert Faculty Form window, the faculty name must not be identical, which may be a potential bug for regular database operations. Some readers may argue with me: The individual faculty member is identified by the faculty ID, not by faculty name, and the faculty ID is the primary key in the Faculty

Figure 6.59 Example of the data validation result.

table. Of course, for regular database operations, one can insert multiple faculty members who have the identical faculty name into the database as long as their faculty IDs are different. Yes, that is true. However, the point is that in this application, we only use the faculty name, not the faculty ID, as the criterion to perform this SELECT query. This means that the query criterion is based on the faculty name, not faculty ID. Multiple records would be returned if the query contains multiple faculty members who have the same name, even they have the different faculty IDs.

Click on the Back and Exit buttons from the Faculty Form and the Selection Form windows to close our project. It is highly recommended to delete those new inserted faculty records from our sample database CSE_DEPT since we want to keep our database neat and clean.

A completed Visual C#.NET project SQLInsertRTOject can be found at the folder DBProjects\Chapter 6 located at the accompanying ftp site (see Chapter 1).

Basically, there is no significant difference between inserting data into the SQL Server, Microsoft Access, or the Oracle databases. The only differences are the configurations of the query strings such as the Connection string and the SELECT query string used in the LogIn Form, the SELECT query string used in the Faculty Form, and the INSERT query string and Parameter strings used in the Insert Faculty Form. All other coding is identical. We will show these differences and discuss how to insert data into the Microsoft Access 2007 database in Section 6.7 and into the Oracle database in Section 6.8.



One possible problem when you test your project by inserting more data into the Faculty table is that too many records are added into the database. To remove those unused records, you can open the Faculty table from the SQL Server Management Studio Express and then delete those records from the Faculty table

6.7 INSERT DATA INTO MICROSOFT ACCESS DATABASE USING RUNTIME OBJECTS

There is no big difference for data insertion between the different databases, and just as we mentioned at the end of the last section, the only differences are query strings used in the different form windows. All other parts of coding are identical without modifications. So we can use most codes in the project `SQLInsertRTOObject` we developed in the last section with small modifications for those query strings to make it work for the Microsoft Access 2007 database.

First, let's modify the project `SQLInsertRTOObject`. Open Windows Explorer to create a new folder such as `Chapter 6` if you have not done that, and then copy the project `SQLInsertRTOObject` from the folder `DBProjects\Chapter 6` located at the accompanying ftp site (see Chapter 1) to our new folder `Chapter 6`. Change the name of this project to `AccessInsertRTOObject`, which includes the following files:

- `AccessInsertRTOObject.sln`
- `AccessInsertRTOObject.csproj`
- `AccessInsertRTOObject.exe`
- `AccessInsertRTOObject.pdb`
- `AccessInsertRTOObject.cshost.exe`
- `AccessInsertRTOObject.xml`

To rename the last four files, you need to use the Project Properties window. Open the Project Properties window, and then change the names for the Assembly name and the Default namespace to `AccessInsertRTOObject`. Then click on the Assembly Information button to open the Assembly Information dialog box, and change the Title and the Product to `AccessInsertRTOObject`. Click on the OK button to close that dialog box.

6.7.1 Modifications to Namespaces

The first modification we need to perform is to change the namespace for the following files:

1. Window Form Object files
2. Window Form Designer Object files
3. Project Main Entry file

Open the following Window Form object files and change the namespace from `SQLInsertRTOObject` to `AccessInsertRTOObject`:

- `LogIn Form.cs`
- `Selection.cs`
- `Faculty Form.cs`
- `Course Form.cs`
- `Insert Faculty Form.cs`
- `SP Form.cs`
- `Student Form.cs`

Open all Window Form Designer files related to Window Form Object files listed above and change the namespace from `SQLInsertRuntimeObject` to `AccessInsertRuntimeObject`.

Open the Project Main Entry file `Program.cs` and change the namespace from `SQLInsertRuntimeObject` to `AccessInsertRuntimeObject`.

In this section, we will use the Faculty Form as an example to illustrate how to insert a new faculty record into the Faculty table in the Microsoft Access 2007 database. We can modify the project `SQLInsertRuntimeObject` to get a new project `AccessInsertRuntimeObject` to perform our data insertion operation using the runtime objects method. Basically, we need to perform the following modifications:

1. Add `System.Data.OleDb` namespace to all Form windows if it has not been added since we need to use Data Components related to OleDb Data Providers to perform any data query to the Microsoft Access 2007 database.
2. Database Connection string—make it connect to the Microsoft Access database.
3. LogIn username and password query strings—complete the login process.
4. Faculty table query string—select the correct faculty information.
5. Modifications to other forms—change the connection object.

Modification items 2 and 3 are included in the LogIn form window with the LogIn data table, and modification item 4 is located in the Faculty Form with the Faculty data table in the Microsoft Access database. Let's do these modifications one by one now.

6.7.2 Remove SP Form and Student Form

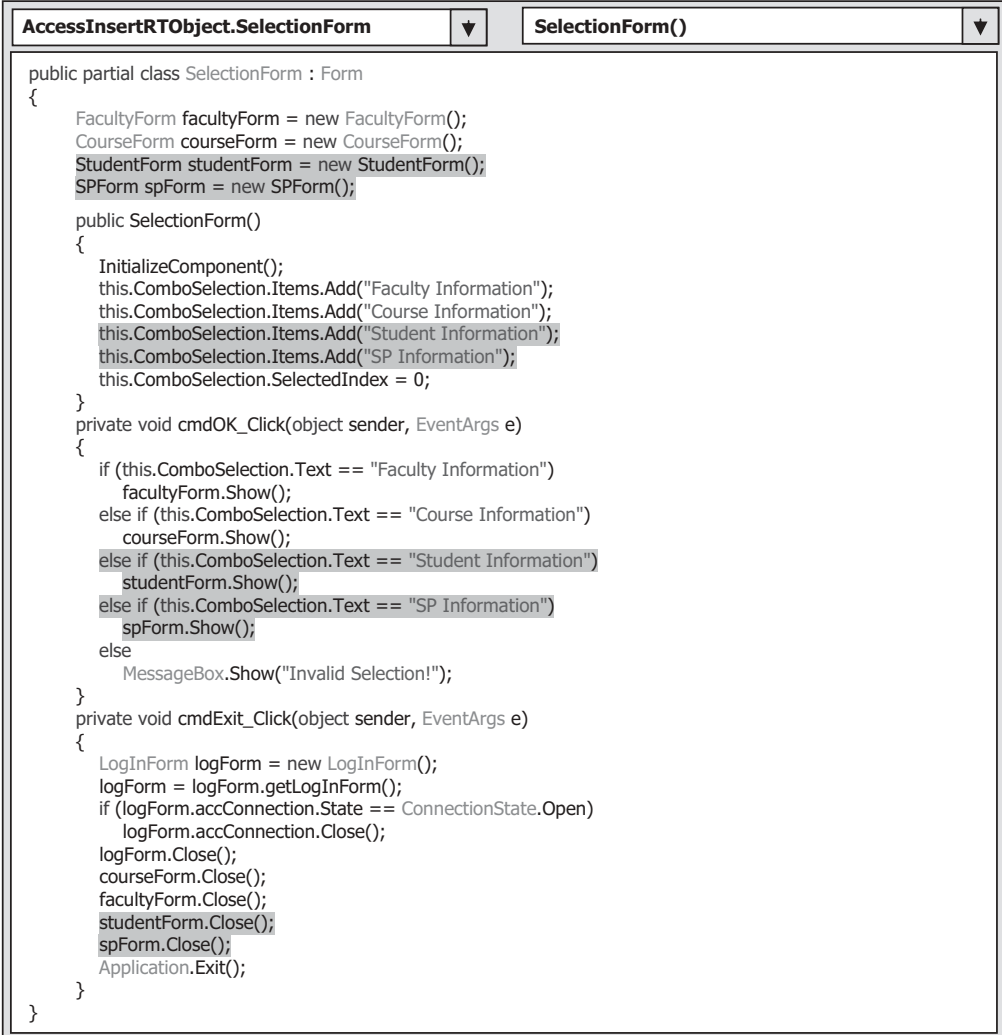
Since we will not use the SP and Student Forms for this data insertion query, we had better remove these two forms from our current project. Another reason to remove these two forms is that both contain stored procedure queries that are related to the SQL Server database, and the syntax of those queries are not compatible with our current database, Access 2007.

Open the Solution Explorer window and right-click on these two items, and click on the Delete to remove them. Then open the code window of the Selection Form and perform the following modifications as shown in Figure 6.60.

Let's have a close look at this piece of code to see the detailed modifications made to this form. All modified parts have been highlighted with shading.

- A. Remove two field-level form objects: `studentForm` and `spForm` since we do not need them in this project.
- B. Remove two `Add()` methods from the constructor of this form since we will not display these two form selections.
- C. Remove two `else if` selection blocks for items Student Information and SP Information since we will not display them and do not allow users to select them as the project runs.
- D. Remove two `Close()` methods for two form objects, `studentForm` and `spForm`, respectively, inside the Exit button's Click method.

Now let's modify the Connection strings used in this project to make them match the Access 2007 database queries.



```

AccessInsertRTOObject.SelectionForm SelectionForm()
public partial class SelectionForm : Form
{
    FacultyForm facultyForm = new FacultyForm();
    CourseForm courseForm = new CourseForm();
    A StudentForm studentForm = new StudentForm();
    SPForm spForm = new SPForm();

    public SelectionForm()
    {
        B InitializeComponent();
        this.ComboSelection.Items.Add("Faculty Information");
        this.ComboSelection.Items.Add("Course Information");
        this.ComboSelection.Items.Add("Student Information");
        this.ComboSelection.Items.Add("SP Information");
        this.ComboSelection.SelectedIndex = 0;
    }
    private void cmdOK_Click(object sender, EventArgs e)
    {
        C if (this.ComboSelection.Text == "Faculty Information")
            facultyForm.Show();
        else if (this.ComboSelection.Text == "Course Information")
            courseForm.Show();
        else if (this.ComboSelection.Text == "Student Information")
            studentForm.Show();
        else if (this.ComboSelection.Text == "SP Information")
            spForm.Show();
        else
            D MessageBox.Show("Invalid Selection!");
    }
    private void cmdExit_Click(object sender, EventArgs e)
    {
        LogInForm logForm = new LogInForm();
        logForm = logForm.getLogInForm();
        if (logForm.accConnection.State == ConnectionState.Open)
            logForm.accConnection.Close();
        logForm.Close();
        courseForm.Close();
        facultyForm.Close();
        studentForm.Close();
        spForm.Close();
        Application.Exit();
    }
}

```

Figure 6.60 Modifications to the Selection Form.

6.7.3 Modify Database Connection String

The Database Connection string is used to connect to the desired database based on the correct syntax and format related to the associated database. To make this modification, first we need to open the constructor of the LogIn form since the Connection string is defined in there.

Open the constructor of the LogIn form and change the name and the content of the Connection string, which is shown in Figure 6.61.

Let's have a close look at these modifications to see how they work.

- A. Add the OleDb namespace located at the namespace System.Data into this code window since we need to use data components related to OleDb Data Provider to perform any data query against the Access 2007 database later.

- B. Change the field-level Connection object's type to OleDbConnection and the object's name to accConnection, respectively. The accessing mode for this object is Public since we need to use this connection object in all of our Forms later to perform the data insertions.
- C. A Connection string has been modified based on database connection requirement of the Access 2007 and the procedure listed in Section 5.19.1. The Connection string indicates the detailed connection information, including the name of the data provider, the location and the name of the database, and username and password used to access the database. In this case, no username and password are utilized for our database, so those two items are missed from this connection string. You can add those two pieces of information if your database did utilize them. An important point to note is that the database driver for Microsoft Access 2007 is Microsoft ACE OLEDB 12.0, not Microsoft Jet OLEDB 4.0 which is used for Access 2003 or earlier. Our sample database file is named CSE_DEPT.accdb and located at the C:\database\Access directory.
- D. By using the keyword new, we create a new instance of the Connection class OleDbConnection with the Connection string we just built in step C.
- E. A try ... catch block is utilized here to try to catch any error caused by opening a connection between our project and the Access database file, and furthermore connecting our project to the database we selected. The advantage of using this kind of strategy is to avoid unnecessary system debug processes and simplify this debug procedure.

```

AccessInsertRTObject.LogInForm | LogInForm()
.....
A using System.Data.OleDb;
namespace AccessInsertRTObject
{
    public partial class LogInForm : Form
    {
B         public OleDbConnection accConnection;
        public LogInForm()
        {
C             InitializeComponent();
            string strConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;" +
D                                     "Data Source=C:\database\Access\CSE_DEPT.accdb;";
E             accConnection = new OleDbConnection(strConnectionString);
            try
            {
F                 accConnection.Open();
            }
            catch (OleDbException e)
            {
G                 MessageBox.Show("Access Error");
                 MessageBox.Show("Error Code = " + e.ErrorCode);
                 MessageBox.Show("Error Message = " + e.Message);
            }
            catch (InvalidOperationException e)
            {
H                 MessageBox.Show("Invalid Message = " + e.Message);
            }
            if (accConnection.State != ConnectionState.Open)
            {
                 MessageBox.Show("Database connection is Failed");
                 Application.Exit();
            }
        }
    }
}

```

Figure 6.61 Modification to the Connection string.

- F. A sequence of OleDbExceptionError messages will be displayed if an error related to the OleDb connection occurred.
- G. An InvalidOperationExceptionError message will also be displayed if an invalid operation error were encountered.
- H. This step is used to confirm that our database connection is successful. If not, an error message is displayed and the project is exited.

After a database connection is successfully made, we need to use this connection to access the database to perform our data query job. Go to the File/Save All to save those modifications. Next let's continue to modify the login query strings in the LogIn form.

6.7.4 Modify LogIn Query Strings

In this application, two LogIn buttons are used for this form since two different login methods are utilized. To save time and space, we only modify one method: the TableAdapter method. Open the TabLogIn button's Click method by double-clicking on the TabLogIn button from the LogIn form window, and perform the modifications shown in Figure 6.62 for this method.

Let's take a look at these modifications to see how they work.

- A. Most parts of this query string are working with the Microsoft Access database, and the only modifications are the LIKE symbol used in the WHERE clause. Change these two

```

AccessInsertRTOject.LogInForm  cmdTabLogIn_Click()
private void cmdTabLogIn_Click(object sender, EventArgs e)
{
    string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
    cmdString += "WHERE (user_name=@name ) AND (pass_word=@word)";
    OleDbDataAdapter LogInDataAdapter = new OleDbDataAdapter();
    DataTable accDataTable = new DataTable();
    OleDbCommand accCommand = new OleDbCommand();
    SelectionForm selfForm = new SelectionForm();
    accCommand.Connection = accConnection;
    accCommand.CommandType = CommandType.Text;
    accCommand.CommandText = cmdString;
    accCommand.Parameters.Add("@name", OleDbType.Char).Value = txtUserName.Text;
    accCommand.Parameters.Add("@word", OleDbType.Char, 8).Value = txtPassWord.Text;
    LogInDataAdapter.SelectCommand = accCommand;
    LogInDataAdapter.Fill(accDataTable);
    if (accDataTable.Rows.Count > 0)
    {
        //MessageBox.Show("LogIn is successful");
        selfForm.Show();
        this.Hide();
    }
    else
        MessageBox.Show("No matched username/password found!");
    accDataTable.Dispose();
    accCommand.Dispose();
    LogInDataAdapter.Dispose();
}

```

Figure 6.62 Modifications to the LogIn query string.

LIKE symbols to the equals symbol (=) before two parameters @name and @word, respectively. This is the syntax used in the Microsoft Access 2007 database.

- B.** Starting from this step, change the prefix for all OleDb classes used in this method from `Sql` to `OleDb`, and the prefix for all objects from `sql` to `acc`. All modifications have been highlighted in bold.
- C.** Change the prefix for all OleDb objects used in this method from `sql` to `acc`, the data type from `SqlDbType` to `OleDbType` for the `Add()` method. All modifications have been highlighted in bold.

You can perform similar modifications to the codes in the `ReadLogIn` and the `Cancel` button's `Click` methods, that is, change the prefix of all data classes from `Sql` to `OleDb`, and change the prefix of all objects from `sql` to `acc`.

Now let's go to the `Faculty` form to modify the `Faculty` table query string.

6.7.5 Modify Faculty Query String

First, make sure that the namespace `System.Data.OleDb`, which contains all data components related to the Access 2007 database, has been added into the namespace area in the code window of the `Faculty` Form, which is located at the top of this code window.

Then open the user-defined `UpdateFaculty()` method and change the prefix of all data classes from `Sql` to `OleDb` and the prefix of all data objects from `sql` to `acc`.

Now open the `Select` button's `Click` method by double-clicking on this button from the `Faculty` Form window, and perform the modifications shown in Figure 6.63 to this method.

Let's take a look at these modifications to see how they work.

- A.** The first modification is to the query string. As we did in the last section, most parts of this query string work for the Microsoft Access 2007 database, and the only modification is to change the statement `LIKE`, which is in the `WHERE` clause located before the dynamic parameter `@facultyName`, to the equals symbol (=) since this is the requirement of the Microsoft Access database.
- B.** Change the prefix of all OleDb data classes from `Sql` to `OleDb`, and change the prefix of all data objects from `sql` to `acc`. Steps **B**, **C**, and **E** are involved in these modifications. All modifications have been indicated in bold.
- D.** Change the prefix of the nominal data type, which is the second argument in the `Add()` method, from `SqlDbType` to `OleDbType`. This modification is also indicated in bold.

Another modification is for the user-defined `FillFacultyReader()` method. The data type of the argument should be changed from `SqlDataReader` to `OleDbDataReader`.

Before we can run the project to insert data into the database, we need to finish the rest of the modifications to other forms. Basically, these modifications are to change the connection object from `SqlConnection` to `OleDbConnection` for all other forms to match the connection requirement of the Microsoft Access database.

6.7.6 Modifications to Other Forms

As we did for the `LogIn` and `Faculty` Forms, the first modification we need to do is to make sure that the namespace `System.Data.OleDb` has been added into the namespace

```

AccessInsertRTOObject.FacultyForm cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
    string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty ";
A cmdString += "WHERE faculty_name = @name";
B OleDbDataAdapter FacultyDataAdapter = new OleDbDataAdapter();
    OleDbCommand accCommand = new OleDbCommand();
    OleDbDataReader accDataReader;
    DataTable accDataTable = new DataTable();
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
C accCommand.Connection = logForm.accConnection;
    accCommand.CommandType = CommandType.Text;
    accCommand.CommandText = cmdString;
D accCommand.Parameters.Add("@name", OleDbType.Char).Value = ComboName.Text;
    string strName = ShowFaculty(ComboName.Text);
    if (strName == "No Match")
        MessageBox.Show("No Matched Faculty Image found!");
    if (ComboMethod.Text == "DataAdapter Method")
    {
E FacultyDataAdapter.SelectCommand = accCommand;
        FacultyDataAdapter.Fill(accDataTable);
        if (accDataTable.Rows.Count > 0)
            FillFacultyTable(ref accDataTable);
        else
            MessageBox.Show("No matched faculty found!");
        accDataTable.Dispose();
        FacultyDataAdapter.Dispose();
    }
    else if (ComboMethod.Text == "DataReader Method")
    {
        accDataReader = accCommand.ExecuteReader();
        if (accDataReader.HasRows == true)
            FillFacultyReader(accDataReader);
        else
            MessageBox.Show("No matched faculty found!");
        accDataReader.Close();
    }
    else
        MessageBox.Show("Invalid Method Selected!");
}
}

```

Figure 6.63 Modifications to the Faculty query string.

area for all three forms: Selection, Course, and Insert Faculty Forms. Open the code window for all of those forms and add that namespace to the namespace area if it has not been added.

The second modification is to modify the Connection object located at those forms. In this project we need to use the LogIn, Faculty, Selection, Course, and Insert Faculty Form; therefore, we only need to modify the Connection object for the last three forms—Selection, Course, and the Insert Faculty Form—since we have finished the modifications for the first two forms.

The following methods contain this Connection object:

- The Select button's Click method and the Course listbox's SelectedIndexChanged method in the Course Form
- The Insert button's Click method in the Insert Faculty Form
- The Exit button's Click method in the Selection Form

Open those methods and change the Connection object from `sqlConnection` to `accConnection`. Also perform the following modifications to the associated items in all forms used in this project:

- Change the prefix of all OleDb-related classes from `Sql` to `OleDb`, and the prefix of all objects from `sql` to `acc`, respectively.
- Change the data type for all passed parameters from `SqlDbType` to `OleDbDbType`, from `SqlCommand` to `OleDbCommand`, and from `SqlDataReader` to `OleDbDataReader`.

The third modification is to change the query string, that is, the assignment operator in the WHERE clause, from `LIKE` to the equals symbol (`=`). The following methods contain the query strings:

- The Select button's Click method and the Course listbox's SelectedIndexChanged method in the Course Form

Open these methods and perform the modifications to the query strings.

The final modification is for the user-defined `InsertParameters()` method in the Insert Faculty Form. Change the data type of the inserting parameters from `SqlDbType` to `OleDbDbType` for all seven inserted parameters.

Now let's run the project to test our data insertion function. No faculty photo will be involved for this data insertion. However, you need to save the faculty photo to the associated folder before you can run this project if you want to include a faculty photo with this data insertion. Click on the Start Debugging button to run the project, and enter the suitable username and password, such as `jhenry` and `test`, to the LogIn form window, and then select the Faculty Information item from the Selection Form to open the Faculty Form. Click on the Insert button to open the Insert Faculty Form window, which is shown in Figure 6.64.

Enter the following data into the associated textboxes as a new faculty record:

- M99558 Faculty ID textbox
- Mattin Kims Faculty Name textbox
- Associate Professor Title textbox

Figure 6.64 Running status of the Insert Faculty Form window.

Figure 6.65 Data validation process.

- MTC-118 Office textbox
- 750-330-7788 Phone textbox
- University of Chicago College textbox
- mkims@college.edu Email textbox

Click on the Insert button to insert this new faculty record into the Faculty table in the database. Immediately the Insert button is disabled after this new data is inserted into the database. Now click on the Back button to return to the Faculty Form to validate this data insertion.

Click on the drop-down arrow of the ComboName combobox from the Faculty Form, and you can find that the name of the new inserted faculty **Mattin Kims** has been added in this box. Click on it to select this faculty and then click on the Select button to try to retrieve this new inserted record from the database and display it in this form. Click on the OK button to the “No Matched Faculty Image Found!” MessageBox since we did not include any faculty photo for this insertion. Immediately you can find that all pieces of information about this new faculty shows up in this form, which is shown in Figure 6.65.

This is the evidence that our data insertion to the Microsoft Access 2007 database is successful! Click on the Back and then on the Exit buttons to close the project. You can remove this added record from this database to keep our tables neat if you like. To do that, open the Access database and the Faculty table, right-click on the new inserted faculty record, and click on the Delete Record item to remove it.

A completed project AccessInsertRTOObject can be found at the folder DBProjects\Chapter 6 located at the accompanying ftp site (see Chapter 1).

6.8 INSERT DATA INTO ORACLE DATABASE USING RUNTIME OBJECTS

As we did in the last section for the Microsoft Access database, we can modify the SQLInsertRTOObject project and make it work for the Oracle database. First, let's modify

the project `SQLInsertRTOject`. Open Windows Explorer to create a new folder such as Chapter 6 if you have not done that, and then copy the project `SQLInsertRTOject` from the folder `DBProjects\Chapter 6` located at the accompanying ftp site (see Chapter 1) to our new folder Chapter 6. Change the name of this project to `OracleInsertRTOject`, which includes the following files:

- `OracleInsertRTOject.sln`
- `OracleInsertRTOject.csproj`
- `OracleInsertRTOject.exe`
- `OracleInsertRTOject.pdb`
- `OracleInsertRTOject.cshost.exe`
- `OracleInsertRTOject.xml`

To rename the last four files, you need to use the Project Properties window. Open the Project Properties window, and then change the names for the Assembly name and the Default namespace to `OracleInsertRTOject`. Then click on the Assembly Information button to open the Assembly Information dialog box, and change the Title and the Product to `OracleInsertRTOject`. Click on the OK button to close that dialog box.

Basically, we need to perform the following modifications:

1. Remove the Student and SP Forms—we do not need these two forms.
2. Modify the Selection Form—remove the Student and SP Form objects.
3. Add the Oracle reference and the Oracle namespace—all Oracle data components are defined there.
4. Modify the project namespace for the form windows—from `SQLInsertRTOject` to `OracleInsertRTOject`.
5. Modify the database Connection string—make it connect to the Oracle database.
6. Modify login username and password query strings—complete the login process.
7. Modify the Faculty table query string—select the correct faculty information.
8. Modify other forms—change the connection object and prefix of the data components.

To perform the first modification, open our new project `OracleInsertRTOject` and the Solution Explorer window. Right-click on the `SP Form.cs` and the `Student Form.cs` items, and click on the `Delete` item to remove these two forms. Modification item 2 is related to the Selection Form. Refer to Section 6.7.2 and Figure 6.60 to perform this modification. Modification items 5 and 6 are included in the `LogIn` Form window with the `LogIn` data table, and modification item 7 is performed in the `Faculty` Form with the `Faculty` data table in the Oracle database. Let's do these modifications one by one starting from modification item 3.

6.8.1 Add Oracle Reference and Oracle Namespace

Unlike Microsoft Access and SQL Server databases, Visual Studio.NET does not set the Oracle namespace as a default data namespace for the database programming. Therefore we need to add this namespace as a reference to our new project before we can utilize

this namespace. Perform the following operations to add this Oracle client reference to our project:

1. Open the Solution Explorer window.
2. Right-click on our project OracleInsertRTOBJECT and select the Add Reference item from the pop-up menu to open the Add Reference dialog box.
3. Browse down the list until you find the item System.Data.OracleClient, select it, and click on the OK button to add this reference into our project.

To confirm this addition, click on the Show All Files button from the Solution Explorer window, and then expand the Reference item, and you can find this reference we just added into the project.

Now let's open the code window of the LogIn form by clicking the View Code button from the Solution Explorer window. On the opened code window, move your cursor to the namespace area located at the top of this window and add one more Oracle namespace into this form:

```
using System.Data.OracleClient;
```

In this way, we finished adding the Oracle namespace in the LogIn Form window. Make the same additions to the following three forms:

- Faculty Form
- Course Form
- Insert Faculty Form

Now let's modify the project namespace for all form windows in this project.

6.8.2 Modify Project Namespaces

The modification we need to perform in this part is to change the project namespace for all following files:

1. Window Form Object files
2. Window Form Designer Object files
3. Project Main Entry file

Open the following Window Form object files, and change the namespace from SQLInsertRTOBJECT to OracleInsertRTOBJECT:

- LogIn Form.cs
- Selection.cs
- Faculty Form.cs
- Course Form.cs
- Insert Faculty Form.cs

Open all Window Form Designer files related to Window Form Object files listed above, and change the namespace from SQLInsertRTOBJECT to OracleInsertRTOBJECT. These files contain:

- LogIn Form.Designer.cs
- Selection.Designer.cs
- Faculty Form.Designer.cs
- Course Form.Designer.cs
- Insert Faculty Form.Designer.cs

Open the Project Main Entry file `Program.cs` and change the namespace from `SQLInsertRTObject` to `OracleInsertRTObject`. In this section, we will use the Insert Faculty Form as an example to illustrate how to insert a new faculty record into the Faculty table in the Oracle Database 10g Express Edition (XE).

Now let's modify the Connection strings used in this project to make them match the Oracle database queries.

6.8.3 Modify Database Connection String

The Database Connection string is used to connect to the desired database based on the correct syntax and format related to the associated database. To make this modification, first we need to open the constructor of the LogIn form since the Connection string is defined in there.

Open the constructor of the LogIn form and change the name and the content of the Connection string, which is shown in Figure 6.66.

Let's take a close look at these modifications to see how they work.

- Add the `OracleClient` namespace located at the namespace `System.Data` into this code window since we need to use data components related to the Oracle Data Provider to perform the data insertion against the Oracle database later.
- Change the field-level Connection object's type to `OracleConnection` and the object's name to `oraConnection`. The accessing mode for this object is `Public` since we need to use this Connection object in all of our forms later to perform the data insertions.
- A Connection string has been modified based on the database connection requirement of Oracle and the procedure listed in Section 5.19.1. The Connection string indicates the detailed connection information, including the name of the data provider, the location and the name of the database, and username and password used to access the database. In this case, no username and password are utilized for our database, so those two items are missing from this connection string. You can add those two pieces of information if your database did utilize them. The data source is XE, which is the database alias for our application. The user ID is our sample database name, `CSE_DEPT`, and the password is the one you created when you download the Oracle Database 10g XE. Refer to Section 5.20.2 to get a detailed description of this connection string. An addition operator (+) can be used to concatenate multiple substrings to form a complete connection string for the Oracle database.
- By using the keyword `new`, we create a new instance of the Connection class `oraConnection` with the Connection string we just built in step C.
- A `try...catch` block is utilized here to try to catch any errors caused by opening a connection between our project and the Oracle database file, and furthermore connecting our project to the database we selected. The advantage of using this kind of strategy is to avoid unnecessary system debug processes and simplify this debug procedure.

```

OracleInsertRTOObject.LogInForm
LogInForm()
.....
A using System.Data.OracleClient;
namespace OracleInsertRTOObject
{
  B public partial class LogInForm : Form
  {
    C public OracleConnection oraConnection;
    public LogInForm()
    {
      D InitializeComponent();
      E string oraString = "Data Source = XE;" +
        "User ID = CSE_DEPT;" +
        "Password = reback";
      F oraConnection = new OracleConnection(oraString);
      try
      {
        G oraConnection.Open();
      }
      catch (OracleException e)
      {
        H MessageBox.Show("Oracle Error");
        MessageBox.Show("Error Code = " + e.ErrorCode);
        MessageBox.Show("Error Message = " + e.Message);
      }
      catch (InvalidOperationException e)
      {
        MessageBox.Show("Invalid Connection Message = " + e.Message);
      }
      if (oraConnection.State != ConnectionState.Open)
      {
        MessageBox.Show("Database connection is Failed");
        Application.Exit();
      }
    }
  }
}

```

Figure 6.66 Modifications to the Connection string in the LogIn form.

- E.** A sequence of OracleExceptionError messages will be displayed if an error related to the Oracle connection has occurred.
- G.** An InvalidOperationExceptionError message will also be displayed if an invalid operation error were encountered.
- H.** This step is used to confirm that our database connection is successful. If not, an error message is displayed and the project is exited.

Go to the File|Save All to save those modifications. Next let's continue to modify the login query strings in the LogIn form.

6.8.4 Modify LogIn Query Strings

In this application, two LogIn buttons are used for this form since two different login methods are utilized. To save time and space, we only modify one method: the TableAdapter method. Open the TabLogIn button's Click method by double-clicking on the TabLogIn button from the LogIn form window, and perform the modifications shown in Figure 6.67 for this method.

```

OracleInsertRTOject.LogInForm cmdTabLogIn_Click()
private void cmdTabLogIn_Click(object sender, EventArgs e)
{
    A      string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
    cmdString += "WHERE (user_name=: name ) AND (pass_word=: word)";
    B      OracleDataAdapter LogInDataAdapter = new OracleDataAdapter();
    DataTable oraDataTable = new DataTable();
    OracleCommand oraCommand = new OracleCommand();
    SelectionForm selForm = new SelectionForm();
    C      oraCommand.Connection = oraConnection;
    oraCommand.CommandType = CommandType.Text;
    oraCommand.CommandText = cmdString;
    D      oraCommand.Parameters.Add("name", OracleType.Char).Value = txtUserName.Text;
    oraCommand.Parameters.Add("word", OracleType.Char, 8).Value = txtPassWord.Text;
    LogInDataAdapter.SelectCommand = oraCommand;
    LogInDataAdapter.Fill(oraDataTable);
    if (oraDataTable.Rows.Count > 0)
    {
        //MessageBox.Show("LogIn is successful");
        selForm.Show();
        this.Hide();
    }
    else
        MessageBox.Show("No matched username/password found!");
    oraDataTable.Dispose();
    oraCommand.Dispose();
    LogInDataAdapter.Dispose();
}

```

Figure 6.67 Modifications to the login query string in the LogIn form.

Let's take a look at these modifications to see how they work.

- A.** Most parts of this query string work with the Oracle database, and the only modifications are the LIKE symbol used in the WHERE clause. Change these two LIKE symbols to the assignment operator (=:) before two parameters `name` and `word`, respectively. This is the syntax used in the Oracle database.
- B.** Starting from step B, change the prefix for all Oracle classes used in this method from `Sql` to `Oracle`. All modifications have been highlighted in bold.
- C.** Starting from step C, change the prefix for all Oracle objects used in this method from `sql` to `ora`. All modifications have been highlighted in bold.
- D.** Modify the first nominal argument `@name` and `@word` in the `Add()` method by removing the `@` symbol before these two arguments since this is the query syntax of the Oracle database. Also change the nominal data type for these two arguments from `SqlDbType` to `OracleType`.

You can perform the similar modifications to the codes in the `ReadLogIn` and the `Cancel` button's methods.

Now let's go to the `Faculty` Form to modify the `Faculty` table query string.

6.8.5 Modify Faculty Query String

First, make sure that the namespace `System.Data.OracleClient`, which contains all data components related to the Oracle database, has been added into the namespace

```

OracleInsertRTOObject.FacultyForm  cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
    string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty ";
    cmdString += "WHERE faculty_name =: name";
    OracleDataAdapter FacultyDataAdapter = new OracleDataAdapter();
    OracleCommand oraCommand = new OracleCommand();
    OracleDataReader oraDataReader;
    DataTable oraDataTable = new DataTable();
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    oraCommand.Connection = logForm.oraConnection;
    oraCommand.CommandType = CommandType.Text;
    oraCommand.CommandText = cmdString;
    oraCommand.Parameters.Add("name", OracleType.Char).Value = ComboName.Text;
    string strName = ShowFaculty(ComboName.Text);
    if (strName == "No Match")
        MessageBox.Show("No Matched Faculty Image found!");
    if (ComboMethod.Text == "DataAdapter Method")
    {
        FacultyDataAdapter.SelectCommand = oraCommand;
        FacultyDataAdapter.Fill(oraDataTable);
        if (oraDataTable.Rows.Count > 0)
            FillFacultyTable(ref oraDataTable);
        else
            MessageBox.Show("No matched faculty found!");
        oraDataTable.Dispose();
        FacultyDataAdapter.Dispose();
    }
    else if (ComboMethod.Text == "DataReader Method")
    {
        oraDataReader = oraCommand.ExecuteReader();
        if (oraDataReader.HasRows == true)
            FillFacultyReader(oraDataReader);
        else
            MessageBox.Show("No matched faculty found!");
        oraDataReader.Close();
    }
    else
        MessageBox.Show("Invalid Method Selected!");
}
}

```

Figure 6.68 Modifications to the query string in the Faculty form.

area in the code window of the Faculty Form, which is located at the top of this code window.

Then open the user-defined UpdateFaculty() method and change the prefix of all data classes from `Sql` to `Oracle`, and the prefix of all data objects from `sql` to `ora`.

Now open the Select button's Click method by double-clicking on this button from the Faculty Form window, and perform the modifications shown in Figure 6.68 to this method.

Let's take a look at these modifications to see how they work.

- A.** The first modification is to the query string. As we did in the last section, most parts of this query string work for the Oracle database, and the only modification is to change the statement `LIKE`, which is in the `WHERE` clause and located before the dynamic parameter `@name`, to the Oracle assignment operator (`=:`) since this is the query requirement of the Oracle database. Also remove the `@` symbol before the parameter name.

- B.** Change the prefix of all Oracle data classes from the `Sql` to the `Oracle`, and change the prefix of all data objects from `sql` to `ora`. Steps involved in these modifications are **B**, **C**, and **E**. All modifications have been indicated in bold.
- C.** Remove the `@` symbol before the first argument “name” in the `Add()` method to meet the needs of the query requirement of the Oracle database. Change the prefix of the nominal data type, which is the second argument in the `Add()` method, from `SqlDbType` to `OracleType`. These modifications are also indicated in bold.

Another modification is for the user-defined `FillFacultyReader()` method. The data type of the argument should be changed from `SqlDataReader` to `OleDbDataReader`.

Before we can run the project to insert data into the database, we need to finish the rest of the modifications to other forms. Basically, these modifications change the connection object from `SqlConnection` to `OleDbConnection` for all other forms to match the connection requirement of the Microsoft Access database.

6.8.6 Modifications to Other Forms

As we did for the `LogIn` and `Faculty` forms, the first modification we need to do is to make sure that the namespace `System.Data.OracleClient` has been added into the namespace area for all three forms: `Selection`, `Course`, and `Insert Faculty` forms. Open the code window for all of these forms and add that namespace to the namespace area if it has not been added.

The second modification is to modify the `Connection` object located at these forms. In this project we need to use the `LogIn`, `Faculty`, `Selection`, `Course`, and the `Insert Faculty Form`; therefore, we only need to modify the `Connection` object for the last three forms—`Selection`, `Course`, and the `Insert Faculty Form`—since we have finished the modifications for the first two forms.

The following methods contain this `Connection` object:

- The `Select` button’s `Click` method and the `Course ListBox`’s `SelectedIndexChanged` method in the `Course Form`
- The `Insert` button’s `Click` method in the `Insert Faculty Form`
- The `Exit` button’s `Click` method in the `Selection Form`

Open those methods and change the `Connection` object from `SqlConnection` to `oraConnection`. Also perform the following modifications to the associated items in all forms used in this project:

- Change the prefix of all Oracle-related classes from `Sql` to `Oracle`, and the prefix of all objects from `sql` to `ora`, respectively.
- Change the data type for all passed parameters from `SqlDbType` to `OracleType`, from `SqlCommand` to `OracleCommand`, and from `SqlDataReader` to `OracleDataReader`.

The third modification is to change the query strings, that is, the assignment operator in the `WHERE` clause, from `LIKE@` to the Oracle assignment operator (`=:`). The following methods contain the query strings:

- The `Course ListBox`’s `SelectedIndexChanged` method in the `Course Form`—change “`LIKE @courseid`” to “`=: courseid`”.

- The Select button's Click method in the Course Form—change the joint query ON clause from “ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.faculty_name LIKE @name)” to “ON (Course.faculty_id = Faculty.faculty_id) AND (Faculty.faculty_name =: name)”
- The Insert button's Click method in the Insert Faculty Form—change the VALUES clause from “VALUES (@faculty_id,@faculty_name,@office,@phone, @college,@title,@email)” to “VALUES (:faculty_id, :faculty_name, :office,:phone, :college, :title,:email)”.

Open these methods and perform the modifications to the query strings.

The final modification is for the user-defined InsertParameters() method in the Insert Faculty Form. Change the data type of the inserting parameters from SqlDbType to OracleType for all seven inserted parameters. Also remove the @ symbol before each parameter in the Add() method.

At this point, we finished modifications to our new project OracleInsertRTOObject, and you can run the project to test the data insertion to the Oracle database. A completed project OracleInsertRTOObject can be found from the folder DBProjects\Chapter 6 located at the accompanying ftp site (see Chapter 1).

In the next section, we will discuss how to insert data using the LINQ query methods.

6.9 INSERT DATA INTO DATABASE USING LINQ QUERIES

As discussed in Chapter 4, Language-Integrated Query (LINQ) is a ground-breaking innovation in Visual Studio 2008 and the .NET Framework version 3.5 that bridges the gap between the world of objects and the world of data. In Visual Studio.NET you can write LINQ queries in C# with SQL Server databases, XML documents, ADO.NET DataSets, and any collection of objects that supports IEnumerable or the generic IEnumerable<T> interface.

LINQ can be considered as a pattern or model that is supported by a collection of so-called Standard Query Operator methods we discussed in Section 4.1, and all those Standard Query Operator methods are static methods defined in either IEnumerable or IQueryable classes in the namespace *System.Linq*. The data operated in LINQ query are object sequences with the data type of either IEnumerable<T> or IQueryable<T>, where T is the actual data type of the objects stored in the sequence.

LINQ is composed of three major components: LINQ to Objects, LINQ to ADO.NET, and LINQ to XML. Where LINQ to ADO.NET contains LINQ to DataSet, LINQ to SQL, and LINQ to Entities. Because there is no LINQ to the Oracle model available, we will concentrate our discussion on inserting data into the SQL Server database using the LINQ to SQL model.

Generally, the popular method to insert a new record into the database using the LINQ query follows the three steps listed below:

1. Create a new object that includes the column data to be submitted.
2. Add the new row object to the LINQ to SQL Table collection associated with the target table in the database.
3. Submit the change to the database.

Two ways can be used to add a new row object into the table: (1) using the Add() method and (2) using the InsertOnSubmit() method. However, both methods must be followed with the SubmitChanges() method to complete this new record insertion. In the following section, let's start with the data insertion using the LINQ to SQL queries to illustrate the second method.

6.9.1 Insert Data into SQL Server Database Using LINQ to SQL Queries

As discussed in Section 4.6, to use LINQ to SQL to perform data queries, we must convert our relational database to the associated entity classes using either SQLMetal or Object Relational Designer tools. Also we need to set up a connection between our project and the database using the DataContext object. Refer to Section 4.6.1 in Chapter 4 to get a clear picture of how to create entity classes and add the DataContext object to connect to our sample database CSE_DEPT.mdf. To perform data insertion using LINQ to SQL model, refer to Sections 4.6.2 and 4.6.2.2 in Chapter 4 to get a detailed description and the coding process of a real project QueryLINQSQL, which is a Console Application, to insert a new record into the Faculty table.

6.10 INSERT DATA INTO DATABASE USING STORED PROCEDURES

In this section, we discuss how to insert data into the database using stored procedures. We provided a very detailed introduction to the stored procedures in Section 5.19.2.7 in Chapter 5 and illustrated how to use this method to perform the data query for the Student form and Student table in Section 5.19.2.7.3 in Chapter 5. Refer to that part to get more detailed descriptions about the stored procedures.

We try to use the Course form and Course table to illustrate how to insert a new course record based on the selected faculty into the Course data table in this part. First, we discuss how to insert a new record into the Course table in the SQL Server database, and then we try to perform the similar function for the Oracle database. Some readers may have noted that we spent a lot of time to modify the codes in the Course form in the last project OracleInsertRTOObject, but we did not use that form in that project. The reason for this issue is that we will use that Course form to illustrate inserting data into the Oracle database in the next section.

6.10.1 Insert Data into SQL Server Database Using Stored Procedures

To save time and space, we can modify the project SQLInsertRTOObject to create a new project named SQLInsertRTOObjectSP and add one more form window to perform the data insertion using the stored procedures. Copy and paste the existing project SQLInsertRTOObject to the folder C:\Chapter 6 and rename it to our new project SQLInsertRTOObjectSP. Refer to Section 6.8.2 to perform the modifications of the project namespaces for all project files. Also refer to Section 6.8 to remove the SP Form and the Student Form as well as to perform the modifications to the Selection Form

since we do not need these two forms in this project. Recall that when we developed that project, an Insert button was added into the Course Form window. We can use this button to trigger a new form to perform the data insertion operation using the stored procedures. First, let's add one more form window into this new project. The name of this new form is Insert Course Form.

6.10.1.1 Add an Inserting Data Form Window: Insert Course Form

The function of this Insert Course form is: As the project runs, after the user has finished a correct login process and selected the item Course Information from the Selection Form, the Course Form window will be displayed. When the user clicks on the Insert button, the Insert Course Form window will appear. This form allows users to insert a new course record into the Course data table in the database using the stored procedures. The form also allows users to enter all pieces of information into the appropriate text-boxes for the new inserted course. By clicking on the Insert button, a new course record related to the selected faculty member is inserted into the database. However, if the user wants to reenter those pieces of information before finishing this insertion, the Cancel button can be used and all information entered will be erased. The Back button is used to allow users to return to the Course Form window to perform the validation to confirm that the data insertion is successful.

Go to the `Project|Add Windows Form` menu item to open the Add New Item dialog box. Keep the default Template, Windows Form, selected and enter the `Insert Course Form.cs` into the Name box as the name for this new form. Then click on the Add button to add this form into our project.

To save time and space, we can copy all controls of this Course Form from the project `SQLInsertWizard Project` we developed in this chapter. Open that project from the folder `DBProjects\Chapter 6` located at the accompanying ftp site (see Chapter 1) and open the Insert Course Form window, and then go to the `Edit|Select All` menu item to select all controls on that form window. Go to the `Edit|Copy` menu item to copy those items. Now open our project `SQLInsertRTObjectSP` and our new form Insert Course Form window, enlarge it to an appropriate size, and go to the `Edit|Paste` menu item to paste those controls into this form.

One important issue to note is that the `SQLInsertWizard Project` is developed using the Visual Studio.NET design tools and wizards, so some objects related to those design tools and wizards such as the Data Binding Source will be added into this form as you paste those controls. Because we don't need those objects in this runtime objects method, delete all of them from this new Insert Course Form window. To do that, right-click on the `CourseBindingSource` from the bottom of this form window and select the `Delete` item from the pop-up menu to remove it.

In addition to removing the components related to the design tools and wizards, you also need to perform the following modifications to this form:

- Remove the combobox control `ComboMethod` from this form since we only use one method, the `ExecuteNonQuery` method of the `Command` class, to execute the stored procedure to perform this data insertion.
- Remove the `Select` button from this form since we will not perform the data validation until we click on the `Back` button to return to the Course Form window. In other words, the data validation is performed in the Course Form.

Figure 6.69 Finished Insert Course Form window.

- Make sure the following properties of the form are set up:
 - Name: **InsertCourseForm**
 - Text: **CSE DEPT Insert Course Form**
 - AcceptButton: **cmdInsert** (select the Insert button as the default button)
 - StartPosition: **CenterScreen** (locate the form in the center of the screen)

Your finished form window, Insert Course Form, should match the one shown in Figure 6.69.

The detailed descriptions of the function of each control on this form can be found in Section 6.3.2 in this chapter. Simply speaking, the Faculty Name combobox is used to allow users to select the desired faculty member to insert a new course for that selected faculty. All seven textboxes allow users to enter the information for a new course to be inserted into the Course table. The Course ID textbox is a key textbox since the Insert button will be enabled if this textbox's content is changed, which means that a new course will be inserted. Generally, the Insert button should be disabled after a new course record has been inserted into the database to avoid multiple insertions of the same record. The Cancel button allows users to clean up the contents of all textboxes (except the Course ID) to reenter the course information. A new course record will be inserted into the database as the Insert button is clicked. The Back button is used to return to the Course form to perform the data validation for the new inserted course.

Next let's begin to develop the codes for this form. However, we need first to take care of our stored procedures issue.

6.10.1.2 Develop Stored Procedures of SQL Server Database

Recall that when we built our sample database CSE_DEPT in Chapter 2, there is no faculty name column in the Course table, and the only relationship that exists between the Faculty and the Course tables is the faculty_id, which is a primary key in the Faculty table but a foreign key in the Course table. As the project runs and the Insert Course Form window appears, the user needs to insert new course data based on the faculty name, not the faculty ID. Therefore, for this new course data insertion, we need to

perform two queries with two tables: First, we need to make a query to the Faculty table to get the faculty_id based on the faculty name selected by the user, and second we can insert a new course record based on the faculty_id we obtained from our first query. These two queries can be combined into a single stored procedure.

Compared with the stored procedure, another solution to avoid performing two queries is to use a joined table query to combine these two queries together to complete a course query, as we did for the Course Form in Section 5.19.2.5 in Chapter 5. However, it is more flexible and convenient to use stored procedures to perform this kind of multiple queries, especially when the queries are performed to multiple different data tables.

Now let's develop our stored procedures to combine these two queries to complete this data insertion. The stored procedure is named `dbo.InsertFacultyCourse`.

Open Visual Studio.NET and the Server Explorer window, click on the plus symbol icon that is next to CSE_DEPT database folder to connect to our database if this database was added into the Server Explorer before. Otherwise you need to right-click on the Data Connections folder to add and connect to our database. Refer to Section 5.19.2.7.3 in Chapter 5 for the detailed information of adding and connecting the database.

Right-click on the Stored Procedures folder and select the Add New Stored Procedure item to open the Add Procedure dialog box, and then enter the codes that are shown in Figure 6.70 into this new procedure. Do not forget to change the procedure's name to `dbo.InsertFacultyCourse`, which is located at the top of this procedure.

The function of this stored procedure is:

- A. All input parameters are listed in this part. The `@FacultyName` is selected by the user from the `ComboName` combobox, and all other input parameters should be entered by the user to the associated textbox in the `Insert Course Form` window.
- B. A local variable `@FacultyID` is declared, and it is used to hold the returned value from the execution of the first query to the Faculty table in step C.
- C. The first query is executed to pick up the matched `faculty_id` from the Faculty table based on the first input parameter, `@FacultyName`.

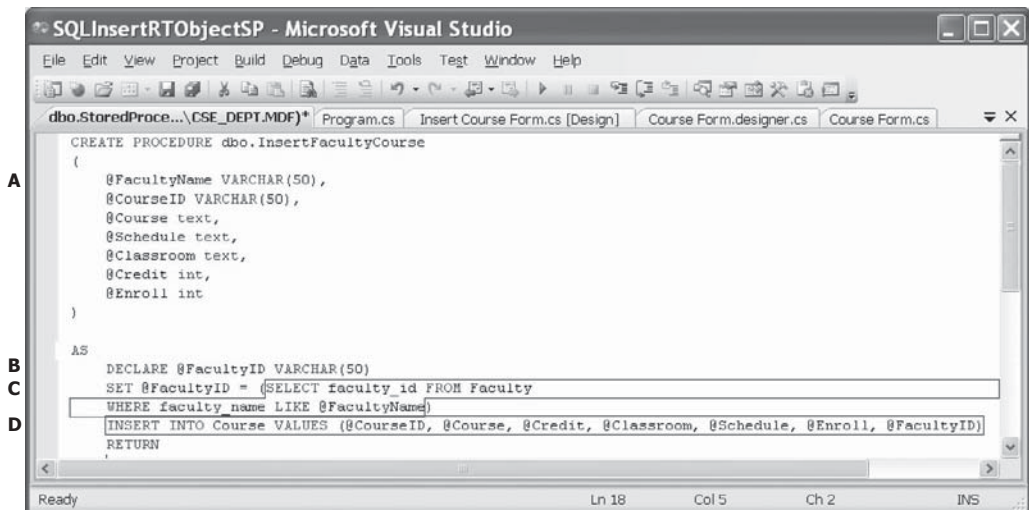


Figure 6.70 Stored procedure `dbo.InsertFacultyCourse`.

- D. The second query is used to insert a new course record into the Course table. The last parameter in the VALUES parameter list is the @FacultyID, which is obtained from the first query.

The coding for this stored procedure is simple and easy to be understood. One point you should know is the order of parameters in the VALUES parameter list. This order must be identical with the column order in the Course table. Otherwise, an error may be encountered when this stored procedure is saved.

Go to the FileSave StoredProcedure1 menu item to save this stored procedure. Now let's test this stored procedure in the Server Explorer environment to make sure that it works fine.

Right-click on our new stored procedure dbo.InsertFacultyCourse from the Server Explorer window, and click on the Execute item from the pop-up menu to open the Run Stored Procedure dialog box. Enter the input parameters into the associated box for a new course record, and your finished parameters dialog box is shown in Figure 6.71.

Click on the OK button to run this stored procedure. The running result is displayed in the Output window at the bottom, which is shown in Figure 6.72.

To confirm this data insertion, open the Course table by first expanding the Tables folder in the Server Explorer window and then right-clicking on the Course folder, and

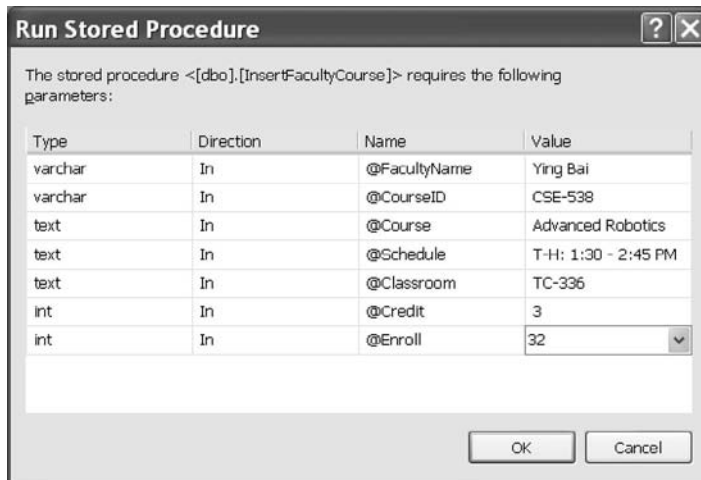


Figure 6.71 Run Stored Procedure dialog box.

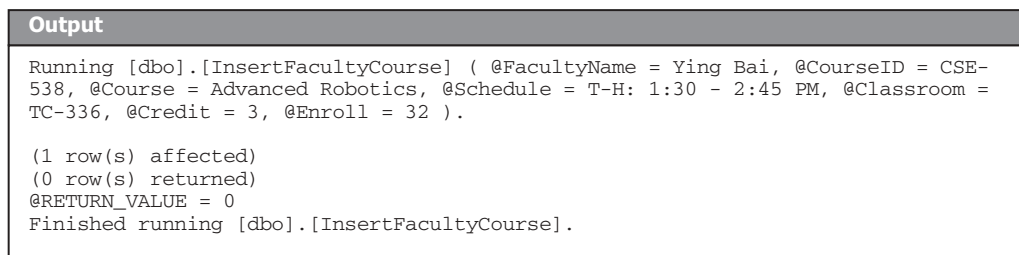


Figure 6.72 Running result of the stored procedure.

select the item **Show Table Data**. Browse the Course table until the last row, and you can find that a new course, CSE-538: Advanced Robotics, has been inserted into this table. OK, our stored procedure is successful!

Next we need to develop the codes in the Visual C#.NET environment to call this stored procedure to insert a new course record into the database from our user interface.

6.10.1.3 Develop Codes to Call Stored Procedures to Insert Data into Course Table

The coding for this data insertion is divided into three steps: the data validation before the data insertion, data insertion using the stored procedure, and the data validation after the data insertion. The purpose of the first step is to confirm that all inserted data that is stored in each associated textbox should be complete and valid. In other words, all textboxes should be nonempty. The third step is used to confirm that the data insertion is successful, in other words, the new inserted data should be in the desired table in the database and can be read back and displayed in the Course form window. Let's begin with the coding for the first step.

6.10.1.3.1 Validate Data Before Data Insertion and Startup Coding First, let's take care of the startup coding. The so-called startup coding includes adding the namespace related to SQL Server Data Provider, the coding for the constructor of the InsertCourseForm class, field-level variables declarations, and coding for the Cancel and the Back buttons' Click methods. Open the code window of the Insert Course Form window and enter the codes shown in Figure 6.73 into this code window.

The codes added into the constructor are used to add all faculty names into the ComboName control to allow users to select one when performing the new course insertion.

Let's take a look at the following pieces of codes to see how they work.

- A. The System.Data.SqlClient namespace is added into the namespace area since we need to use data components related to SQL Server Data Provider to perform this new course insertion.
- B. A field-level string array CourseInfo() is created first, and this array is used to store all information related to the new course to be inserted into the database.
- C. In the constructor of the InsertCourseForm class, all faculty members are added into the combobox ComboName by executing the Add() method, and the first item is selected as the default faculty member. The user can select a desired faculty member from this combobox and insert a new course for that selected faculty as the project runs.
- D. The user-defined InitCourseInfo() method is used to set up a one-to-one relationship between each item in the CourseInfo string array and each textbox that contains a piece of new course information. In this way, it is easier to scan and check each textbox to make sure that none of them is empty when the user-defined CheckCourseInfo() method is executed later.
- E. To check each textbox, a for loop is utilized to scan the CourseInfo array. A warning message would be displayed, and the method returns a nonzero value to the calling method to indicate that this checking has failed if any textbox (except the Faculty ID) is empty. Otherwise a zero is returned to indicate that this checking is successful. A trick is that the for loop starts from 1, not 0, which means that this check does not include the Faculty

SQLInsertRTOjectSP.InsertCourseForm	InsertCourseForm()
<pre> A using System.Data.SqlClient; namespace SQLInsertRTOjectSP { public partial class InsertCourseForm : Form { B string[] CourseInfo = new string[7]; public InsertCourseForm() { C InitializeComponent(); ComboName.Items.Add("Ying Bai"); ComboName.Items.Add("Satish Bhalla"); ComboName.Items.Add("Black Anderson"); ComboName.Items.Add("Steve Johnson"); ComboName.Items.Add("Jenney King"); ComboName.Items.Add("Alice Brown"); ComboName.Items.Add("Debby Angles"); ComboName.Items.Add("Jeff Henry"); ComboName.SelectedIndex = 0; } D private void InitCourseInfo() { CourseInfo[0] = txtFacultyID.Text; CourseInfo[1] = txtCourseID.Text; CourseInfo[2] = txtCourse.Text; CourseInfo[3] = txtSchedule.Text; CourseInfo[4] = txtClassRoom.Text; CourseInfo[5] = txtCredits.Text; CourseInfo[6] = txtEnroll.Text; } E private int CheckCourseInfo() { int pos = 0, check = 0; for (pos = 1; pos <= 6; pos++) { if (CourseInfo[pos] == String.Empty) check++; } return check; } F private void cmdBack_Click(object sender, EventArgs e) { this.Close(); } G private void cmdCancel_Click(object sender, EventArgs e) { txtFacultyID.Text = String.Empty; txtCourse.Text = String.Empty; txtSchedule.Text = String.Empty; txtClassRoom.Text = String.Empty; txtCredits.Text = String.Empty; txtEnroll.Text = String.Empty; } } </pre>	

Figure 6.73 Startup coding.

ID textbox. That is correct because at this moment we may not have any knowledge for this information.

- F. The Cancel button's Click method is used to clean up all textboxes' content, except the Course ID. The reason for this is that after a new course is inserted into the database, the Insert button will be disabled to avoid multiple insertions of the same course. But this button will be enabled again as soon as the content of the Course ID textbox is changed, which means that a different new course will be inserted. In order to avoid enabling this button mistakenly, we keep the Course ID textbox, unchanged, that is, not cleaned.
- G. The coding for the Back button's Click method is easy, and the `this.Close()` method is used to close the Insert Course Form window since we do not need this form when a new course record has been inserted into the database.

Now let's do our coding for the Insert button's Click method to perform the data validation before the data insertion.

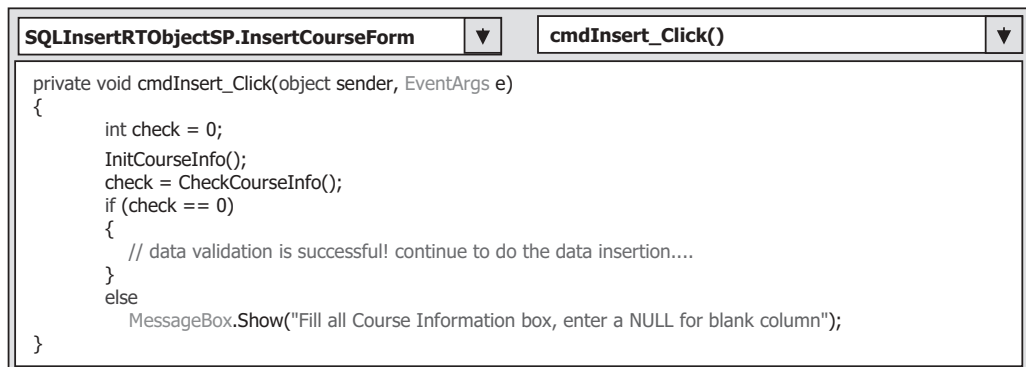
This data validation can be performed by calling the `InitCourseInfo()` method and the `CheckCourseInfo()` method, which we discussed above, in the Insert button's Click method. Open the Insert button's Click method by double-clicking on the Insert button from the form window of the Insert Course Form, and enter the codes that are shown in Figure 6.74 into this method.

The function of this piece of code is straightforward and easy to understand. First, the `InitCourseInfo()` method is called to set up a one-to-one relationship between each item in the `CourseInfo()` array and each associated textbox that stores a piece of course information. Next the `CheckCourseInfo()` method is executed to make sure that the new course information is completed and valid. In other words, no textbox is empty. A returned zero indicates that no empty textbox has been detected and this data validation is successful. Otherwise a warning message is displayed to remind users to refill all textboxes if a nonzero is returned.

Now let's develop and complete the codes to call the stored procedure to perform the new course insertion.

6.10.1.3.2 Develop Codes to Call Stored Procedures Open the Insert button's Click method and then add the codes shown in Figure 6.75 into this method.

The codes we developed in the last section have been highlighted with shading. Let's take a look at these new added codes to see how they work.



```

private void cmdInsert_Click(object sender, EventArgs e)
{
    int check = 0;
    InitCourseInfo();
    check = CheckCourseInfo();
    if (check == 0)
    {
        // data validation is successful! continue to do the data insertion....
    }
    else
        MessageBox.Show("Fill all Course Information box, enter a NULL for blank column");
}

```

Figure 6.74 First coding for the Insert button's Click method.

```

SQLInsertRTOBJECTSP.InsertCourseForm  cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
  A   int check = 0, intInsert = 0;
  B   string cmdString = "dbo.InsertFacultyCourse";
  C   SqlCommand sqlCommand = new SqlCommand();
      LogInForm logForm = new LogInForm();
      InitCourseInfo();
      check = CheckCourseInfo();
      if (check == 0)
      {
  D       logForm = logForm.getLogInForm();
  E       sqlCommand.Connection = logForm.sqlConnection;
          sqlCommand.CommandType = CommandType.StoredProcedure;
          sqlCommand.CommandText = cmdString;
          InsertParameters(ref sqlCommand);
  F       intInsert = sqlCommand.ExecuteNonQuery();
  G       sqlCommand.Dispose();
  H       if (intInsert == 0)
  I         MessageBox.Show("The data insertion is failed");
  J       else
          {
              cmdCancel.PerformClick();    // clean up all faculty information
              cmdInsert.Enabled = false;
          }
      }
      else
  I     MessageBox.Show("Fill all Course Information box, enter a NULL for blank column");
}
}

```

Figure 6.75 Modifications to the Insert button's Click method.

- A. A local integer variable `intInsert` is created and initialized with zero. This variable works as a value holder to hold the returned value of executing the insert method—`ExecuteNonQuery()`.
- B. The query string is assigned with the name of the stored procedure developed in Section 6.10.1.2. One of the most important points to call stored procedures is that the query string must be exactly identical with the name of the stored procedure to be called. Visual C#.NET project could not find the stored procedures, and a runtime error would be encountered if the query string does not match the name of the stored procedure.
- C. Some other components and variables used in this method are declared here. The Command object and an instance of the LogIn form class are also created here. Since we need to use the Connection object created in the LogIn form window for this data insertion, we need to create an instance to access it.
- D. The `getLogInForm()` method defined inside the LogIn Form class is called to get the current instance of the LogInForm class and assign it to our new created instance `logForm`. In this way, we can use and access our original LogInForm instance, and furthermore to use and access the Connection object, we create that object instead of using a new instance of the LogInForm class.
- E. The Command object is initialized with the suitable components. Two important points to be noted are `CommandType` and `CommandText`. The former must be assigned with the property of `StoredProcedure` to indicate that the command type of this Command object is a stored procedure, and a stored procedure will be called when this Command is executed. Second, the name of the stored procedure to be called must be assigned to the

CommandText property of the Command object to indicate to the Visual C#.NET where to find this stored procedure.

- F.** The user-defined InsertParameters() method, whose detailed coding is shown in Figure 6.76, is executed to fill all input parameters into the Parameters collection of the Command object to finish the initialization of the Command object. A passing-by-reference mode is used for this parameter passing since we want to make sure that all modifications to the passed parameters are permanent.
- G.** The ExecuteNonQuery() method of the Command class is executed to call the stored procedure to perform the new data insertion.
- H.** The Command object is cleaned up after the data insertion.
- I.** The ExecuteNonQuery() method will return an integer to indicate whether this calling is successful or not. The returned value equals the number of rows or records that have been successfully inserted into the database. A zero means that no row or record has been inserted into the database, and this data insertion has failed. In that case, a warning message is displayed.
- J.** Finally, after one data insertion, the information stored in all textboxes, except the Course ID, are cleaned up to make it ready for the next data insertion. Also the Insert button is disabled to avoid multiple insertions of the same data into the database.

The detailed coding for the user-defined InsertParameters() method is shown in Figure 6.76.

The function of this method is to assign each piece of information stored in each textbox to the associated input parameter we defined in the stored procedure `dbo.InsertFacultyCourse`. One key point is that the name of each passed parameter, which is represented as a string and located at the first argument's position, must be identical with the name of each input parameter we defined in the stored procedure. For example, the name of the parameter `@FacultyName` used here must be identical with the input parameter's name `@FacultyName` that exists in the input parameter's list we defined at the beginning of the stored procedure `dbo.InsertFacultyCourse`. A run time error would be encountered if a name of parameter is not matched with the associated parameter's name in the stored procedure as the project runs. Refer to Figure 6.70 for the detailed list of all parameters' names defined in the stored procedure.

Now we have finished coding for this data insertion operation. Before we can run the project to test the functionality of this data insertion, we need to figure out how to open and start the Insert Course Form to perform this new course insertion. There are

SQLInsertRTOBJECTSP.InsertCourseForm	InsertParameters()
<pre>private void InsertParameters(ref SqlCommand cmd) { cmd.Parameters.Add("@FacultyName", SqlDbType.Char).Value = ComboName.Text; cmd.Parameters.Add("@CourseID", SqlDbType.Char).Value = txtCourseID.Text; cmd.Parameters.Add("@Course", SqlDbType.Char).Value = txtCourse.Text; cmd.Parameters.Add("@Schedule", SqlDbType.Char).Value = txtSchedule.Text; cmd.Parameters.Add("@Classroom", SqlDbType.Char).Value = txtClassRoom.Text; cmd.Parameters.Add("@Credit", SqlDbType.Char).Value = txtCredits.Text; cmd.Parameters.Add("@Enroll", SqlDbType.Char).Value = txtEnroll.Text; }</pre>	

Figure 6.76 Coding for the InsertParameters method.


```

namespace SQLInsertRTOjectSP
{
    public partial class CourseForm : Form
    {
        private TextBox[] CourseTextBox = new TextBox[6];
A         InsertCourseForm InsertCourse = new InsertCourseForm(); // added in Nov. 26, 2008
        .....

        private void cmdInsert_Click(object sender, EventArgs e)
        {
B             InsertCourse.Show(); // added in Nov. 26, 2008
        }
    }
}

```

Figure 6.77 Modifications to the codes in the Course Form window.

two methods to open the Insert Course Form as the project runs; one method directly starts this form and inserts data, and another method is to use the Course Form to trigger this form. We prefer to use the second method since we need to use the Course Form to perform the data validation after a new record is inserted in the Course table.

Let's make two modifications to the Course Form to trigger the Insert Course Form.

- The first modification is to add a field-level object of the Insert Course Form class, and this object is used to start the Insert Course Form window as the Insert button in the Course Form is clicked. To do that, add the following command under the class header, which is shown in step **A** in Figure 6.77.
- The second modification is to call the Show() method of the Insert Course Form object to display that form when the Insert button in the Course Form is clicked. To do that, open the Insert button's Click method in the Course Form and add the command shown in step **B** in Figure 6.77 into this method.

Now let's run the project to test the new data insertion using the stored procedures. Click on the Start Debugging button to start the project, enter the suitable username and password, such as jhenry and test to the LogIn form, and select the Course Information item from the Selection form to open the Course Form window. Click on the Insert button to open the Insert Course Form window to perform the new course insertion.

Enter the following data into the associated textbox as the information for a new course and leave the Faculty ID textbox empty since we don't know that piece of information at this moment, and this information can be retrieved by the stored procedure:

- Ying Bai Selected from the Faculty Name combobox
- CSE-668 Course ID textbox
- Modern Controls Course Title textbox
- M-W-F: 9:00–9:55 AM Schedule textbox
- TC-309 Classroom textbox
- 3 Credits textbox
- 30 Enrollment textbox

Your finished information window should match the one shown in Figure 6.78.

Figure 6.78 Running status of the Insert Course Form window.

Click on the Insert button to call the stored procedure to insert this new course record into the database. Immediately the Insert button is disabled after this data insertion. Is our data insertion successful? To answer this question, we need to perform the data validation in the next section.

6.10.1.3.3 Validate Data After Data Insertion Now click on the Back button to return to the Course Form window to validate this data insertion.

Click on the drop-down arrow of the combobox control ComboName and select the faculty member Ying Bai from the list since we have just inserted a new course for this faculty in the last section. Click on the Select button to try to retrieve all courses, including the new inserted course, taught by this faculty and display them in the CourseListBox in this form window. All courses taught by the selected faculty are displayed in the CourseList box. The last item is just the course we added into the Course table in the last section. Click on that item and all information related to that new course is displayed in this form, which is shown in Figure 6.79. This is the evidence that our data insertion using the stored procedure is successful!

A completed project SQLInsertRTObjectSP that includes the data insertion using the stored procedure can be found from the folder DBProjects\Chapter 6 located at the accompanying ftp site (see Chapter 1).

6.10.2 Insert Data into Oracle Database Using Stored Procedures

There is no significant difference between inserting data into the SQL Server database and Oracle database using the stored procedures. One of the most important differences is the prototype of the stored procedure defined in two different databases. In Oracle database, a package component must be used to contain the stored procedure if the stored procedure needs to return any data item.

In this section, we still want to use the Course Form and Course table to show readers how to insert a new course record into the Course table using the stored procedure in the Oracle database environment.

The screenshot shows a window titled "CSE DEPT Course Form" with a standard Windows title bar (minimize, maximize, close). The window is divided into several sections:

- Name and Method:** Contains two dropdown menus. "Faculty Name" is set to "Ying Bai" and "Query Method" is set to "DataAdapter Method". To the right of these are three buttons: "Select", "Insert", and "Back".
- Course List:** A list box containing the following course IDs: CSC-132B, CSC-234A, CSE-434, CSE-438, CSE-538, CSE-566, and CSE-668. The course "CSE-668" is currently selected and highlighted.
- Course Information:** A form with five input fields:
 - Course:** Modern Controls
 - Schedule:** M-W-F: 9:00 - 9:55 AM
 - Classroom:** TC-309
 - Credits:** 3
 - Enrollment:** 30

Figure 6.79 Data validation process.

To illustrate how to insert data into the Oracle database using stored procedures or packages, we will utilize the following three steps:

1. Develop a user interface—Insert Course Form window.
2. Develop a package to contain stored procedures to perform inserting data into the Oracle database.
3. Develop the codes to call the package developed in step 2 to complete the data insertion function.
4. Validate the data insertion using the Course Form window.

Step 1 is similar to the step we did in the last project `SQLInsertRTOObjectSP`, and the only difference is that the Faculty ID textbox should be removed from this form window since it does not contain any input information. Steps 3 and 4 are very similar to those steps we developed for the same project in the last section. We will emphasize and highlight the different coding when we develop those steps below.

To save time and space, we modify the project `OracleInsertRTOObject` we developed in Section 6.8 and create a new project named `OracleInsertRTOObjectSP`. Refer to Section 6.8 to get more detailed information on how to modify a current project to create a new project.

Now let's start from step 2, develop the stored procedure in the Oracle databases.

6.10.2.1 Develop Stored Procedures in Oracle Database

A very detailed discussion of creating and manipulating packages and stored procedures in the Oracle database is provided in Section 5.20.3.6. in Chapter 5. Refer to that section to get more detailed information for creating Oracle's stored procedures.

The topic we are discussing in this section is the insertion of data into the database. Thus no returned data is needed for this section. Therefore we only need to create stored procedures in the Oracle database, not package, to perform the data insertion function.

As discussed in Section 5.20.3.6 in Chapter 5, different methods can be used to create Oracle's stored procedures. In this section, we will use the Object Browser page provided by Oracle Database 10g XE to create our stored procedures

Open the Oracle Database 10g XE home page by going to Start|All Programs|Oracle Database 10g Express Edition|Go To Database Home Page items. Finish the login process by entering the correct username and password (in our case, the username is our database name CSE_DEPT and the password is reback). Click on the Object Browser and select the Create|Procedures item to open the Create Procedure window. Click the Create button and select the Procedure icon from the list to open this window. The opened window is shown in Figure 6.80.

Enter `InsertFacultyCourse` into the Procedure Name box and keep the Include Argument checkbox checked, and click on the Next button to go to the next page.

The next window is used to enter all input parameters. For this stored procedure we need to perform two queries; therefore, we have seven input parameters. The first query is to get the `faculty_id` from the Faculty table based on the faculty name that is an input and selected by the user from the ComboName combobox control from the Course Form window, and the second query is to insert a new course record that contains six pieces of information related to a new course into the Course table based on the `faculty_id` that is obtained from the first query. The seven input parameters are Faculty Name, Course ID, Course Title, Credit, Classroom, Schedule, and Enrollment. The first input parameter Faculty Name is used by the first query, and the following six input parameters are used by the second query.

Enter those input parameters one by one into the argument box. The point is that the data type of each input parameter must be identical with the data type of each data column in the Course table. Refer to Section 2.11.2.3 to get a detailed list of data types used for those data columns in the Course data table.

For the Input/Output type of the parameters, select **IN** for all seven parameters since no output is needed for this data insertion query. Your finished argument list should match the one shown in Figure 6.81.

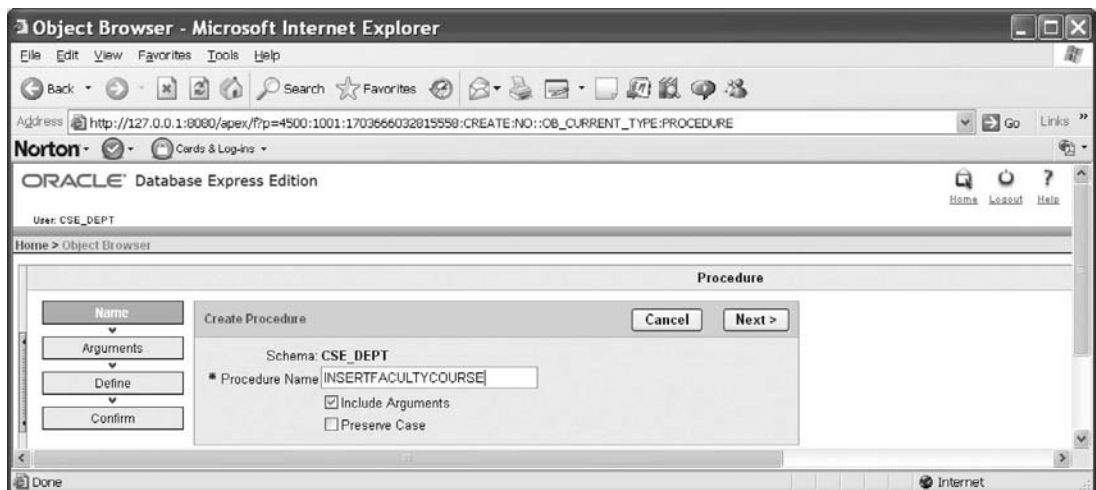


Figure 6.80 Opened Create Procedure window.

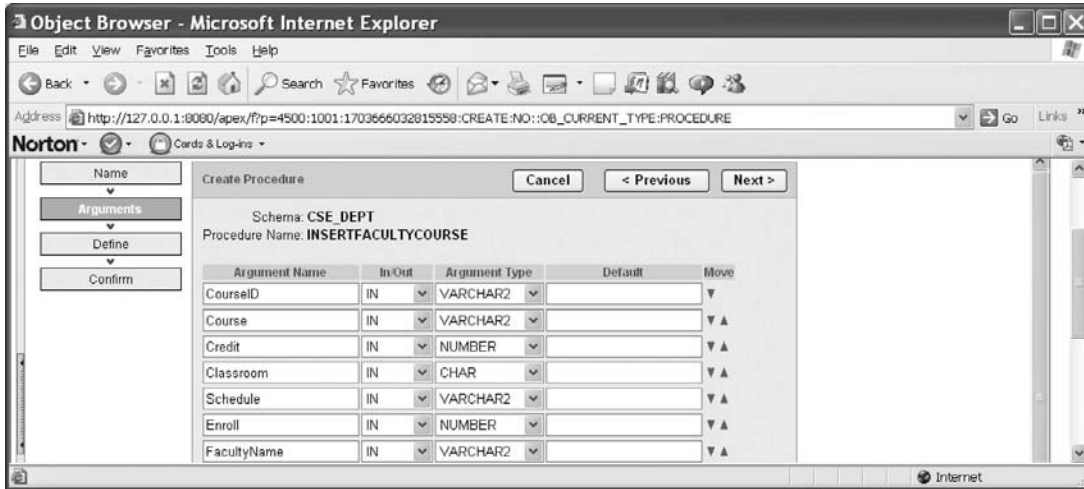


Figure 6.81 Finished argument list.

Click on the **Next** button to go to the procedure-defining page. Enter the codes shown in Figure 6.82a into this new procedure as the body of the procedure, using the language called Procedural Language Extension for SQL or PL-SQL. Then click on the **Next** and the **Finish** buttons to confirm creating this procedure. Your finished stored procedure should match the one that is shown in Figure 6.82b.

Seven input parameters are listed at the beginning of this procedure with the keyword **IN** to indicate that these parameters are inputs to the procedure. The intermediate parameter `faculty_id` is obtained from the first query in this procedure from the Faculty table. The data type of each parameter is indicated after the keyword **IN**, and it must be identical with the data type of the associated data column in the Course table. An **IS** command is attached after the procedure header to indicate that an intermediate query result, `faculty_id`, will be held by a local variable `facultyID` declared later.

Two queries are included in this procedure. The first query is used to get the `faculty_id` from the Faculty table based on the input parameter `FacultyName`, and the second query is to insert seven input parameters into the Course table based on the `faculty_id` obtained from the first query. A semicolon must be attached after each PL-SQL statement and after the command end.

One important issue is that you need to create one local variable `facultyID` and attach it after the **IS** command as shown in Figure 6.83, and this coding has been highlighted with a shading. Click on the **Edit** button to add this local variable. This local variable is used to hold the returned `faculty_id` from the execution of the first query.

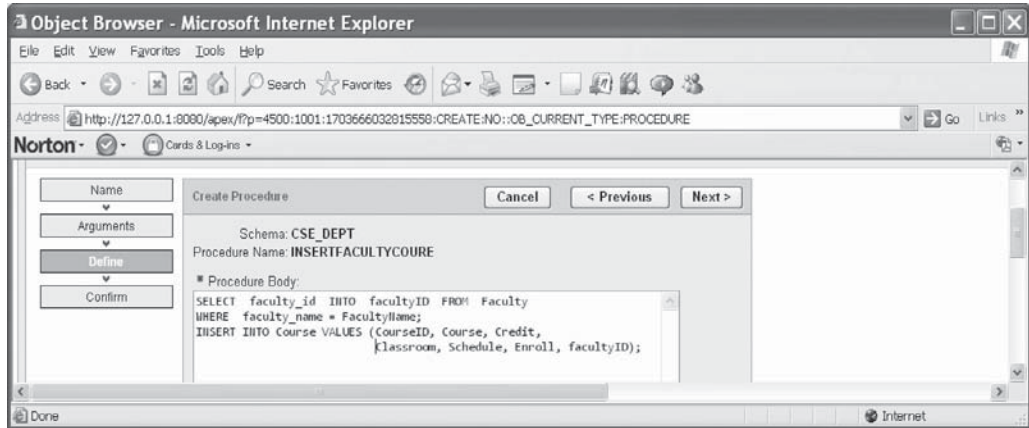
Another important issue in distributing the input parameters or arguments in an **INSERT** command is that the order of those parameters or arguments must be identical with the order of the data columns in the associated data table. For example, in the Course table, the order of the data columns is `course_id`, `course`, `credit`, `classroom`, `schedule`, `enrollment`, and `faculty_id`. Accordingly, the order of input parameters placed in the **INSERT** argument list must be identical with the data columns' order displayed in above.

```

SELECT faculty_id INTO facultyID FROM Faculty
WHERE faculty_name = FacultyName;
INSERT INTO Course VALUES (CourseID, Course, Credit,
Classroom, Schedule, Enroll, facultyID);

```

(a)



(b)

Figure 6.82 Stored procedure body.

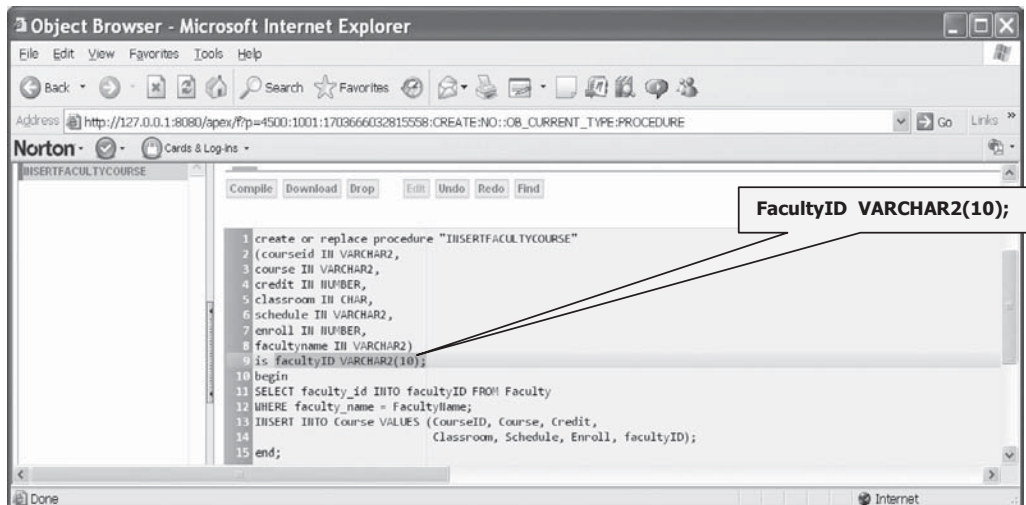


Figure 6.83 Completed stored procedure.

To make sure that this procedure is error free, we need to compile it first. Click on the **Compile** button to compile and check our procedure. A successful compilation message should be displayed if our procedure is a bug-free stored procedure.

Close the Oracle Database 10g Express Edition by clicking on the **Close** button. Next we need to develop the codes in our Visual C#.NET project to call this stored procedure to perform the data insertion function.

6.10.2.2 Develop Codes to Call Stored Procedures to Insert Data into Course Table

Basically, the coding to be developed in this section is very similar to the coding we developed in Section 6.10.1.3. The procedure of this coding is to develop a new form `InsertCourseForm` to enable users to enter seven pieces of information related to a new course into the `Course` table via this form. In order to save time and space, you can insert that `InsertCourseForm` from the project `SQLInsertRTOBJECTSP` to this project. To do that, open our new project `OracleInsertRTOBJECTSP` and go to the menu item `Project/Add Existing Item` to open the `Add Existing Item` dialog box. Browse to the project `SQLInsertRTOBJECTSP` and select the item `Insert Course Form.cs`, and click on the `Add` button to add this form into our new project. In the following sections, we only emphasize and highlight the important and different parts of the coding for the Oracle database.

6.10.2.2.1 Validate Data Before Data Insertion and Startup Coding This section's coding is identical with the coding Section in 6.10.1.3.1. The only difference is the namespace related to the Oracle data components. Since we are using the Oracle database, the namespace should be `System.Data.OracleClient`, which contains all data components provided by the Oracle Data Provider. In Section 6.10.2, we modified the project `OracleInsertRTOBJECT` and created a new project `OracleInsertRTOBJECTSP`. Now open this new project and code window of the `Insert Course Form`. Add the namespace `System.Data.OracleClient` to the namespace area of this code window.

Also remove the coding related to the `Faculty ID` textbox. Two methods contained the coding related to that textbox: `InitCourseInfo()` and `Cancel` button's `Click` methods.

6.10.2.2.2 Develop Codes to Call Stored Procedures The main codes to call the stored procedure are developed inside the `Insert` button's `Click` method and the user-defined `InsertParameters()` method in the `Insert Course Form` window. The codes for these methods are very similar to the codes we did for the same method in the last project `SQLInsertRTOBJECTSP`. Some modifications are made for these two methods to make them work for the Oracle database. First, let's concentrate on the modifications for the `Insert` button's `Click` method. Perform the following modifications that have been highlighted in bold and are shown in Figure 6.84 for this method. Let's give a detailed discussion of these modifications one by one based on the steps defined in Figure 6.84.

- A. Change the content of the query string, which should be the name of the stored procedure, to the procedure's name that we defined when we created this stored procedure using the `Object Browser` page in Oracle Database 10g XE in Section 6.10.2.1.
- B. Change the prefix for all Oracle data classes from **Sql** to **Oracle** and all Oracle data objects from **sql** to **ora**.
- C. Change the prefix before all Oracle objects from **sql** to **ora**. Steps involved in this change are from **C** to **H**.

Now let's do the modifications to the user-defined `InsertParameters()` method. Perform the following modifications shown in Figure 6.85 to this method, and all modifications have been highlighted in bold and are shown in Figure 6.85.

The first modification is to change the data type of the passed object from `SqlCommand` to `OracleCommand` since we are using Oracle `Command` object. The second modification is to remove the `@` symbol before the name of each parameter, which is represented as


```

OracleInsertRTOBJECTSP.InsertCourseForm  cmdInsert_Click()
private void cmdInsert_Click(object sender, EventArgs e)
{
  A   int check = 0, intInsert = 0;
  B   string cmdString = "InsertFacultyCourse";
      OracleCommand oraCommand = new OracleCommand();
      LogInForm logForm = new LogInForm();
      InitCourseInfo();
      check = CheckCourseInfo();
      if (check == 0)
      {
        C   logForm = logForm.getLogInForm();
        D   oraCommand.Connection = logForm.oraConnection;
        E   oraCommand.CommandType = CommandType.StoredProcedure;
        F   oraCommand.CommandText = cmdString;
        G   InsertParameters(ref oraCommand);
        H   intInsert = oraCommand.ExecuteNonQuery();
          oraCommand.Dispose();
          if (intInsert == 0)
            MessageBox.Show("The data insertion is failed");
          else
          {
            cmdCancel.PerformClick();           // clean up all faculty information
            cmdInsert.Enabled = false;
          }
        }
      }
      else
        MessageBox.Show("Fill all Course Information box, enter a NULL for blank column");
}

```

Figure 6.84 Modifications to the coding of Insert button's Click method.

```

OracleInsertRTOBJECTSP.InsertCourseForm  InsertParameters()
private void InsertParameters(ref OracleCommand cmd)
{
  cmd.Parameters.Add("FacultyName", OracleType.Char).Value = ComboName.Text;
  cmd.Parameters.Add("CourseID", OracleType.Char).Value = txtCourseID.Text;
  cmd.Parameters.Add("Course", OracleType.Char).Value = txtCourse.Text;
  cmd.Parameters.Add("Schedule", OracleType.Char).Value = txtSchedule.Text;
  cmd.Parameters.Add("Classroom", OracleType.Char).Value = txtClassRoom.Text;
  cmd.Parameters.Add("Credit", OracleType.Char).Value = txtCredits.Text;
  cmd.Parameters.Add("Enroll", OracleType.Char).Value = txtEnroll.Text;
}

```

Figure 6.85 Modifications to the InsertParameters method.

a string and located at the first argument's position. Finally, change the data type for each parameter from SqlDbType to OracleType.

The function of this method is to assign each piece of information stored in each textbox to the associated input parameter we defined in the Oracle stored procedure InsertFacultyCourse. One key point of this coding is that the name of each parameter, which is represented as a string and located at the first argument's position, must be identical with each input parameter's name we defined in the stored procedure. For example, the name of the parameter FacultyName used here must be identical with the input parameter's name FacultyName that exists in the input parameter's list we defined

at the beginning of the stored procedure `InsertFacultyCourse`. A runtime error would be encountered if a name of a parameter is not matched with the associated parameter's name in the stored procedure as the project runs. Refer to Figure 6.81 to get a detailed list of all parameter names defined in the stored procedure.

Now we have finished coding for this data insertion operation. Before we can run the project to test the functionality of this data insertion, we need to figure out how to open and start the Insert Course Form to perform this new course insertion. There are two methods to open the Insert Course Form: one method is to directly start this form and insert data, and another method is to use the Course Form to trigger this form. We prefer to use the second method since we need to use the Course Form to perform the data validation after a new record is inserted into the Course table.

Let's make two modifications to the Course Form to trigger the Insert Course Form.

- Add a field-level object of the `InsertCourseForm` class, and this object is used to start the Insert Course Form window as the Insert button in the Course Form is clicked. To do that, add the following command under the class header:

```
InsertCourseForm InsertCourse = new InsertCourseForm();
```

- The second modification is to call the `Show()` method of the Insert Course Form object to display that form when the Insert button in the Course Form is clicked. To do that, open the Insert button's `Click` method and add the following command into this method in the Course Form, `cmdInsert_Click()`:

```
InsertCourse.Show()
```

Now let's run the project to test the new data insertion using the stored procedures. Click on the Start Debugging button to start the project, enter the suitable username and password, such as `jhenry` and test to the `LogIn` form, and select the Course Information item from the Selection Form to open the Course Form window. Click on the Insert button to open the Insert Course Form window to perform the new course insertion.

Enter the following data into the associated textbox as the information for a new course:

- Ying Bai Selected from the Faculty Name combobox
- CSE-669 Course ID textbox
- Modern Controls II Course Title textbox
- T-H: 9:30–10:45 AM Schedule textbox
- TC-213 Classroom textbox
- 3 Credits textbox
- 25 Enrollment textbox

Your finished information window should match the one shown in Figure 6.86.

Click on the Insert button to call the stored procedure to insert this new course record into the database. Immediately the Insert button is disabled after this data insertion. Is our data insertion successful? To answer this question, we need to perform the data validation in the next section.

6.10.2.2.3 Validate Data After Data Insertion Now click on the Back button to return to the Course Form window to validate this data insertion.

Figure 6.86 Running status of the Insert Course Form window.

Figure 6.87 Data validation process.

Click on the drop-down arrow of the combobox control `ComboName` and select the Ying Bai from the list since we inserted a new course for this faculty in the last section. Click on the `Select` button to try to retrieve back all courses, including the new inserted course record, from the database and display them in the `Course ListBox` in this `Course Form` window. All courses taught by the selected faculty are displayed in the `CourseList` box. The course next to the last course in the `Course ListBox` is just the course we added into the `Course` table in the last section. Click on that item and all information related to that new course is displayed in this form, which is shown in Figure 6.87. This is the evidence that our data insertion using the stored procedure is successful!

A completed project OracleInsertRTObjectSP that includes the data insertion using the stored procedure can be found at the folder DBProjects\Chapter 6 located at the accompanying ftp site (see Chapter 1).

6.11 CHAPTER SUMMARY

Five popular data insertion methods are discussed and analyzed with three different databases—Microsoft Access 2007, SQL Server, and Oracle:

1. Using TableAdapter's DBDirect methods TableAdapter.Insert() method
2. Using the TableAdapter's Update() method to insert new records that have already been added into the DataTable in the DataSet
3. Using the Command object's ExecuteNonQuery() method
4. Using LINQ to SQL query method
5. Using stored procedures method

Method 1 is developed using the Visual Studio.NET design tools and wizards, and it allows users to directly access the database and execute the TableAdapter's methods such as TableAdapter.Insert() and TableAdapter.Update() to manipulate data in the database without requiring DataSet or DataTable objects to reconcile changes in order to send updates to a database. As we mentioned at the beginning of this chapter, inserting data into a table in the DataSet is different with inserting data into a table in the database. If you are using the DataSet to store data in your applications, you need to use the TableAdapter.Update() method since the Update() method can trigger and send all changes (updates, inserts, and deletes) to the database.

A good habit to develop is to try and use the TableAdapter.Insert() method when your application uses objects to store data (e.g., you are using textboxes to store your data), or when you want finer control over creating new records in the database.

Method 2 allows users to insert new data into a database with two steps. First, the new record should be added into the data table located in the DataSet, and second the TableAdapter.Update() method can be executed to update the whole table in the DataSet to the associated table in the database.

Method 3 is a runtime objects method, and this method is more flexible and convenient, and it allows users to insert data into multiple data tables with the different functionalities.

Method 4 is a new technique coming with .NET Framework 3.5, Visual Studio.NET 2008, and LINQ that were released by Microsoft in 2008.

Method 5 uses stored procedures to replace the general query functions, and this method promises users more powerful controllability and flexibility on data insertions, especially for data insertions with multiple queries to multiple tables.

This chapter is divided into two parts. Part I provides a detailed discussion and analysis of inserting data into three different databases using the Visual Studio.NET design tools and wizards. It is simple and easy to develop a data insertion project with these tools and wizards. The disadvantage of using these tools and wizards is that the data can only be inserted to limited destinations, for example, a certain data table. Part II presents the runtime objects method, LINQ to SQL, and stored procedure methods to improve the efficiency of the data insertion and provides more flexibility in data insertion.

Eight real projects are provided in this chapter to give readers a clear and direct picture in developing professional data insertion applications in Visual C#.NET environment.

HOMWORK

I. True/False Selections

- ___ 1. Three popular data insertion methods are the TableAdapter.Insert(), TableAdapter.Update(), and ExecuteNonQuery() methods of the Command class.
- ___ 2. Unlike the Fill() method, a valid database connection must be set before a new data can be inserted in the database.
- ___ 3. One can directly insert new data or new records into the database using the TableAdapter.Update() method.
- ___ 4. When executing an INSERT query, the order of the input parameters in the VALUES list can be different with the order of the data columns in the database.
- ___ 5. To insert data into the Oracle database using stored procedures, an Oracle Package must be developed to include stored procedures.
- ___ 6. To insert a new record into the SQL Server database using the LINQ to SQL query, the insertion cannot be completed until the SubmitChanges() method is executed.
- ___ 7. When performing the data insertion, the same data can be inserted into the database multiple times.
- ___ 8. To insert data into the database using the TableAdapter.Update() method, the new data should be first inserted into the table in the DataSet, and then the Update() method is executed to update that new data into the table in the database.
- ___ 9. To insert data into the SQL Server database using the stored procedures, one can create and test the new stored procedure in the Server Explorer window.
- ___ 10. To call stored procedures to insert data into a database, the parameters' names must be identical with those names of the input parameters defined in the stored procedures.

II. Multiple Choices

- 1. To insert data into the database using the TableAdapter.Insert() method, one needs to use the _____ to build the _____.
 - a. Data Source, Query Builder
 - b. TableAdapter Query Configuration Wizard, Insert query
 - c. Runtime object, Insert query
 - d. Server Explorer, Data Source
- 2. To insert data into the database using the TableAdapter.Update() method, one needs first to add new data into the _____, and then update that data into the database.
 - a. Data table
 - b. Data table in the database
 - c. DataSet
 - d. Data table in the DataSet
- 3. To insert data into the database using the TableAdapter.Update() method, one can update _____.
 - a. One data row only
 - b. Multiple data rows

- c. The whole data table
 - d. Any of the above
4. Because the ADO.NET provides a disconnected mode to the database, to insert a new record into the database, a valid _____ must be established.
- a. DataSet
 - b. TableAdapter
 - c. Connection
 - d. Command
5. The _____ operator should be used as an assignment operator for the WHERE clause with a dynamic parameter for a data query in Oracle database.
- a. =:
 - b. LIKE
 - c. =
 - d. @
6. To confirm the stored procedure built in the Object Browser page in the Oracle database, one can _____ the stored procedure to make sure it works.
- a. Build
 - b. Test
 - c. Debug
 - d. Compile
7. To confirm the stored procedure built in the Server Explorer window for the SQL Server database, one can _____ the stored procedure to make sure it works.
- a. Build
 - b. Execute
 - c. Debug
 - d. Compile
8. To insert data into an Oracle database using the INSERT query, the parameters' data type must be _____.
- a. OleDbType
 - b. SqlDbType
 - c. OracleDbType
 - d. OracleType
9. To insert data using stored procedures, the CommandType property of the Command object must be equal to _____.
- a. CommandType.InsertCommand
 - b. CommandType.StoredProcedure
 - c. CommandType.Text
 - d. CommandType.Insert
10. To insert data using stored procedures, the CommandText property of the Command object must be equal to _____.
- a. The content of the CommandType.InsetCommand
 - b. The content of the CommandType.Text
 - c. The name of the Insert command
 - d. The name of the stored procedure

III. Exercises

1. Figure 6.88 shows a stored procedure developed in the SQL Server database. Please develop a piece of code in Visual C#.NET to call this stored procedure to insert a new data into the database.
2. Figure 6.89 shows a piece of code developed in Visual C#.NET, and this coding is used to call a stored procedure in the Oracle database to insert a new record into the database. Please create the associated stored procedure in the Oracle database using the PL- SQL language (assume a valid database connection has been set).
3. Using the tools and wizards provided by Visual Studio.NET and ADO.NET to perform the data insertion for the Student Form in the InsertWizard project (the project file is located at the folder DBProjects\Chapter 6 located at the accompanying ftp site (see Chapter 1).
4. Using the runtime objects to complete the insert data query for the Student Form by using the project AccessInsertRTOObject (the project file is located at the folder DBProjects\Chapter 6 located at the accompanying ftp site (see Chapter 1).
5. Using the stored procedure to complete the insert data query for the Student Form to the Student table by using the project OracleInsertRTOObjectSP (the project file is located at the folder DBProjects\Chapter 6 located at the accompanying ftp site (see Chapter 1).

```

CREATE OR REPLACE PROCEDURE dbo.InsertStudent
(@Name IN VARCHAR(20),
@Major IN text,
@SchoolYear IN int,
@Credits IN float,
@Email IN text)
AS
INSERT INTO Student VALUES (@Name, @Major, @SchoolYear, @Credits, @Email)
RETURN

```

Figure 6.88

```

string cmdString = "InsertCourse";
int intInsert = 0;
OracleCommand oraCommand = new OracleCommand();
oraCommand.Connection = oraConnection;
oraCommand.CommandType = CommandType.StoredProcedure;
oraCommand.CommandText = cmdString;
oraCommand.Parameters.Add("Name", OracleType.Char).Value = ComboName.Text;
oraCommand.Parameters.Add("CourseID", OracleType.Char).Value = txtCourseID.Text;
oraCommand.Parameters.Add("Course", OracleType.Char).Value = txtCourse.Text;
oraCommand.Parameters.Add("Schedule", OracleType.Char).Value = txtSchedule.Text;
oraCommand.Parameters.Add("Classroom", OracleType.Char).Value = txtClassRoom.Text;
oraCommand.Parameters.Add("Credit", OracleType.Char).Value = txtCredits.Text;
intInsert = oraCommand.ExecuteNonQuery();

```

Figure 6.89

Chapter 7

Data Updating and Deleting with Visual C#.NET

In this chapter, we will discuss how to update and delete data in the databases. Generally, many different methods are provided and supported by Visual C#.NET and .NET Framework to help users perform data updating and deleting in the database. Among them, three popular methods are widely implemented:

1. Use TableAdapter DBDirect methods such as TableAdapter.Update() and TableAdapter.Delete() to update and delete data directly in the databases.
2. Use the TableAdapter.Update() method to update and execute the associated TableAdapter's properties such as UpdateCommand or DeleteCommand to save changes made to the table in the DataSet to the table in the database.
3. Use the runtime objects method to develop and execute the Command's method ExecuteNonQuery() to update or delete data in the database directly.

Both methods 1 and 2 need to use Visual Studio.NET Design Tools and Wizards to create and configure suitable TableAdapters, build the associated queries using the Query Builder, and call those queries from Visual C#.NET applications. The difference between method 1 and 2 is that method 1 can be used to directly access the database to perform data updating and deleting in a single step. Method 2, however, needs two steps to finish the data updating or deleting. First, the data updating or deleting is performed on the associated tables in the DataSet, and then those updated or deleted data are updated to the tables in the database by executing the TableAdapter.Update() method.

This chapter is divided into two parts: Part I provides discussions on data updating and deleting using methods 1 and 2, or in other words, using the TableAdapter.Update() and TableAdapter.Delete() methods developed in Visual Studio.NET Design Tools and Wizards. Part II presents data updating and deleting using the runtime objects method by developing command objects to execute the ExecuteNonQuery() method dynamically. Updating and deleting data using the LINQ to SQL query is also discussed in that part.

When finished with this chapter, you will:

- Understand the working principle and structure of updating and deleting data in the database using the Visual Studio.NET Design Tools and Wizards.
- Understand the procedures of how to configure the TableAdapter object by using the TableAdapter Query Configuration Wizard and build the query to update and delete data in the database.
- Design and develop special procedures to validate data before and after data updating and deleting.
- Understand the working principle and structure of updating and deleting data in the database using the runtime objects method.
- Design and build LINQ to SQL query to update and delete data.
- Design and build stored procedures to perform data updating and deleting.

To successfully complete this chapter, you need to understand topics such as fundamentals of databases, introduced in Chapter 2, and ADO.NET, discussed in Chapter 3. Also a sample database CSE_DEPT that was developed in Chapter 2 will be used throughout this chapter.

Three kinds of databases will be used in the example projects to illustrate how to perform data updating and deleting in this chapter. They are Microsoft Access 2007, SQL Server 2005 SP2, and Oracle Database 10g XE Release 2.

In order to save time and avoid repeatability, we will use the sample projects such as InsertWizard, SQLInsertWizard, AccessInsertRTOObject, SQLInsertRTOObject, and OracleInsertRTOObject we developed in the last chapter, modify them to create new associated projects, and use them in this chapter. Recall that some command buttons on the different form windows in those projects have not been coded, such as Update and Delete, and those buttons, or those methods related to those buttons, will be developed and built in this chapter. We only concentrate on the coding for the Update and Delete buttons in this chapter.

PART I DATA UPDATING AND DELETING WITH VISUAL STUDIO.NET DESIGN TOOLS AND WIZARDS

In this part, we discuss updating and deleting data in the database using the Visual Studio.NET Design Tools and Wizards. We will develop two methods to perform these data actions: First, we use the TableAdapter DBDirect methods, TableAdapter.Update() and TableAdapter.Delete(), to directly update or delete data in the database. Second, we show readers how to update or delete data in the database by first updating or deleting records in the DataSet, and then updating those records' changes from the DataSet to the database using the TableAdapter.Update() method. Both methods utilize the so-called TableAdapter's direct and indirect methods to complete data updating or deleting. The database we use is the Microsoft Access 2007 database, CSE_DEPT.accdb, which was developed in Chapter 2, and it is located at the folder Database\Access located at the ftp://ftp.wiley.com/public/sci_tech_med/practical_database site. You can use any other databases such as Microsoft SQL Server 2005 SP2 or Oracle Database 10g XE. The only caution is that you need to select and connect the correct database with your applications when you use the Data Source window to set up your data source for your Visual C#.NET data-driven applications.

7.1 UPDATE OR DELETE DATA IN DATABASES

We have already provided a very detailed discussion about the `TableAdapter` `DBDirect` methods in Section 6.1.1. To use these methods to directly access the database to make the desired manipulations to the data stored in the database, one needs to use Visual Studio.NET Design Tools and Wizards to create and configure the associated `TableAdapter`. Some limitations exist when using these `DBDirect` methods to update or delete data in databases. For example, each `TableAdapter` is associated with a unique data table in the `DataSet`; therefore, data updating or deleting can only be executed for that unique data table by using the associated `TableAdapter`. In other words, the specified `TableAdapter` cannot update or delete data from any other data tables except the data table related to the created `TableAdapter`.

7.1.1 Updating and Deleting Data in Related Tables in DataSet

When updating or deleting data in related tables in a `DataSet`, it is important to update or delete data in the proper sequence in order to reduce the chance of violating referential integrity constraints. The order of command execution will also follow the indices of the `DataRowCollection` in the `DataSet`. To prevent data integrity errors from being raised, the best practice is to update or delete data in the database in the following sequence:

1. Child table: Delete records.
2. Parent table: Insert, update, and delete records.
3. Child table: Insert and update records.

For our sample database `CSE_DEPT`, all five tables are related with different primary keys and foreign keys. For example, among the `LogIn`, `Faculty`, and `Course` tables, the `faculty_id` works as a key to relate these three tables together. The `faculty_id` is a primary key in the `Faculty` table, but a foreign key in both the `LogIn` and the `Course` tables. In order to update or delete data from any of those tables, one needs to follow the sequence above. For example, when updating or deleting a record in the database, the following data operation sequence needs to be performed:

1. First, remove or delete that record from the child tables, `LogIn` and `Course` tables.
2. Then update or delete that record in the parent table, `Faculty` table.
3. Finally that updated record can be inserted into the child tables such as `LogIn` and `Course` tables for the data updating operation. There is no data actions for the data-deleting operations for the child tables.

It would be terribly complicated if we update a full record (including updating the primary key) for an existing record in our sample database. In practice, it is unnecessary to update a primary key for any record since the primary key has the same lifetime as a database. A better and more popular way to do this updating is to remove those undesired records and then insert new records with new primary keys. Therefore, in this chapter, we will concentrate on updating existing data in our sample database without touching the primary key. For data deleting, we can delete a full record with the primary key involved, and all related records in the child tables will also be deleted since all tables

Table 7.1 TableAdapter DBDirect Methods

TableAdapter DBDirect Method	Description
TableAdapter.Insert	Adds new records into a database allowing you to pass in individual column values as method parameters.
TableAdapter.Update	Updates existing records in a database. The Update method takes original and new column values as method parameters. The original values are used to locate the original record, and the new values are used to update that record. The TableAdapter.Update method is also used to reconcile changes in a data set back to the database by taking a DataSet, DataTable, DataRow, or array of DataRows as method parameters.
TableAdapter.Delete	Deletes existing records from the database based on the original column values passed in as method parameters.

have been set in a Cascade Delete mode when we built these data tables for our sample database CSE_DEPT in Section 2.9.4 in Chapter 2.

7.1.2 Update or Delete Data in Database Using TableAdapter DBDirect Methods—TableAdapter.Update and TableAdapter.Delete

Three typical TableAdapter's DBDirect methods are listed in Table 6.1. For your convenience, we repeat that table in this section as Table 7.1.

Both DBDirect methods, TableAdapter.Update() and TableAdapter.Delete(), need the original column values as the parameters when these methods are executed. The TableAdapter.Update() method needs both the original and the new column values to perform the data updating. Another point to note is that when the application uses the object to store the data, for instance, in our sample project we use textbox objects to store our data, you should use this DBDirect method to perform the data manipulations in the database.

7.1.3 Update or Delete Data in Database Using TableAdapter.Update Method

You can use the TableAdapter.Update() method to update or edit records in a database. The TableAdapter.Update() method provides several overloads that perform different operations depending on the parameters passed in. It is important to understand the results of calling these different method signatures.

To use this method to update or delete data in the database, one needs to perform the following two steps:

Table 7.2 Variations of `TableAdapter.Update()` Method

Update Method	Description
<code>TableAdapter.Update(DataTable)</code>	Attempts to save all changes in the <code>DataTable</code> to the database. (This includes removing any rows deleted from the table, adding rows inserted to the table, and updating any rows in the table that have changed.)
<code>TableAdapter.Update(DataSet)</code>	Although the parameter takes a data set, the <code>TableAdapter</code> attempts to save all changes in the <code>TableAdapter</code> 's associated <code>DataTable</code> to the database. (This includes removing any rows deleted from the table, adding rows inserted in the table, and updating any rows in the table that have changed.)
<code>TableAdapter.Update(DataRow)</code>	Attempts to save changes in the indicated <code>DataRow</code> to the database.
<code>TableAdapter.Update(DataRows())</code>	Attempts to save changes in any row in the array of <code>DataRows</code> to the database.
<code>TableAdapter.Update("new column values", "original column values")</code>	Attempts to save changes in a single row that is identified by the original column values.

1. Change or delete records from the desired `DataTable` based on the selected data rows from the table in the `DataSet`.
2. After the rows have been modified or deleted from the `DataTable`, call the `TableAdapter.Update()` method to reflect those modifications to the database. You can control the amount of data to be updated by passing an entire `DataSet`, a `DataTable`, an array of `DataRows`, or a single `DataRow`.

Table 7.2 describes the behavior of the various `TableAdapter.Update()` methods.

Different parameters or arguments can be passed into these five variations of this method. The parameter `DataTable`, which is located in a `DataSet`, is a data table mapping to a real data table in the database. When a whole `DataTable` is passed, any modification to that table will be updated and reflected in the associated table in the database. Similarly, if a `DataSet` is passed, all `DataTables` in that `DataSet` will be updated and reflected to those tables in the database.

The last variation of this method is to pass the original columns and the new columns of a data table to perform this updating. In fact, this method can be used as a `DBDirect` method to access the database to manipulate data.

In order to provide a detailed discussion and explanation of how to use these two methods to update or delete records in the database, a real example will be very helpful. Let's first create a new Visual C#.NET project to handle these issues.

7.2 UPDATE AND DELETE DATA FOR MICROSOFT ACCESS DATABASE

We have provided a very detailed introduction about the Design Tools and Wizards in Visual Studio.NET in Section 5.2 in Chapter 5. The popular Design Tools and Wizards

include the DataSet, BindingSource, TableAdapter, Data Source window, Data Source Configuration window, and DataSet Designer. We need to use these to develop our data-updating and deleting sample project based on the InsertWizard project we developed in the last chapter. First, let's copy that project and do some modifications on that project to get our new project. The advantage of creating our new project in this way is that you don't need to redo the data source connection and configuration since those operations have been performed in the previous chapter.

7.2.1 Create New Project Based on InsertWizard Project

Open Windows Explorer and create a new folder such as Chapter 7, and then browse to our project InsertWizard that can be found at the folder DBProjects\Chapter 6 located at the accompanying ftp site (See Chapter 1). Copy this project to our new folder Chapter 7. Change the name of the solution and the project from InsertWizard to AccessUpdateDeleteWizard. Double-click on the AccessUpdateDeleteWizard Project.csproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to the Project\AccessUpdateDeleteWizard Project Properties menu item to open the project's property window. Change the Assembly name from InsertWizard Project to AccessUpdateDeleteWizard Project and the Default namespace from InsertWizard Project to AccessUpdateDeleteWizard Project.
- Click on the Assembly Information button to open the Assembly Information dialog box, change the Title and the Product to AccessUpdateDeleteWizard Project. Click on OK to close this dialog box.
- Change the project namespace for all files in our current new project from InsertWizard Project to AccessUpdateDeleteWizard Project. To do this, perform the following operations:
 - Go to the Edit\Quick Find menu item to open the Find and Replace dialog box.
 - Enter InsertWizard Project into the Find what box.
 - Select Entire Solution from the Look in box.
 - Then click on the Find Next button to find each old project namespace InsertWizard Project, and replace it with our new namespace AccessUpdateDeleteWizard Project.

Go to the File\Save All to save those modifications. Now we are ready to develop our graphic user interfaces for our new project AccessUpdateDeleteWizard Project.

7.2.2 Application User Interfaces

Recall that when we developed the project InsertWizards, there were five command buttons located in the Faculty Form window: Select, Insert, Update, Delete, and Back. In this section, we need to use both Update and Delete buttons, that is, these two buttons' Click methods, to perform the data updating and deleting actions in the database. Unlike adding a new record into the database, for the update and delete operations, we don't need to develop any new form windows as the user interfaces to collect the new records to perform those updating and deleting operations. Instead we can use the Faculty Form window to do these data updating and deleting actions with some modifications.

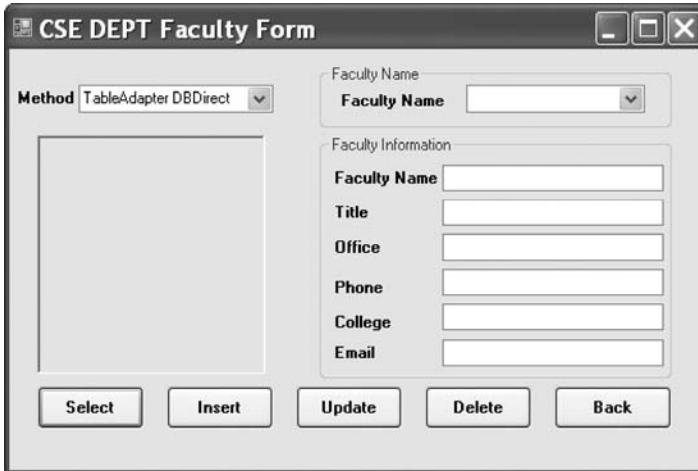


Figure 7.1 Modified Faculty Form window.

7.2.2.1 Modify Faculty Form Window

Since we have two methods to perform the data updating or deleting, we need to add one more control to the Faculty Form window to enable users to select each of these two methods as the project runs. Add a combobox control named `ComboMethod` to the upper-left corner of the Faculty Form window, which is shown in Figure 7.1.

Also we need to change the five label controls located inside the Faculty Information GroupBox to five textbox controls because we need to change the faculty information by entering new data into those textboxes. Besides these five textbox controls, add one more textbox into this GroupBox and name it as `txtName` for the `faculty_name` column in the Faculty table. This new textbox is used to hold the updating Faculty Name information. The modified Faculty Form window is shown in Figure 7.1.

Another advantage of using this Faculty Form as our user interface to perform the data updating and deleting is that we don't even need to use a searching process to find the data items to be updated or deleted from the database. Instead, we can first get the data to be updated or deleted by performing a data query using the Select button's Click method. Then we can easily update or delete that data by modifying any piece of information that is related to that data and stored in the associated textbox, in the Faculty Form window.

7.2.2.2 Bind Data for All Textboxes of Faculty Form Window

This data binding is necessary since we need to call the Select button's Click method to execute the data query when we perform the data validations for data inserting, updating, and deleting later. Open the Faculty Form window if it is not opened, and then select the first textbox Faculty ID, which we just added. Go to the Properties window and expand the (DataBindings) item to the Text property, and then click on the drop-down arrow that is next to the Text property to open the Add Project Data Source dialog box. Select the `faculty_name` from the list by clicking on it, which is shown in Figure 7.2a. In this

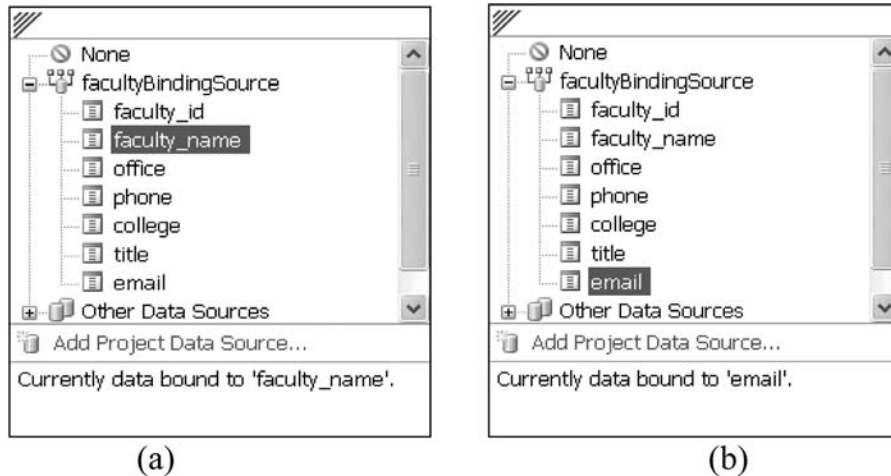


Figure 7.2 Data-binding process.

way, a binding relationship between the Faculty Name textbox in the Faculty Form and the `faculty_name` column in the DataSet is set up.

In a similar way, you can finish the data bindings for all other five textboxes. An example of binding for the Email textbox is shown in Figure 7.2b.

7.2.3 Validate Data Before Data Updating and Deleting

This data validation can also be neglected because when we performed a data query by clicking on the Select button, the retrieved data should be complete data and can be displayed in the Faculty Form window. This means that all textboxes have been filled by the related faculty information and none are empty. No matter if we do some modifications or not, all textboxes should be full. So this data validation before the data updating or deleting can be avoided.

7.2.4 Build Update and Delete Queries

As we mentioned, two methods will be discussed in this part: One is to update or delete records using the TableAdapter DBDirect method, and the other one is to use the TableAdapter.Update() method to update modified records from the DataSet into the database. First, let's concentrate on the first method.

Let's start this data updating and deleting with building our data updating and deleting queries using the TableAdapter Query Configuration Wizard and Query Builder.

7.2.4.1 Build Data Updating Query Function

Open the Data Source window by going to the `Data>Show Data Sources` menu item. On the opened window, click on the `Edit the DataSet with Designer` button that is

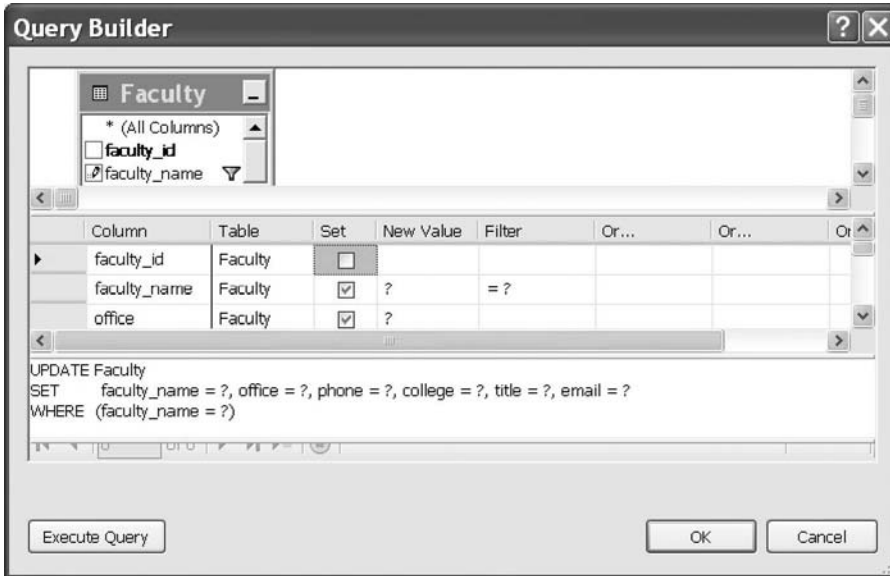


Figure 7.3 Query Builder for the Update query.

located second from the left on the toolbar in the Data Source window to open this Designer. Then right-click on the bottom item from the Faculty table and select the Add Query item from the pop-up menu to open the TableAdapter Query Configuration Wizard. Keep the default selection: Use SQL statements unchanged and click on Next to go to the next window. Select and check the UPDATE item from this window since we need to perform a data updating query, and then click on Next again to continue. Click on the Query Builder button since we want to build our updating query. The opened Query Builder window is shown in Figure 7.3.

Uncheck the checkbox located at the Set column and the faculty_id row to deselect this item from this data updating action because we will update all pieces of faculty information except the faculty_id in this chapter. Remove the default question mark (=?) from the faculty_id row under the column Filter, and enter a question mark (?) to the faculty_name row under the column Filter, and press the Enter key on the keyboard. Remove all rows under the last row—email—from the midpane, and your finished query builder window should match the one shown in Figure 7.3.

Click on the OK button to go to the next window. Click on Next to confirm this query and continue to the next step. Modify the query function name from the default one to the UpdateFaculty and click on Next to go to the last window. Click on the Finish button to complete this query building and close the wizard. Immediately you can find that a new query function has been added into the Faculty TableAdapter as the last item.

Next let's build our Delete query function using the Query Builder.

7.2.4.2 Build Data Deleting Query Function

Reopen the Edit DataSet with Designer window and right-click on the last item from the Faculty table and select the Add Query item to open the TableAdapter Query

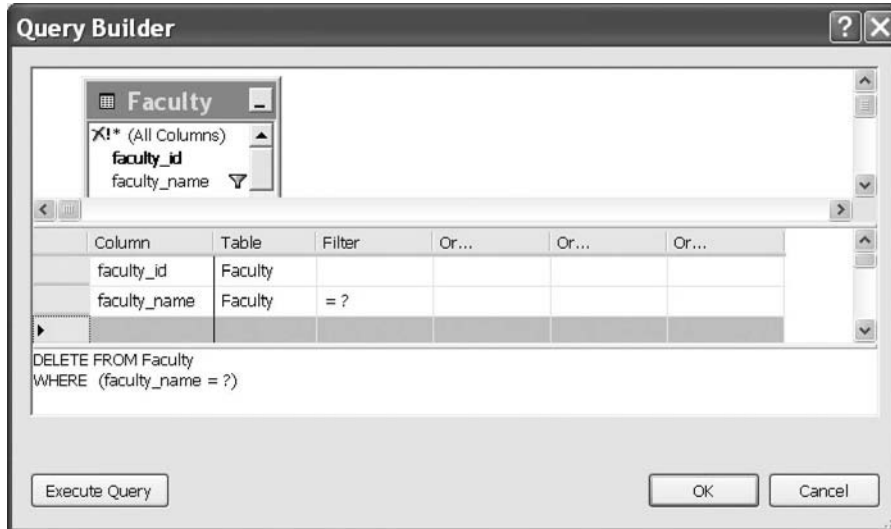


Figure 7.4 Query Builder for the Delete query.

Configuration Wizard if it is not opened. On the opened wizard, keep the default selection: Use SQL statements unchanged and click on Next to go to the next window. Select and check the DELETE item from this window since we need to perform data deleting query, and then click on Next again to continue. Click on the Query Builder button since we want to build our deleting query. The opened Query Builder window is shown in Figure 7.4.

Remove the question mark (=?) from the `faculty_id` row under the column Filter. Go to the middle pane, and then type `faculty_name` into the second row that is just under the row of `faculty_id` in the first column. Move the cursor to the Filter column along the `faculty_name` row and enter a question mark (?) into that field, press the Enter key on the keyboard. Then, delete all rows under the `faculty_name` row from the midpane. Your finished query builder should match the one shown in Figure 7.4.

Click on the OK button to go to the next window. Click on Next to confirm this query and continue to the next step. Modify the query function name from the default one to the `DeleteFaculty` and click on Next to go to the last window. Click on the Finish button to complete this query building and close the wizard. Immediately you can find that a new query function has been added into the Faculty TableAdapter as the last item.

7.2.5 Develop Codes to Update Data Using TableAdapter DBDirect Method

To perform the data updating using this method, some modifications to the original coding are necessary. We divided the coding operation into two subsections: modification coding and updating coding.

7.2.5.1 Modifications of Coding

The first modification is to modify the codes inside the `FacultyForm_Load()` method in the `FacultyForm` class, that is, change the query methods in the combobox `ComboMethod` to two new methods:

1. `TableAdapter.DBDirect`
2. `TableAdapter.Update`

To do this, open this method and change the two methods' names in the `Add()` method.

The second modification is to remove the `if` block and its codes for the `LINQ to DataSet` method in the `Select` button's `Click` method since we will not use this method in these data updating and deleting actions. Open that method and delete the `if` block and its codes. Also remove the `else` statement and its opening-ending braces, which is a pair with the removed `if` block.

The third modification is to remove the user-defined `LINQtoDataSet()` method since we do not need this method for our data updating and deleting actions. Open the code window of the `Faculty Form` window and delete that method from this window.

Now we can try to build our modified project by selecting the menu item `Build|Build AccessUpdateDeleteWizard Project`. You may encounter some compiling errors, but do not worry about them. The reason for those compiling errors is the mismatched namespace between our new project `AccessUpdateDeleteWizard_Project` and the old project `InsertWizard_Project` that was modified to become to our new project. To correct those errors, find and correct those errors by:

1. Go to the `Edit|Quick Find` menu item to open the `Find and Replace` dialog box.
2. Enter `InsertWizard_Project` into the `Find what` box.
3. Select `Entire Solution` from the `Look in` combobox.
4. Click on the `Find Next` button to find those errors one by one.
5. Replace those error namespace `SampleWizards_Project` with our current correct namespace `AccessUpdateDeleteWizard_Project`.

Most or all of those mismatched namespace errors may occur in our `DataSet Design` file `CSE_DEPTDataSet.Designer.cs` file.

7.2.5.2 Updating Coding

The main codes to perform this data updating are developed inside the `Update` button's `Click` method. Open the project `AccessUpdateDeleteWizard` and the `Faculty Form` window, then double-click on the `Update` button to open its `Click` method. Enter the codes shown in Figure 7.5 into this method.

Let's take a look at this piece of new added code to see how it works.

- A. All objects and variables used for this data updating action are declared here. A new row object of `FacultyRow` class is created here since we need this object to update the data in the `DataSet` to the database later when we use the second method, `TableAdapter.Update()`, to perform the data updating. The local integer variable `intUpdate` is used to hold the returned value of calling the `TableAdapter.DBDirect` method to update the data, and the

```

AccessUpdateDeleteWizard_Project.FacultyF  cmdUpdate_Click()
private void cmdUpdate_Click(object sender, EventArgs e)
{
  A   CSE_DEPTDataSet.FacultyRow FacultyRow;
      int intUpdate = 0;
      string strFacultyID;
  B   if (ComboMethod.Text == "TableAdapter DBDirect")
          intUpdate = facultyTableAdapter.UpdateFaculty(txtName.Text, txtOffice.Text, txtPhone.Text,
              txtCollege.Text, txtTitle.Text, txtEmail.Text, ComboName.Text);
  C   else // TableAdapter Update method selected
  D   if (intUpdate == 0)
          MessageBox.Show("Faculty Table Updating is failed!");
}

```

Figure 7.5 Coding for the Update command button's Click method.

local String variable `strFacultyID` is used to hold the returned `faculty_id` value when executing the second method, `TableAdapter.Update()`, to update the data in the next step.

- B.** If the user selected the first method, `TableAdapter DBDirect` method, to perform this data updating, the updating function we built in Section 7.2.4.1 is called to update the selected faculty. This function will return a value to indicate whether this function calling is successful or not. The value returned is equal to the number of rows or records that have been updated in the database successfully.
- C.** If the user selected the second method, `TableAdapter.Update()`, to update this data, the related codes that will be developed later are executed to first update data in the `DataSet`, and then update data to the database.
- D.** If the returned value of executing the updating function is equal to 0, which means that no row or 0 record has been updated after calling that query function, a warning message is displayed.

Now let's continue to develop the codes for the second data updating method.

7.2.6 Develop Codes to Update Data Using `TableAdapter.Update` Method

Open the Update button's Click method if it is not opened and add the codes shown in Figure 7.6 into this method, that is, into the `else` block. The codes we developed in the previous section have been highlighted by shading.

Let's take a look at this piece of new added code to see how it works.

- A.** In order to update a selected row from the Faculty table in the `DataSet`, we need first to identify that row. Visual Studio.NET provides a default method, `FindBy()`, to do that. But that method needs a primary key as a criterion to perform a query to locate the desired row from the table. In our case, the primary key for our Faculty table is the `faculty_id`. To find this `faculty_id`, we can use the query function `FindFacultyIDByName()` we built in Section 5.14 in Chapter 5 with the Faculty Name as a criterion. One key point to run this function is that the parameter Faculty Name must be an old faculty name because in order to update a faculty, we must first find the old faculty row based on the old `faculty_name`. Therefore the combobox `ComboName.Text` is used as the old faculty name.

```

AccessUpdateDeleteWizard_Project.FacultyFo  cmdUpdate_Click()
private void cmdUpdate_Click(object sender, EventArgs e)
{
    CSE_DEPTDataSet.FacultyRow FacultyRow;
    int intUpdate = 0;
    string strFacultyID;

    if (ComboMethod.Text == "TableAdapter DBDirect")
        intUpdate = facultyTableAdapter.UpdateFaculty(txtName.Text, txtOffice.Text, txtPhone.Text,
            txtCollege.Text, txtTitle.Text, txtEmail.Text, ComboName.Text);
    else // TableAdapter Update method selected
    {
        strFacultyID = facultyTableAdapter.FindFacultyIDByName(ComboName.Text);
        FacultyRow = cSE_DEPTDataSet.Faculty.FindByfaculty_id(strFacultyID);
        FacultyRow = UPFacultyRow(ref FacultyRow);
        this.Validate();
        facultyBindingSource.EndEdit();
        intUpdate = facultyTableAdapter.Update(cSE_DEPTDataSet.Faculty);
    }
    if (intUpdate == 0)
        MessageBox.Show("Faculty Table Updating is failed!");
}

```

A
B
C
D
E
F

Figure 7.6 Coding for the second data updating method.

- B.** After the `faculty_id` is found, the default system method `FindByfaculty_id()` is executed to locate the desired row from the Faculty table, and the desired data row is returned and assigned to the local variable `FacultyRow`.
- C.** A user-defined `UPFacultyRow()` method is called to assign all updated faculty information to the selected row. In this way, the faculty information, that is, a row in the Faculty table in the DataSet, is updated.
- D.** The `Validate()` command closes out the editing of a control; in our case, it closes any editing for textbox control in the Faculty form.
- E.** The `EndEdit()` method of the binding source writes any edited data in the controls back to the record in the DataSet. In our case, any updated data entered into the textbox control will be reflected to the associated column in the DataSet.
- F.** Finally the `Update()` method of the TableAdapter sends updated data back to the database. The argument of this method can be a whole DataSet, a DataTable in the DataSet, or a DataRow in a DataTable. In our case, we used the Faculty DataTable as the argument for this method.

The detailed codes for the user-defined `UPFacultyRow()` method are shown in Figure 7.7. The function of this method is straightforward and easy to understand.

The argument of this method is a `DataRow` object, and it is passed by a reference to the method. The advantage of passing an argument in this way is that any modifications performed to `DataRow` object inside the method can be returned to the calling procedure without needing another returned variable to be created. The updated faculty information stored in the associated textbox is assigned to the associated column of the `DataRow` in the Faculty table in the DataSet. In this way, the selected `DataRow` in the Faculty table is updated.

At this point, we finished the coding development for two methods to update the data in a database. Next we discuss how to delete data from the database.

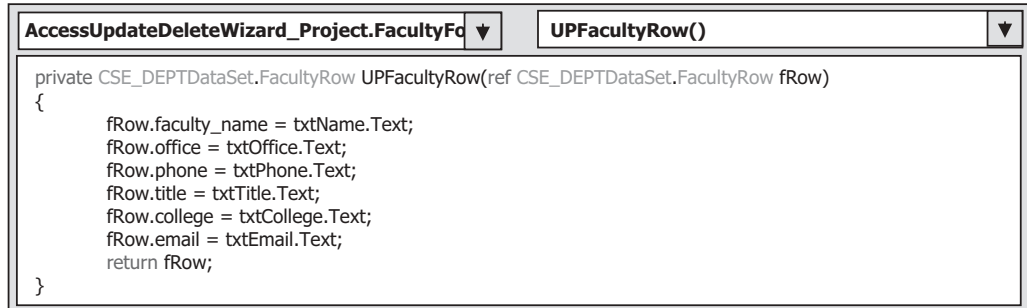


Figure 7.7 Coding for the user-defined UPFacultyRow method.

7.2.7 Develop Codes to Delete Data Using TableAdapter DBDirect Method

To delete data from a database, you can use either the TableAdapter DBDirect method `TableAdapter.Delete()` or the `TableAdapter.Update()` method. Or, if your application does not use TableAdapters, you can use the runtime objects method to create command objects to delete data from a database (e.g., `ExecuteNonQuery()`).

The `TableAdapter.Update()` method is typically used when your application uses DataSets to store data, whereas the `TableAdapter.Delete()` method is typically used when your application uses objects, for example, in our case we used textboxes, to store data.

As mentioned in the previous section, to delete a record from our sample database, no new user interface is needed, and we can use the Faculty Form window to perform this data deleting action. Open the Faculty Form window and double-click on the Delete button to open its Click method. Enter the codes shown in Figure 7.8 into this method.

Let's take a look at this piece of code to see how it works.

- A. All data components and objects as well as variables used in this method are declared and created here. A button object of the `MessageBoxButtons` class is created, and we need to use these two buttons to confirm the data deleting later. The `FacultyRow` is used to locate the `DataRow` in the `Faculty DataTable`, and it is used for the second deleting method. The local variable `Answer` is an instance of `DialogResult` and it is used to hold the returned value of calling the `MessageBox` function, and this variable can be replaced by an integer variable if you like.
- B. First, a `MessageBox` is called to confirm that a data deleting will be performed from the `Faculty data table`.
- C. If the returned value of calling this `MessageBox` is `Yes`, which means that the user has confirmed that this data deleting is fine, the data deleting will be performed in the next step.
- D. If the user selected the first method, the `TableAdapter DBDirect` method, the query function we built in Section 7.2.4.2 will be called to perform the data deleting from the `Faculty table` in the database.
- E. The execution result of the first method is stored in the local variable `intDelete`.
- F. If the user selected the second method, `TableAdapter.Update()`, the associated codes that will be developed in the next step will be executed to delete data first from the `DataTable` in the `DataSet` and then from the data table in the database by executing the `Update()` method.

```

AccessUpdateDeleteWizard_Project.FacultyFo  cmdDelete_Click()
private void cmdDelete_Click(object sender, EventArgs e)
{
A   MessageBoxButtons vbButton = MessageBoxButtons.YesNo;
    CSE_DEPTDataSet.FacultyRow FacultyRow;
    DialogResult Answer;
    string strFacultyID;
    int intDelete = 0;

B   Answer = MessageBox.Show("Sure to delete this record?", "Delete", vbButton);
C   if (Answer == System.Windows.Forms.DialogResult.Yes)
    {
D       if (ComboBox.Text == "TableAdapter DBDirect")
E           intDelete = facultyTableAdapter.DeleteFaculty(ComboBox.Text);
F       else
        {
            // TableAdapter Update() method is selected....
        }
    }
G   if (intDelete == 0)
        MessageBox.Show("Faculty Table Deleting is failed!");
}

```

Figure 7.8 Coding for the Delete button's Click method.

- G.** The returned value of calling either the `TableAdapter.Delete()` method or the `TableAdapter.Update()` method is an integer value, and it is stored in the variable `intDelete`. The value of this returned data is equal to the number of deleted data rows in the database or deleted `DataRow`s in the `DataSet`. A returned zero value means that no data row has been deleted and this data deleting has failed. In that case, a warning message is displayed and the procedure is exited.

Now let's take a look at the coding for the data deleting using the second method.

7.2.8 Develop Codes to Delete Data Using `TableAdapter.Update` Method

Add the codes shown in Figure 7.9 into the Delete button's Click method, that is, into the `else` block. The codes we developed in the previous step have been highlighted with shading. Let's take a close look at this piece of new added code to see how it works.

- A.** To identify the `DataRow` to be deleted from the `DataTable`, a default system method `FindBy()` will be utilized. However, that method needs to use the `faculty_id` as a criterion; therefore, we first need to retrieve the `faculty_id` from the Faculty table based on the Faculty Name selected by the user.
- B.** After the `faculty_id` is found, the default method `FindByfaculty_id()` is executed to locate the desired `DataRow` from the Faculty table, and the desired `DataRow` is returned and assigned to the local variable `FacultyRow`.
- C.** The `Delete()` method of the `FacultyRow` is executed to delete the selected `DataRow` from the Faculty `DataTable` in the `DataSet`.
- D.** The `TableAdapter.Update()` method is executed to update the deleted `DataRow` from the `DataSet` to the data row in the database.

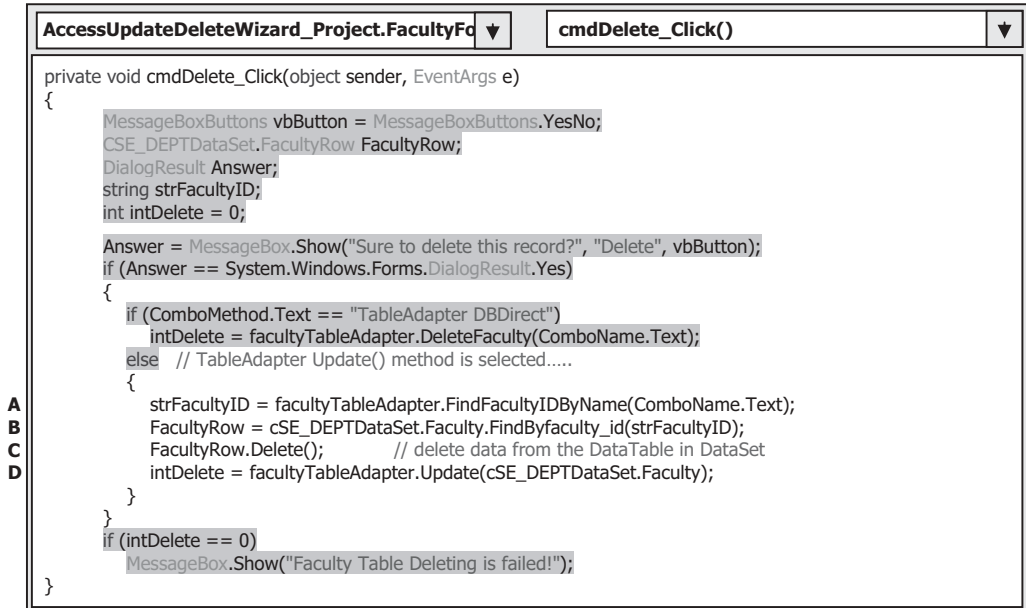


Figure 7.9 Coding for the second data deleting method.

Before we can run the project to test our coding for the data updating and deleting, let's first complete the coding for the data validations after those data actions.

7.2.9 Validate Data After Data Updating and Deleting

As we mentioned in the previous section, we do not need to develop any code for these data validations, and we can use the coding we developed for the Select button's Click method to perform these validations.

Fine. Now let's run the project to test our coding for the data updating and data deleting. Click on the Start Debugging button to run the project, and enter the suitable username and password, such as *jhenry* and *test*, to the LogIn Form, and then select the Faculty Information item from the Selection Form window to open the Faculty Form. Make sure that the faculty member Ying Bai is selected from the Faculty Name combobox ComName, and then click on the Select button to retrieve and display all pieces of information for that selected faculty.

Now update this record by entering the information listed below to modify this faculty record:

- Professor Title textbox
- MTC-219 Office textbox
- 750-378-5577 Phone textbox

Keep the content of all other textboxes unchanged, and click on the Update button to update this new record to the database. You can use either the TableAdapter DBDirect or the TableAdapter.Update method by selecting it from the Method combobox.

The screenshot shows a window titled "CSE DEPT Faculty Form". On the left, there is a "Method" dropdown menu set to "TableAdapter DBDirect" and a small photo of a man. On the right, there is a "Faculty Name" dropdown menu set to "Ying Bai". Below this is a "Faculty Information" section with several text boxes: "Faculty Name" (Ying Bai), "Title" (Professor), "Office" (MTC-219), "Phone" (750-378-5577), "College" (Florida Atlantic University), and "Email" (ybai@college.edu). At the bottom, there are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 7.10 Updated faculty record.

To validate this updating, make sure that the faculty name of the updated record is in the ComboName combobox and then click on the Select button to retrieve that updated record. You can see that the faculty record is indeed updated, as shown in Figure 7.10.

To delete this faculty record, click on the Delete button with either the TableAdapter DBDirect or the TableAdapter.Update method as you like. A MessageBox is displayed to ask you to confirm this deletion. Click on Yes if you want to delete it. Then you can validate that deletion by clicking on the Select button to try to retrieve that deleted record. What happened after you clicked on the Select button? A message “No matched faculty found” is displayed to indicate that the faculty record has been deleted from the database.

Note that when you update the faculty name by changing the content of the Faculty Name textbox, you go to the ComboName combobox to select the modified faculty name from that box to perform the data validation by clicking on the Select button after you finished updating that record. You need to perform the same operations if you want to delete that record from the database. The key is that the content of the Faculty Name textbox may differ from the content of the Faculty Name combobox ComboName, and the former is an updated faculty name and the latter is an old faculty name if an updating of the faculty name is performed.

Our project is very successful!

In order to keep our sample database neat and complete, it is highly recommended to recover those deleted faculty records in the Faculty table and courses related to the deleted faculty member in the Course table to our sample database CSE_DEPT. You can perform recovering the deleted faculty member in either way:

1. Open our sample database CSE_DEPT.accdb to recover the deleted faculty record.
2. Use the Insert Faculty Form window to do that recovery.

When using the second method to do this recovery, enter the following information to the associated textbox in the Insert Faculty Form window to recover the deleted faculty member, Ying Bai:

- 78880 Faculty ID textbox
- Ying Bai Faculty Name textbox
- Associate Professor Title textbox
- MTC-211 Office textbox
- 750-378-1148 Phone textbox
- Florida Atlantic University College textbox
- ybai@college.edu Email textbox

We prefer to use the second method since it is more professional to do this recovery job. Refer to Table 2.15 in Section 2.9.3 in Chapter 2 to recover those deleted courses related to the deleted faculty. Four courses related to the deleted faculty member Ying Bai are: CSC-132B, CSC-234A, CSE-434, and CSE-438. Recover these courses one by one based on Table 2.15. A complete project `AccessUpdateDeleteWizard` is located at the folder `DBProjects\Chapter 7` located at the accompanying ftp site (See Chapter 1).

7.3 UPDATE AND DELETE DATA FOR SQL SERVER DATABASE

In order to save time and space, we can still modify the existing project `SQLInsertWizard` we developed in the last chapter to create our new project named `SQLUpdateDeleteWizard` and use it in this chapter. Open Windows Explorer and create a new folder, such as `Chapter 7` if you have not created it, and browse to our project `SQLInsertWizard` that can be found from the folder `DBProjects\Chapter 6` located at the accompanying ftp site (See Chapter 1). Copy this project to our new folder `Chapter 7`. Change the name of the solution and the project from `SQLInsertWizard` to `SQLUpdateDeleteWizard` and double-click on this new project `SQLUpdateDeleteWizard.csproj` to open it.

On the opened project, perform the following modifications to get our desired project:

- Go to the `Project\SQLUpdateDeleteWizard Project Properties` menu item to open the project's property window. Change the `Assembly name` from `SQLInsertWizard` to `SQLUpdateDeleteWizard` and the `Default namespace` from `SQLInsertWizard` to `SQLUpdateDeleteWizard`, respectively.
- Click on the `Assembly Information` button to open the `Assembly Information` dialog box, change the `Title` and the `Product` to `SQLUpdateDeleteWizard`. Click on `OK` to close this dialog box.
- Update the project namespace for all project files by changing it from the old namespace `SQLInsertWizard` to `SQLUpdateDeleteWizard`, which is our new project namespace. Follow the steps below to complete this namespace updating:
 1. Go to `Edit\Quick Find` menu item to open the `Find and Replace` dialog box.
 2. Enter `SQLInsertWizard` into the `Find what` box.
 3. Select `Entire Solution` from the `Look in` box.
 4. Click on `Find Next` button to find each old namespace.
 5. Then click on the `Quick Replace` button and enter `SQLUpdateDeleteWizard` into the `Replace with` box.
 6. Click on the `Replace` or `Replace All` button to complete this namespace updating.

Repeat steps 4 to 6 to update all project namespaces. Then go to the `File\Save All` to save those modifications.

Because of the similarity between this project `SQLUpdateDeleteWizard` and the `AccessUpdateDeleteWizard` project developed in the last section, we will not duplicate any identical parts. Basically, both the graphical user interfaces and the coding of this project are exactly identical with those of the project developed in the last section. Perform the following operations to finish the modifications to this project:

1. Refer to Section 7.2.2.1 to modify the graphical user interface, which include the modifications to the Faculty Form window.
2. Refer to Section 7.2.2.2 to finish the data bindings between the textbox controls on the Faculty Form and the associated columns in the Faculty table in the DataSet.
3. Refer to Section 7.2.4 to finish building of the Update and Delete queries.
4. Refer to Sections 7.2.5 and 7.2.6 to finish the coding for the updating data using the TableAdapter `DBDirect` method and `TableAdapter.Update()` method.
5. Refer to Sections 7.2.7 and 7.2.8 to finish the coding for deleting data using the TableAdapter `DBDirect` method and `TableAdapter.Update()` method.
6. Refer to Section 7.2.9 to finish the coding for validating the data after data updating and deleting.

Some important points related to these modifications are discussed below.

To make modifications on step 1 simple, you can first delete all controls from the Faculty Form window in the project `SQLUpdateDeleteWizard` by (1) going to the `Edit>Select All` menu item to select all controls and then (2) going to the `Edit>Delete` menu item. Then open the project `AccessUpdateDeleteWizard` and its Faculty Form window, select all controls from that form by going to the `Edit>Select All` menu item and copy all of them by going to the `Edit|Copy` menu item. Finally paste these controls to the Faculty Form window in the project `SQLUpdateDeleteWizard`.

Note that this copy-paste operation a `FacultyBindingSource`, which belongs to the Faculty Form in the project `AccessUpdateDeleteWizard`, will also be copied and pasted to our new project. We need to delete it from our new project since we will not use that binding source in this project.

The modifications to step 3 are to build the Update and Delete queries. There are some differences between building those queries in Microsoft Access database and the SQL Server database. First let's discuss how to build the Update query in the SQL Server database environment.

Open the Data Source window and the TableAdapter Query Configuration Wizard. Right-click on the last item in the Faculty table and select the `Add Query` item from the pop-up menu. Keep the default selection: `Use SQL statements unchanged` and click on `Next` to go to the next window. Select and check the `UPDATE` item from this window since we need to perform updating data query, and then click on `Next` again to continue. Click on the `Query Builder` button to open the `Query Builder` dialog box since we want to build our updating query. The opened `Query Builder` dialog is shown in Figure 7.11.

Remove the item `=@Original` under the Filter column along the `faculty_id` row, and place a question symbol? under the Filter column along the `faculty_name` row in the midpane, and press the `Enter` key on the keyboard. Uncheck the `Set` checkbox for the `faculty_id` row to remove this item. Your finished update `Query Builder` should match the one shown in Figure 7.11.

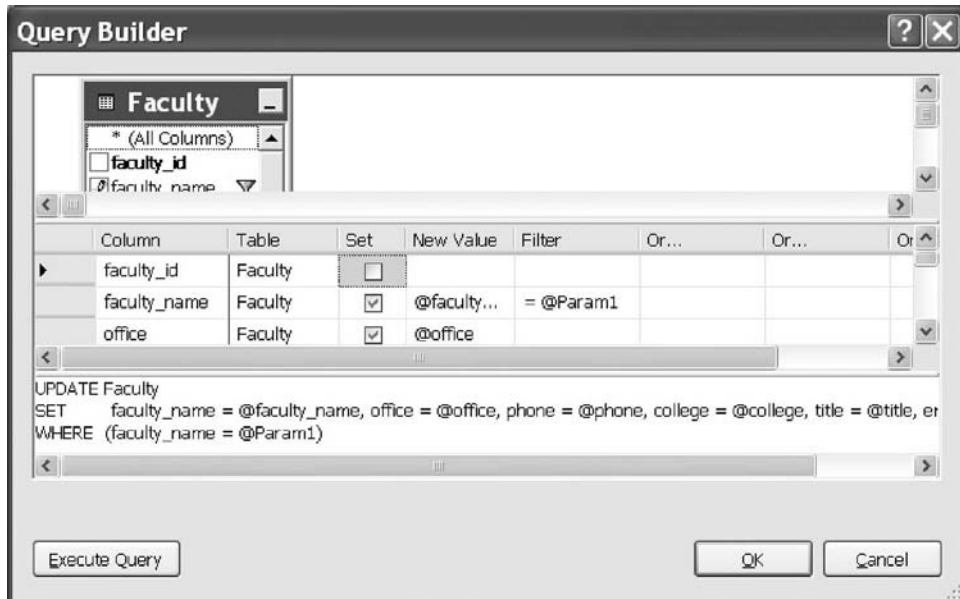


Figure 7.11 Update Query Builder.

Click on OK the button to continue. In the next window, remove the SELECT query since we do not need that query. Click on Next to go to the next window. Change the function name to UpdateFaculty, click on the Next and Finish buttons to close this query builder.

Perform similar operations to build the Delete query, which is shown in Figure 7.12, and name the query function as DeleteFaculty.

To add the faculty_name column into this Delete query as a criterion, click on the second row, which is just under the first row faculty_id, from the midpane, and then click on the drop-down arrow to select the faculty_name column from the list. Type a question mark? in the Filter column along the faculty_name row. Also remove the item =@Original from the Filter column along the faculty_id row in the midpane. Your finished Delete query should match the one shown in Figure 7.12.

Perform steps 4 to 6 listed above to complete these modifications to our new project. Now you can try to run this project to test the data updating and deleting functions.

Recall that when we built our sample database CSE_DEPT, that is, when we set up the relationships among tables, we selected the Cascade mode for both Update and Delete Rules for INSERT and UPDATE Specification field between the Faculty and LogIn, Faculty and Course, and Course and StudentCourse tables. Five points are important after you run this new project to test the data updating and deleting functions:

1. First, it is highly recommended that you recover the deleted faculty member after you have finished testing the delete query in this project since we want to keep our database neat and complete. Refer to Table 2.19 in Section 2.10.2 in Chapter 2 to get detailed information for faculty members to recover the deleted faculty member in the Faculty table.
2. Refer to Table 2.18 in Section 2.10.1 in Chapter 2 to recover the deleted faculty login username and password for the deleted faculty member Ying Bai in the LogIn table.

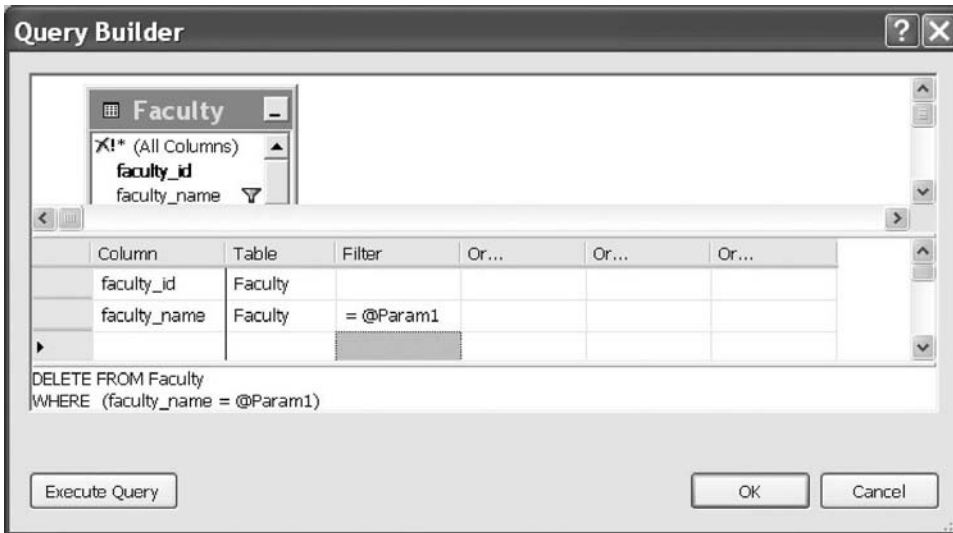


Figure 7.12 Delete Query Builder.

3. Refer to Table 2.20 in Section 2.10.3 in Chapter 2 to recover those deleted courses related to the deleted faculty in the Course table. Four courses related to the deleted faculty member Ying Bai are CSC-132B, CSC-234A, CSE-434, and CSE-438. Recover those course one by one based on Table 2.20. The reason for these deleted courses is that we set the Cascade Delete mode when we built the relationships for our sample database CSE_DEPT in Chapter 2. This means that when a faculty member is deleted from the Faculty table (parent table), all course records related to that deleted faculty will also be cascade deleted from the Course table (child table).
4. Refer to Table 2.22 in Section 2.10.3 in Chapter 2 to recover those deleted courses related to the deleted faculty in the StudentCourse table. Four courses related to the deleted faculty member Ying Bai are CSC-132B, CSC-234A, CSE-434, and CSE-438. Recover those courses one by one based on Table 2.22. The reason for those deleted courses is that we set the Cascade Delete mode when we built the relationships for the Course and the StudentCourse tables. This means that when a course is deleted from the Course table (parent table), all courses (course_id) related to that deleted course in the Course table will also be cascade deleted from the StudentCourse table (child table).
5. Some buttons may not response to your clicking on them as the project runs. This is because we delete all original controls from this Faculty Form window, and copy and paste all controls from the Faculty Form in another project AccessUpdateDeleteWizard to this Faculty Form. The relationship between each button and its delegated method is also removed with our deletion for those original controls in the Faculty Form. To fix these errors, we need to re-do these delegation relationships between each button and its method one by one. Just double-click on each button to open its delegated method and move the codes from the old method to this new one. You can also change the name of the new opened method to the original one if you like. However, you need to first remove the old method, and then change the name of the new method to the original one from the associated Form's Designer.cs file by compiling and building the project.

A complete project SQLUpdateDeleteWizard can be found at the folder DBProjects\Chapter 7 located at the accompanying ftp site (See Chapter 1).

7.4 UPDATE AND DELETE DATA FOR ORACLE DATABASE

It is very similar to develop a Visual C#.NET project to modify data against the Oracle database using the Update and Delete commands. The only difference is the Data Source to be connected to your applications. Refer to Appendix E to add and connect the sample Oracle 10g XE database CSE_DEPT with the Visual C#.NET application using the Design Tools and Wizards. Also refer to Appendix F to get detailed information on how to use the sample database. All user interfaces and codes are identical with those codes we developed for the last project. Appendix E also provides sample coding to use different query methods to perform the data accessing and insertion functions.

PART II DATA UPDATING AND DELETING WITH RUNTIME OBJECTS

To update or delete data against the database using the runtime objects method is a flexible and professional way to perform the data modification jobs in the Visual C#.NET environment. Compared with the method discussed in Part I, in which Visual Studio.NET Design Tools and Wizards are utilized to update or delete data against the database, the runtime objects method provides more sophisticated techniques to do this job efficiently and conveniently. However, a more complicated coding job is needed. Relatively speaking, the method we discussed in the first part is easy to learn and code, but some limitations exist for that method. First, each TableAdapter can only access the associated data table to perform data actions such as updating or deleting data against that table only. Second, each query function built by using the TableAdapter Query Configuration Wizard can only perform a single query such as data updating or deleting. Third, after the query function is built, no modifications can be made to that function dynamically, which means that the only time you can modify that query function is either before the project runs or after the project terminates. In other words, you cannot modify that query function as the project runs.

To overcome these shortcomings, we will discuss how to update or delete data using the runtime objects method in this part. Basically, you need to use the TableAdapter to perform data actions in the database if you develop your applications using the Visual Studio.NET Design Tools and Wizards in the design time. However, you should use the DataAdapter to make those data manipulations if you develop your project using the runtime objects method.

7.5 RUNTIME OBJECTS METHOD

We have provided a very detailed introduction and discussion about the runtime objects method in Section 5.17 in Chapter 5. Refer to that section for more detailed information about this method. For your convenience, we highlight some important points and general methodologies of this method as well as some key points in using this method to perform data updating and deleting in the databases.

As you know, ADO.NET provides different classes to help users develop professional data-driven applications by using the different methods to perform specific data

actions such as updating data and deleting data. Among them, two popular methods are widely applied:

1. Update or delete records from the desired data table in the DataSet, and then call the `DataAdapter.Update()` method to update the updated or deleted records from the table in the DataSet to the table in the database.
2. Build the update or delete commands using the Command object, and then call the Command's method `ExecuteNonQuery()` to update or delete records in the database. Or you can assign the built command object to the `UpdateCommand` or `DeleteCommand` properties of the `DataAdapter` and call the `ExecuteNonQuery()` method from the `UpdateCommand` or `DeleteCommand` property.

The first method is to use the so-called DataSet-DataAdapter method to build a data-driven application. DataSet and DataTable classes can have different roles when they are implemented in a real application. Multiple DataTables can be embedded into a DataSet and each table can be filled, inserted, updated, and deleted by using the different properties of a DataAdapter such as the `SelectCommand`, `InsertCommand`, `UpdateCommand`, or `DeleteCommand` when the DataAdapter's `Update()` method is executed. The DataAdapter will perform the associated operations based on the modifications you made for each table in the DataSet. For example, if you deleted rows from a table in the DataSet, then call this DataAdapter's `Update()` method. This method will perform a `DeleteCommand` based on your modifications. This method is relatively simple since you do not need to call some specific methods such as the `ExecuteNonQuery()` to complete these data queries. However, this simplicity brings some limitations for your applications. For instance, you cannot access different data tables individually to perform multiple specific data operations. This method is very similar to the second method we discussed in Part I; therefore, we will not continue the discussion for this method.

The second method allows us to use each object individually, which means that you do not have to use the DataAdapter to access the Command object or use the DataTable together with the DataSet. This provides more flexibility. In this method, no DataAdapter or DataSet is needed, and you only need to create a new Command object with a new Connection object, and then build a query statement and attach some useful parameters into that query for the new created Command object. Then you can update or delete data against any data table by calling the `ExecuteNonQuery()` method, which belongs to the Command class. We will concentrate on this method in this part.

In this section, we provide three sample projects named `SQLUpdateDeleteRTObject`, `AccUpdateDeleteRTObject`, and `OracleUpdateDeleteRTObject` to illustrate how to update or delete records in three different databases using the runtime object method. Because of the coding similarity between these three databases, we will concentrate on updating and deleting data in the SQL Server database using the sample project `SQLUpdateDeleteRTObject` first, and then illustrate the coding differences between these databases by using the real codes for the rest of two sample projects.

In addition to those three sample projects, we will also discuss data updating and deleting in our sample databases using the LINQ to SQL query method. A sample project `LINQSQLUpdateDelete` will be developed in this chapter to show readers how to build an actual data-driven project to update and delete data against our sample databases using the LINQ to SQL query method.

7.6 UPDATE AND DELETE DATA FOR SQL SERVER DATABASE USING RUNTIME OBJECTS

Now let's first develop the sample project SQLUpdateDeleteRuntimeObject to update and delete data in the SQL Server database using the runtime objects method. Recall in Sections 5.19.2.3 to 5.19.2.7 in Chapter 5, we discussed how to select data for the Faculty, Course, and Student Form windows using the runtime objects method. For the Faculty Form, a regular runtime selecting query is performed, and for the Course Form, a runtime joined-table selecting query is developed. For the Student table, the stored procedures are used to perform the runtime data query.

Similarly in this part, we divide this discussion into two sections:

1. Update and delete data in the Faculty table from the Faculty Form window using the runtime objects method.
2. Update and delete data in the Faculty table from the Faculty Form using the runtime stored procedure method.

In order to avoid duplication of the coding, we will modify an existing project named SQLInsertRuntimeObject developed in Chapter 6 to create our new project SQLUpdateDeleteRuntimeObject used in this section.

Open Windows Explorer and create a new folder such as Chapter 7 if you have not, and then browse to the folder DBProjects\Chapter 6 located at the accompanying ftp site (See Chapter 1), and copy the project SQLInsertRuntimeObject to the new folder C:\Chapter 7 we just created. Change the name of the project from SQLInsertRuntimeObject to SQLUpdateDeleteRuntimeObject. Double-click on the SQLUpdateDeleteRuntimeObject.csproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to Project\SQLUpdateDeleteRuntimeObject Properties menu item to open the project's property window. Change the Assembly name and the Default namespace from SQLInsertRuntimeObject to SQLUpdateDeleteRuntimeObject, respectively.
- Click on the Assembly Information button to open the Assembly Information dialog box, and change the Title and the Product to SQLUpdateDeleteRuntimeObject. Click on the OK to close this dialog box.
- Change the project namespace for all project files from SQLInsertRuntimeObject to SQLUpdateDeleteRuntimeObject using the Find and Replace dialog box.

Go to the File|Save All to save those modifications. Now we are ready to develop our graphic user interfaces based on our new project SQLUpdateDeleteRuntimeObject.

7.6.1 Update Data in Faculty Table for SQL Server Database

Let's first discuss updating data in the Faculty table for the SQL Server database. To update data in the Faculty data table, we do not need to add any new window forms, and we can use the Faculty Form window as the user interface. However, we need to perform the following four steps to modify this project:

1. Modify the current Faculty Form window.
2. Modify the codes in the Faculty Form and the Insert Faculty Form code windows.
3. Develop codes to update data.
4. Validate the data updating.

First, we need to modify the Faculty Form to make it suitable for our data updating.

7.6.1.1 Modify Faculty Form Window

Recall that when we developed the Faculty Form for the project SQLInsertRTOBJect in the last chapter, five labels were developed in that form to store the faculty information. In order to update records in the Faculty table, we need a way to enter new faculty information into some controls and update that record later. The textbox is a good candidate to receive and store a piece of new faculty information. Therefore the first job we need to do is to replace all of those five labels with five textbox controls and add one more textbox, Faculty Name textbox, into the Faculty Information group box since we can update a faculty member with six pieces of new information without touching the `faculty_id` column.

A good and simple way to modify this Faculty Form window is to first remove all controls from the current Faculty Form window, and then copy all controls from the Faculty Form window in the project SQLUpdateDeleteWizard we developed in this chapter, and paste them into the current Faculty Form window. To do that, first remove all controls from the current Faculty Form window by clicking on the **Edit/Select All**, and then **Edit/Delete** menu items. Next open the project SQLUpdateDeleteWizard and its Faculty Form window, select **Edit/Select All**, and then **Edit/Copy** to copy all controls from that Faculty Form window. Now open our current Faculty Form window, going to the **Edit/Paste** menu item to paste them into our current form. Your finished Faculty Form window should match the one shown in Figure 7.13.

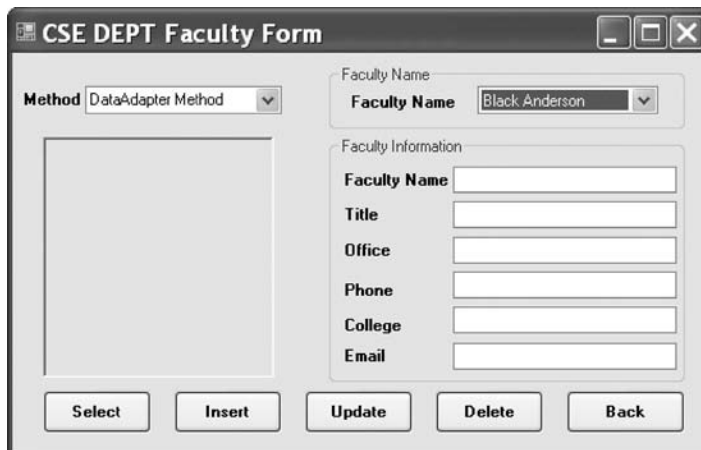


Figure 7.13 Modified Faculty form window.

Note that when you perform this copy-paste operation, an object `FacultyBindingSource` that belongs to the project `SQLUpdateDeleteWizard` will also be copied and pasted into the Faculty Form. Remove this object since we do not need it in this project.

7.6.1.2 *Modify Original Coding in Faculty Form*

The code modifications to the Faculty Form can be divided into two parts: (1) replace the field-level label array `FacultyLabel[]` used to store faculty information with textbox array `FacultyTextBox[]` and (2) modify the codes related to those textboxes.

Let's begin this modification from the first part. Open the code window of the Faculty Form window, and change the field-level label collection `FacultyLabel` to the textbox collection `FacultyTextBox`. Your finished textbox collection `FacultyTextBox` should look like:

```
private TextBox[] FacultyTextBox = new TextBox[7];
```

The code modification in the second part relates to codes in the three user-defined methods: `FillFacultyTable()`, `MapFacultyTable()`, and `FillFacultyReader()`. Open these methods and perform the following modifications shown in Figure 7.14 to these methods. All modified codes have been highlighted in bold.

Let's take a close look at these modifications to see how they work.

- A. Replace the `FacultyLabel` array with the `FacultyTextBox` array in the user-defined method `FillFacultyTable()`. Also change the class name from `Label` to `TextBox`. Perform the same modifications to the codes inside the method `FillFacultyReader()`, which is shown in step E in Figure 7.14.
- B. Change the passing-argument object in the user-defined method `MapFacultyTable()` from the label array `FacultyLabel` to the textbox array `FacultyTextBox`. Perform the same modifications to the codes inside the method `FillFacultyReader()`, which is shown in step F in Figure 7.14.
- C. Change the label array `FacultyLabel` to the textbox array `FacultyTextBox` inside the `foreach` loop.
- D. Change the nominal passing-argument object in the user-defined method `MapFacultyTable()` from the label array `fLabel` to the textbox array `fTextBox`. Expand the lower bound of the `FacultyTextBox` array to 1 since we need to update six pieces of faculty information in our database. Also replace six label objects with six textbox objects and assign them to the associated textbox control in the Faculty Form window.
- E. Change the label array `FacultyLabel` to the textbox array `FacultyTextBox` inside the `for` loop in the `FillFacultyReader()` method.

Note that some buttons may not respond to your clicking on them as the project runs. These buttons may include the Select, Back, and Insert buttons. This is because we delete all original controls from this Faculty Form window, and copy and paste all controls from the Faculty Form in another project `SQLUpdateDeleteWizard` to this Faculty Form. The relationship between each button and its delegated method is also removed with our deletion for those original controls in the Faculty Form. To fix these errors, we can re-do these delegation relationships between each button and its method one by one. Just double-click on each button to open its delegated method and move the codes from the old method to this new one. You can also change the name of the new opened method


```

SQLUpdateDeleteRTOject.FacultyForm | FillFacultyReader()
private void FillFacultyTable(ref DataTable FacultyTable)
{
    int pos1 = 0;
    for (int pos2 = 0; pos2 <= 6; pos2++) //Initialize the object array
    A   FacultyTextBox[pos2] = new TextBox(); // modified on 12-3-2008
    B   MapFacultyTable(FacultyTextBox); //
    foreach (DataRow row in FacultyTable.Rows)
    {
        foreach (DataColumn column in FacultyTable.Columns)
        C   {
            FacultyTextBox[pos1].Text = row[column].ToString(); //
            pos1++;
        }
    }
    D private void MapFacultyTable(Object[] fText) // modified on 12-3-2008
    {
        fText[1] = txtName;
        fText[2] = txtOffice; //The order must be identical
        fText[3] = txtPhone; //with the real order in the query string
        fText[4] = txtCollege;
        fText[5] = txtTitle;
        fText[6] = txtEmail;
    }
    private void FillFacultyReader(SqlDataReader FacultyReader)
    {
        int intIndex = 0;
        for (int intIndex = 0; intIndex <= 6; intIndex++) //Initialize the object array
        E   FacultyTextBox[intIndex] = new TextBox();
        F   MapFacultyTable(FacultyTextBox);
        while (FacultyReader.Read())
        {
            for (int intIndex = 0; intIndex <= FacultyReader.FieldCount - 1; intIndex++)
            G   FacultyTextBox[intIndex].Text = FacultyReader.GetString(intIndex);
        }
    }
}

```

Figure 7.14 Modifications to the Faculty Form.

to the original one if you like. However, you need to first remove the old method, and then change the name of the new method to the original one from the associated Form's Designer.cs file by compiling and building the project.

Well, quite a few modifications have made in this part. However, that is a good thing since we can save a lot of time when we develop the next project, OracleUpdate DeleteRTOject, by just making a few modifications to the current project. Now let's begin to develop the codes for our data updating and deleting parts.

7.6.1.3 Develop Codes to Update Data

As we mentioned in the previous sections, to update or delete an existing record from our related tables, one must follow the three steps listed in Section 7.1.1. Open the Update button's Click method by double-clicking on the Update button from the Faculty Form window and enter the codes shown in Figure 7.15 into this method.

```

SQLUpdateDeleteRTOject.FacultyForm  cmdUpdate_Click()
private void cmdUpdate_Click(object sender, EventArgs e)
{
A   string cmdString = "UPDATE Faculty SET faculty_name = @name, office = @office, phone = @phone, " +
    "college = @college, title = @title, email = @email " +
    "WHERE (faculty_name LIKE @Param1)";
B   LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    SqlCommand sqlCommand = new SqlCommand();
    int intUpdate = 0;
C   sqlCommand.Connection = logForm.sqlConnection;
    sqlCommand.CommandType = CommandType.Text;
    sqlCommand.CommandText = cmdString;
D   UpdateParameters(ref sqlCommand);
E   intUpdate = sqlCommand.ExecuteNonQuery();
F   sqlCommand.Dispose();
    ComboName.Items.Clear();
    UpdateFaculty();
G   if (intUpdate == 0)
        MessageBox.Show("The data updating is failed");
}
H private void UpdateParameters(ref SqlCommand cmd)
{
    cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text;
    cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text;
    cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text;
    cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text;
    cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text;
    cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text;
    cmd.Parameters.Add("@Param1", SqlDbType.Char).Value = ComboName.Text;
}

```

Figure 7.15 Coding for the data updating operation.

Let's take a look at this piece of code to see how it works.

- A.** The Update query string is defined first at the beginning of this method. All six data columns in the Faculty table are input parameters. The dynamic parameter `@Param1` represents the old faculty name, which is the faculty name that has not been updated.
- B.** All data components and local variables are declared here such as the Command object, LogInForm object, and `intUpdate`. The LogInForm object and the `getLogInForm()` method are used to access the Connection object we built in the LogIn Form object and initialize the Command object with the Connection object below. The integer variable `intUpdate` is used to hold the returned data from calling the `ExecuteNonQuery()` method.
- C.** The Command object is initialized and built using the Connection object and the Parameter object.
- D.** A user-defined method `UpdateParameters()` is called to add all updating parameters into the Command Parameters' property. The passing mode used for the passed argument is passing by reference, which means that a valid starting address of that Command object is passed into the method, and any modification to this Command object is permanent and it can be returned to the calling method.
- E.** Then the `ExecuteNonQuery()` method of the Command class is executed to update the faculty table. The running result of this method is returned and stored in the local variable `intUpdate`.

- F. The Command object is released after this data updating, and the updated faculty members are refreshed in the ComboName box by executing the UpdateFaculty() method. Before this updating can be refreshed, the ComboName box is cleaned up by running its Clear() method.
- G. The returned value from calling the ExecuteNonQuery() method is equal to the number of rows that have been updated in the Faculty table. A zero means that no row has been updated, an error message is displayed, and the procedure is exited if this situation occurred.
- H. The detailed coding for the user-defined method UpdateParameters() is shown in this step. Six pieces of new faculty information are assigned to the associated columns in the Faculty table.

At this point, we have finished the coding for the data updating operation for the Faculty table. Next let's take care of the data validation after this data updating to confirm that our data updating is successful.

7.6.1.4 Validate Data Updating

We do not need to add any new form windows to perform this data validation, and we can use the Faculty Form window to perform this validation operation. By clicking on the Select button on the Faculty Form window, we can perform the selection query to retrieve the updated faculty record from the database and display it on the Faculty Form.

Before we can run the project to test the data updating function, we prefer to first complete the coding for the data deleting operation.

7.6.2 Delete Data from Faculty Table for SQL Server Database

As we mentioned in the previous section, to delete a faculty record from our database, we have to follow the two steps listed below:

1. First, delete records from the child tables (LogIn and Course tables).
2. Second, delete records from the parent table (Faculty table).

The data deleting function can be performed by using the Delete button's Click method in the Faculty Form window. Therefore, the main coding for this functionality is developed inside that method.

7.6.2.1 Develop Codes to Delete Data

Open the Delete button's Click method by double-clicking on the Delete button from the Faculty Form window, and enter the codes shown in Figure 7.16 into this method.

Let's take a close look at this piece of code to see how it works.

- A. The deleting query string is declared first at the beginning of this method. The only input parameter is the `faculty_name`. Although the primary key of the Faculty table is `faculty_id`, in order to make it convenient to the user, the `faculty_name` is used as the criterion for this data deleting query. A potential problem of using the `faculty_name` column as the deleting criterion is that no duplicated `faculty_name` should exist in the Faculty table for this application. In other words, each faculty name must be unique

```

SQLUpdateDeleteRTOject.FacultyForm  cmdDelete_Click()
private void cmdDelete_Click(object sender, EventArgs e)
{
A   string cmdString = "DELETE FROM Faculty WHERE (faculty_name LIKE @Param1)";
B   MessageBoxButtons vbButton = MessageBoxButtons.YesNo;
C   LogInForm logForm = new LogInForm();
   logForm = logForm.getLogInForm();
   SqlCommand sqlCommand = new SqlCommand();
   DialogResult Answer;
   int intDelete = 0;
D   Answer = MessageBox.Show("Do you want to delete this record?", "Delete", vbButton);
E   if (Answer == System.Windows.Forms.DialogResult.Yes)
   {
F       sqlCommand.Connection = logForm.sqlConnection;
G       sqlCommand.CommandType = CommandType.Text;
H       sqlCommand.CommandText = cmdString;
I       sqlCommand.Parameters.Add("@Param1", SqlDbType.Char).Value = ComboName.Text;
       intDelete = sqlCommand.ExecuteNonQuery();
       sqlCommand.Dispose();
       if (intDelete == 0)
           MessageBox.Show("The data Deleting is failed");
J       for (intDelete = 0; intDelete < 7; intDelete++) // clean up the Faculty textbox array
           FacultyTextBox[intDelete].Text = string.Empty;
   }
}

```

Figure 7.16 Coding for the data deleting query.

in the Faculty table. A solution to this potential problem is that we can use the `faculty_id` as the criterion for the data deleting query in the future.

- B.** A `MessageBox` button's object is created, and this object is used to display both buttons in the `MessageBox`, Yes and No, when the project runs.
- C.** All data components and local variables used in this method are declared here, too. The data type of the variable `Answer` is `DialogResult`. However, one can use an integer variable to replace it. The `LogInForm` object and the `getLogInForm()` method are used to access the `Connection` object we built in the `LogInForm` object and initialize the `Command` object with that `Connection` object below. The integer variable `intDelete` is used to hold the returned data from calling the `ExecuteNonQuery()` method to delete a record from the Faculty table.
- D.** As the `Delete` button is clicked when the project runs, first a `MessageBox` is displayed to confirm that the user wants to delete the selected member from the Faculty table.
- E.** If the user's answer to the `MessageBox` is Yes, then the deleting operation begins to be processed. The `Command` object is initialized and built by using the `Connection` object and the `Command` string we defined at the beginning of this procedure.
- F.** The dynamic parameter `@Param1` is replaced by the real parameter, the faculty name stored in the combobox `ComboName`. A key point to note is that you must use the faculty name stored in the combobox control, which is an existing faculty name. However, you cannot use the faculty name stored in the Faculty Name textbox since that is an updating faculty name.
- G.** The `ExecuteNonQuery()` method of the `Command` class is called to execute the data deleting query to the Faculty table. The running result of calling this method is stored in the local variable `intDelete`.

- H. The Command object is released after the data deleting.
- I. The returned value from calling of the `ExecuteNonQuery()` method is equal to the number of rows that have been successfully deleted from the Faculty table. If a zero returns, which means that no row has been deleted from the Faculty table and this data deleting has failed, an error message is displayed and the method is exited if that situation occurred.
- J. After the data deleting is done, all faculty information stored in the six textboxes should be cleaned up. A `for` loop is used to finish this cleaning job.

Finally let's take care of the coding to validate the data deleting query.

7.6.2.2 *Validate Data Updating and Deleting*

As we did for the validation of the data updating in the last section, we do not need to create any new form window to do this validation, and we can use the Faculty Form to perform this data validation.

Now let's run the project to test both data updating and data deleting operations. Before we can run the project, make sure that a default faculty photo file named `Default.jpg` has been stored in the default folder in our project if an updating photo is involved in this data updating. In this application, this default folder is the folder in which the executable file of our Visual C#.NET project is located, which is `C:\Chapter 7\SQLUpdateDeleteRTOject\bin\Debug`.

Click on the Start Debugging button to start our project, enter the suitable username and password to the LogIn form, and select the item Faculty Information from the Selection Form to open the Faculty Form window. First, let's select a faculty member and retrieve all information related to the selected faculty from our database, and display those pieces of information in this form. Then we can update this faculty by modifying some pieces of information stored in related textboxes. In this test, we select Ying Bai as a faculty member and enter the following updated information to the related textboxes:

- Professor Title textbox
- MTC-305 Office textbox
- 750-378-1127 Phone textbox

Click on the Update button in the Faculty Form window to update this record.

To validate this data updating, first select another faculty from the combobox control `ComboName` and click on the Select button to retrieve all information for that faculty. Then go to the combobox again and select our updated faculty name Ying Bai from the box, and click on the Select button to retrieve back the updated information for that selected faculty member. Immediately you can find that all pieces of updated information related to the selected faculty are displayed in this form. This means that our data updating is successful. Your updated faculty information window should match the one shown in Figure 7.17.

Next let's test the data deleting function by clicking on the Delete button to try to delete this updated faculty record from the Faculty table. Click on Yes to the message box and all updated faculty information stored in six textboxes is gone. Is our data deleting successful? To answer this question, click on the Select button again to try to retrieve that deleted faculty information from the Faculty table. What happens after you click on



Figure 7.17 Updated faculty information window.

the Select button? A message “No matched faculty found” shows up, and this means that the selected faculty and all information related to that selected faculty have been successfully deleted from the Faculty table. Yes, our data deleting is successful.

To keep our sample database neat and complete, it is highly recommended to recover those updated and deleted faculty records after this testing. To perform this recovering work successfully, the following points must be kept in mind:

1. Recall that when we built our sample database, that is, when we set up the relationships among tables, we selected the **Cascade** mode for both **Update** and **Delete** Rules for **INSERT** and **UPDATE** Specification fields between the Faculty and LogIn, Faculty and Course, and Course and StudentCourse tables. This means that among these tables, the Faculty is a parent table for the LogIn and the Course tables and the LogIn and the Course are child tables to the Faculty table. Similarly, the Course is a parent table for the StudentCourse table, and the StudentCourse is a child table to the Course table. The cascade updating and deleting means that when a record in the parent table is updated or deleted, all related records in the child tables are also updated or deleted. An example of cascade updating and deleting is shown in Figure 7.18. The faculty member Ying Bai with a `faculty_id=B78880` is selected for this cascade updating and deleting example.
2. As shown in Figure 7.18, if the faculty member with a `faculty_id=B78880`, which is a primary key in the Faculty table but a foreign key in the LogIn and the Course tables, is updated or deleted from the Faculty table, the related records in the child tables, LogIn and Course, with the same `faculty_id` will also be updated or deleted with the associated columns. Two associated columns, `user_name` and `pass_word`, located in the LogIn table and four associated columns, `course_id`, located in the Course table will also be updated or deleted. Similarly, if a `course_id` that is a primary key in the Course table but a foreign key in the StudentCourse table is updated or deleted from the Course table, all related records in the child table, StudentCourse, will also be updated or deleted. Figure 7.18 shows the associated columns that will be affected when these cascade updating or deleting actions are performed for the selected faculty member Ying Bai.
3. An important issue is the order to recover these deleted records. You have to first recover the records in the parent table (Faculty and Course tables) and then recover the records in

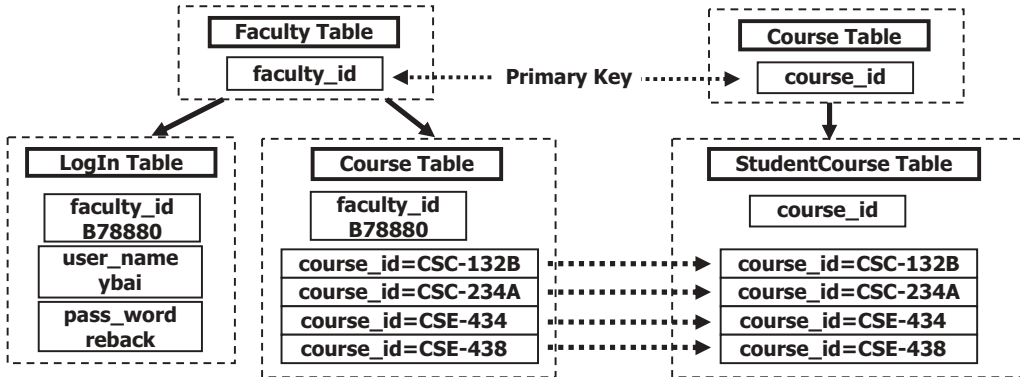


Figure 7.18 Relationships among tables.

Table 7.3 Data to be recovered in the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu

Table 7.4 Data to be recovered in the LogIn Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	NULL

Table 7.5 Data to be recovered in the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880

Table 7.6 Data to be recovered in the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1005	J77896	CSC-234A	3	CS/IS
1009	A78835	CSE-434	3	CE
1014	A78835	CSE-438	3	CE
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE

the child tables. Follow the table order in Figure 7.18 and refer to Sections 2.10.1 to 2.10.3 in Chapter 2 and Tables 7.3, 7.4, 7.5 and 7.6 to complete this record's recovery.

A completed project SQLUpdateDeleteRTOObject can be found from the folder DBProjects\Chapter 7 located at the accompanying ftp site (see Chapter 1).

7.7 UPDATE AND DELETE DATA FOR ORACLE DATABASES USING RUNTIME OBJECTS

Because of the coding similarity between the SQL Server and the Oracle databases for the data updating and deleting, in this section we only show the differences in the coding for both databases. The main differences between the SQL Server and the Oracle databases are the syntax in the query strings for data deleting and updating. In this section, we concentrate on these query strings.

First, let's modify an existing project `SQLUpdateDeleteRTOject` we developed in the last section to create our new project `OracleUpdateDeleteRTOject` used in this section. Open that project and perform the following operations to make it a new project.

Open Windows Explorer and create a new folder such as `Chapter 7` if you have not, and then browse to the folder `DBProjects\Chapter 7` located at the accompanying ftp site (see Chapter 1). Copy the project `SQLUpdateDeleteRTOject` to the new folder `C:\Chapter 7`. Change the name of the project from `SQLUpdateDeleteRTOject` to `OracleUpdataDeleteRTOject`. Double-click on the `OracleUpdataDeleteRTOject.csproj` to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to `Project\OracleUpdataDeleteRTOject Properties` menu item to open the project's property window. Change the `Assembly name` and the `Default namespace` from `SQLUpdateDeleteRTOject` to `OracleUpdataDeleteRTOject`, respectively.
- Click on the `Assembly Information` button to open the `Assembly Information` dialog box, change the `Title` and the `Product` to `OracleUpdataDeleteRTOject`. Click on `OK` to close this dialog box.
- Change the project namespace for all files from `SQLUpdateDeleteRTOject` to `OracleUpdataDeleteRTOject` using the `Find and Replace` dialog box.

Go to the `File\Save All` to save those modifications. Now we are ready to develop our codes based on our new project `OracleUpdataDeleteRTOject`.

We can use graphical user interfaces in this modified project, and the only modifications we need to do are the coding parts for each form window. Basically, we need to perform the following modifications on the coding:

1. Add the Oracle namespace reference to the project.
2. Modify the connection string in the `LogIn Form`.
3. Modify the `SELECT` query string for the `LogIn` button's `Click` method in the `LogIn Form`.
4. Modify the `SELECT` query string for the `Select` button's `Click` method in the `Faculty Form`.
5. Modify the `UPDATE` query string for the `Update` button's `Click` method in the `Faculty Form`.
6. Modify the `DELETE` query string for the `Delete` button's `Click` method in the `Faculty Form`.
7. Modify the parameters' names for the `UPDATE` and the `DELETE` command objects in the `Faculty Form`.
8. Modify two `SELECT` query strings for the `Select` button's `Click` method and the `SelectedIndexChanged` method of the `Course` listbox in the `Course Form`.
9. Modify all prefixes for all Oracle classes and objects used in this project.

Well, it looks like we need to do many modifications for this project. However, it is easy to handle these operations. Let's begin our first modification.

7.7.1 Add Oracle Namespace Reference

Open our new project and go to the Solution Explorer window, right-click on the project, and select **Add Reference** item to open the **Add Reference** dialog box. Browse down along the list until you find the item **System.Data.OracleClient**, select it by clicking on it, and click on the **OK** button to add this reference to our project.

Open the code windows of the following forms from the current project:

- LogIn
- Faculty
- Course

Add the Oracle namespace statement using **System.Data.OracleClient**; to the namespace section for all the form windows. Since we will not use the Student, Insert Faculty, and the SP Forms for this project, leave them unchanged.

7.7.2 Modify Connection String and Query String for LogIn Form

The modifications to the LogIn Form can be divided into three parts: Modifications to the connection string in the constructor, modifications to the **SELECT** query string in the TableAdapter LogIn button's Click method, and modifications to the **SELECT** query string in the DataReader LogIn button's Click method.

7.7.2.1 Modify Connection String in Constructor of LogIn Class

Open the constructor of the LogIn Form class and change the connection string to:

```
string oraString = "Data Source=XE;" +
                  "User ID=CSE_DEPT;" + "Password=reback";
```

Also change the prefixes of all data classes from **Sql** to **Oracle**, the prefixes of all data objects from **sql** to **ora**, respectively, in this constructor.

7.7.2.2 Modify SELECT Query String in TabLogIn Button Click Method

The only modification we need to do to this method is to change the syntax of the query string and make it compatible with the query string used in the Oracle database. Open the TabLogIn button's Click method and change the **SELECT** query string to:

```
string cmdString
    = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
cmdString += "WHERE user_name=:name AND pass_word=:word";
```

Also change the prefixes of all data classes from **Sql** to **Oracle** and the prefixes of all data objects from **sql** to **ora**. Change two dynamic parameters' names from **@name** to **name** and from **@word** to **word** for the **Add()** method in the **Command's Parameters**

property, respectively. Also change the data type for those two dynamic parameters to `OracleType`.

7.7.2.3 *Modify SELECT Query String in ReadLogIn Button Click Method*

Open the `ReadLogIn` button's `Click` method and change the `SELECT` query string to:

```
string cmdString
    = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
cmdString += "WHERE user_name=:name AND pass_word=:word";
```

Also change the prefixes of all data classes from `Sql` to `Oracle` and the prefixes of all data objects from `sql` to `ora`. Change two dynamic parameters' names from `@name` to `name` and from `@word` to `word` for the `Add()` method in the `Command`'s `Parameters` property, respectively. Also change the data type for those two dynamic parameters to `OracleType`.

7.7.3 **Modify Query Strings in Faculty Form**

This modification can also be divided into three parts: Modifications to the query string for the `Select` button's `Click` method, modifications to the query string for the `Update` button's `Click` method, and modifications to the query string for the `Delete` button's `Click` method.

7.7.3.1 *Modify SELECT Query String for Select Button Click Method*

Open the `Faculty Form` window and the `Select` button's `Click` method, and change the query string to:

```
string cmdString = "SELECT faculty_id, faculty_name, office, phone,
                    college, title, email FROM Faculty ";
cmdString += "WHERE faculty_name =: FacultyName";
```

Also change the prefixes of all data classes from `Sql` to `Oracle` and the prefixes of all data objects from `sql` to `ora`. Change the dynamic parameter's name from `@FacultyName` to `FacultyName` for the `Add()` method in the `Command` `Parameter`'s property. Also change the data type for that dynamic parameter from `SqlDbType` to `OracleType`.

7.7.3.2 *Modify UPDATE Query String for Update Button Click Method*

Open the `Update` button's `Click` method and change the query string to:

```
string cmdString
    = "UPDATE Faculty SET faculty_name=:name, office=:office,
      phone=:phone, " + "college=:college, title=:title,
      email=:email " + "WHERE (faculty_name =: Param1)";
```

Change the prefixes of all data classes from `Sql` to `Oracle` and the prefixes of all data objects from `sql` to `ora`. Also modify the data types and the name of the dynamic parameters inside the `UpdateParameters()` method as below:

- Change the data type for all parameters from `SqlDbType` to `OracleType`.
- Remove the `@` symbol before each parameter's name.

7.7.3.3 *Modify DELETE Query String for Delete Button Click Method*

Open the Delete button's Click method and change the query string to:

```
string cmdString = "DELETE FROM Faculty WHERE (faculty_name
=: Param1)";
```

Change the prefixes of all data classes from `Sql` to `Oracle` and the prefixes of all data objects from `sql` to `ora`. Also change the dynamic parameter's name from `@Param1` to `Param1` and the data type from `SqlDbType` to `OracleType` for the `Add()` method in the Command Parameter's property.

7.7.4 *Modify Query Strings for Course Form*

The modification to this form can be divided into two parts: modifications to the query string for the Select button's Click method and modifications to the query string for the Course Listbox's `SelectedIndexChanged` method.

7.7.4.1 *Modify SELECT Query String for Select Button Click Method*

Open the Course Form window and the Select button's Click method and change the query string to:

```
string strCourse
= "SELECT Course.course_id, Course.course FROM Course, Faculty ";
strCourse += "WHERE (Course.faculty_id=Faculty.faculty_id) AND
(Faculty.faculty_name=:name)";
```

Change the prefixes of all data classes from `Sql` to `Oracle` and the prefixes of all data objects from `sql` to `ora`. Also change the dynamic parameter's name from `@name` to `name` and the data type from `SqlDbType` to `OracleType` for the `Add()` method in the Command Parameter's property.

Another modification is to change the data type of the nominal argument `CourseReader` from `SqlDataReader` to `OracleDataReader` in the user-defined method `FillCourseReader()`.

7.7.4.2 *Modify SELECT Query String for CourseList Click Method*

Open the Course Form window and the Course Listbox's `SelectedIndexChanged` method and change the query string to:

```
string cmdString = "SELECT course, credit, classroom, schedule,
enrollment, course_id FROM Course ";
cmdString += "WHERE course_id =: courseid";
```

Change the prefixes of all data classes from `Sql` to `Oracle` and the prefixes of all data objects from `sql` to `ora`. Also change the dynamic parameter's name from `@courseid` to `courseid` and the data type from `SqlDbType` to `OracleType` for the `Add()` method in the Command Parameter's property. Another modification is to change the data type of the nominal argument `CourseReader` from the `SqlDataReader` to the `OracleDataReader` in the user-defined method `FillCourseReaderTextBox()`.

7.7.5 Other Modifications

Change the prefixes of all data classes from `Sql` to `Oracle` and the prefixes of all data objects from `sql` to `ora`. Change the data type from `SqlDbType` to `OracleType` and `DataReader` from `SqlDataReader` to `OracleDataReader`. These modifications include the following methods:

- The Cancel button's Click method in the LogIn Form
- The user-defined method `UpdateFaculty()` in the Faculty Form
- The user-defined method `FillFacultyReader()` in the Faculty Form
- The constructor of the Course Form
- The constructor of the Insert Faculty Form
- The Exit button's Click method in the Selection Form

After completing the modifications listed above, we may still encounter some other compiling errors as we build the project. These errors may include:

- Mismatched Connection object inside the Select button's Click method in the Insert Faculty Form window
- Mismatched Connection object inside the user-defined method `BuildCommand()` in the Student Form window
- Mismatched Connection object inside the user-defined method `BuildCommand()` in the SP Form window

Just comment out these instructions to avoid the related compiling errors since we will not use those methods in this project. In addition to commenting out these compiling error instructions, you also need to comment out the following two instructions since (1) they will be executed as a new instance of either the Student Form or the SP Form class, and (2) we will not use these two forms in this project:

- Calling of the Select button's Click method in the constructor of the Student Form class: `cmdSelect_Click(this.cmdSelect, null);`
- Calling of the Select button's Click method in the constructor of the SP Form class: `cmdSelect_Click(this.cmdSelect, null);`

At this point, we have finished all modifications to the project, and now we can run the project to test the data updating and deleting functions. Click on the Start Debugging button to run the project. Enter the suitable username and password such as `jhenry` and test to the LogIn Form, and select the item Faculty Information from the Selection Form to open the Faculty Form window.

To test the data updating, first let's select a faculty member such as **Ying Bai** from the combobox `ComboName`, and click on the Select button to retrieve all six pieces of information related to that selected faculty and display them in this form. Then update this faculty by changing the following information in the associated textbox:

- Peter Bai Faculty Name textbox
- Research Professor Title textbox
- MTC-335 Office textbox
- 750-330-5555 Phone textbox
- pbai@college.edu Email textbox

The screenshot shows a window titled "CSE DEPT Faculty Form". It contains a "Method" dropdown set to "DataAdapter Method". Below it is a photo of a man. To the right, there is a "Faculty Name" dropdown set to "Peter Bai". Below that is a "Faculty Information" section with several textboxes: "Faculty Name" (Peter Bai), "Title" (Research Professor), "Office" (MTC-335), "Phone" (750-330-5555), "College" (Florida Atlantic University), and "Email" (pbai@college.edu). At the bottom, there are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 7.19 Confirmation of the data updating operation.

Click on the Update button to update this record in the Faculty table in the database.

To confirm this data updating, click on the drop-down arrow on the combobox control ComboName. First, we can select any other faculty from the list and click on the Select button to show the information for that faculty. Then select the updated faculty (Peter Bai) from the combobox control ComboName and click on the Select button to try to retrieve this updated faculty information and display it in this form. Immediately you can find that the selected faculty information has been updated and displayed, which is shown in Figure 7.19. Our data updating is successful.

Now let's test our data deleting function. Keep the updated faculty name unchanged in the combobox control ComboName and click on the Delete button to try to delete it from the Faculty table in the database. Click on Yes to the confirmation message box, and then you can find that all information related to the deleted faculty is removed from all textboxes. To confirm that data deleting, click on the Select button to try to retrieve the deleted record from the Faculty table. A message "No matched faculty found" is displayed to indicate that the selected faculty and the related information have been deleted from the database. Yes, our data deleting is also successful.

Before we can finish this section, it is highly recommended to recover the deleted faculty information involved in the Faculty, LogIn, Course, and the StudentCourse tables in our sample Oracle database CSE_DEPT. Open the sample Oracle database and refer to Section 7.6.2.2 in this Chapter to complete these data recoveries.

A complete project OracleUpdateDeleteRTOObject can be found from the folder DBProjects\Chapter 7 located in the accompanying ftp site (see Chapter 1).

7.8 UPDATE AND DELETE DATA IN DATABASE USING STORED PROCEDURES

As we mentioned in the previous sections, performing the data updating among related tables is a very challenging topic. But the good news is that most of the time it is unneces-

sary to update the primary key, or the `faculty_id`, in our Faculty table if we want to update any faculty information from the Faculty table in the database. Basically, it is much better to insert a new faculty record with a new `faculty_id` into the Faculty table than updating that record including the primary key `faculty_id` because the primary key or `faculty_id` is good for the lifetime of the database in actual applications. Therefore, based on the analysis above, we will perform the data updating for all columns in the Faculty table except the `faculty_id` in this section.

To delete records from related tables, we need to perform two steps: First, we need to delete records from the child tables, and then we can delete those records from the parent table. For example, if we want to delete a record from the Faculty table, first we need to delete those records that are related to the record to be deleted from the Faculty table from the LogIn and the Course tables (child tables), and then we can delete the record from the Faculty table (parent table).

We divide this discussion into three parts based on three types of databases we used in this book: using stored procedures to update and delete data in (1) Microsoft Access 2007 database, (2) SQL Server database, and (3) Oracle database.

To save time and space, we will not duplicate any project, and we want to modify some existing projects to create our desired projects.

7.8.1 Update and Delete Data in Access Database Using Stored Procedures

We want to modify the project `SQLUpdateDeleteRTOObject` to create our desired project `AccessUpdateDeleteSP` to discuss the data updating and deleting in the Faculty table using the stored procedures for the Microsoft Access database.

Perform the following tasks to finish this project.

1. Modify the existing project to access the Microsoft Access database.
2. Create stored procedures in the Microsoft Access database.
3. Call the stored procedure to update and delete the faculty information.
4. Confirm the updated and deleted faculty information.

Now let's start from the first part to modify the project.

7.8.1.1 Modify Existing Project

Open Windows Explorer and create a new folder such as Chapter 7 if you have not, and then browse to the folder `DBProjects\Chapter 7` located at the accompanying ftp site (see Chapter 1). Copy the project `SQLUpdateDeleteRTOObject` to the new folder Chapter 7. Change the names of the solution and the project to `AccessUpdateDeleteSP`. Double-click on the `AccessUpdateDeleteSP.csproj` to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to Project! `AccessUpdateDeleteSP` Properties menu item to open the project's property window. Change the `Assembly name` and the `Default namespace` from `SQLUpdateDeleteRTOObject` to `AccessUpdateDeleteSP`, respectively.

- Click on the **Assembly Information** button to open the **Assembly Information** dialog box, change the **Title** and the **Product** to **AccessUpdateDeleteSP**. Click on **OK** to close this dialog box.
- Change the project namespace for all files from **SQLUpdateDeleteRTObject** to **AccessUpdateDeleteSP** using the **Find and Replace** dialog box.

Go to the **File|Save All** to save those modifications. Now we are ready to modify our codes based on our new project **AccessUpdateDeleteSP**.

The code modifications include the following parts:

1. Add the **OleDb Data Provider** namespace and change the connection string in the **LogIn Form**.
2. Change the query strings for the **LogIn** button's **Click** method in the **LogIn Form**.
3. Change the query strings for the **Select**, **Update**, and **Delete** buttons' **Click** methods in the **Faculty Form**.
4. Change the prefixes of all data classes from **Sql** to **OleDb** and the prefixes of all data objects from **sql** to **acc** for the **LogIn**, **Faculty**, and **Selection Forms**.
5. Other modifications.

Let's start from the first modification—modify the **Imports** commands

7.8.1.1.1 Modify Namespaces and Connection String Open the code window of the **LogIn Form** class and add the **OleDb Data Provider** namespace to the namespace section of that form by entering the following statement:

```
using System.Data.OleDb;
```

Perform the same namespace addition to all other forms in this new project.

Open the constructor of the **LogIn Form** class and change the connection string to:

```
string accString = "Provider=Microsoft.ACE.OLEDB.12.0;" +  
                  "Data Source=C:\\database\\Access\\CSE_DEPT.accdb;"
```

Also change the prefixes of all data classes from **Sql** to **OleDb** and the prefixes of all data objects from **sql** to **acc** in the constructor.

7.8.1.1.2 Modify the Query Strings in the LogIn Button Click Method There are two query strings located at two different **LogIn** buttons' **Click** methods: the **TabLogIn** and the **ReadLogIn**. Open these two methods and modify these two query strings. This modification is very easy and the only change is to remove the **@** symbol before each dynamic parameter in the **WHERE** clause and in the **Add()** method in the **Parameters** property of the **Command** class. An example of this modification is to change the dynamic parameters **@name** to **name** and **@word** to **word**, respectively, for the **TabLogIn** query string and the **Add()** method of the **Parameters** property. Another modification is to change the data type of the dynamic parameters in the **Add()** method of the **Parameters** property of the **Command** class from **SqlDbType** to **OleDbType**. Perform this modification to two query strings. Also change the prefixes of all data classes from **Sql** to **OleDb** and the prefixes of all data objects from **sql** to **acc** in these two methods.

7.8.1.1.3 Modify Query Strings in Select, Update, and Delete Buttons Click Methods Open the **Faculty Form** window and these three buttons' **Click** methods one

by one to modify each query string. For the Select button's Click method, replace the symbol LIKE @ with the equal symbol = in the WHERE clause for the query string. Remove the @ symbol before the dynamic parameter (@name) in the Add() method in the Parameters property of the Command class. Also change the data type of the dynamic parameters in the Add() method of the Parameters property of the Command class from SqlDbType to OleDbType.

For the Update button's Click method, replace the symbol LIKE @ with the equal symbol = in the WHERE clause for the query string. Also Remove the @ symbol before each updating dynamic parameter in the updating query string. Another modification is to change the data type of each dynamic parameter in the Add() method of the Parameters property of the Command class in the user-defined method UpdateParameters() from SqlDbType to OleDbType, and remove the @ symbol from each dynamic parameter in that method. Finally change the data type of the passed Command object from SqlCommand to OleDbCommand in the user-defined method UpdateParameters().

For the Delete button's Click method, replace the symbol LIKE @ with the equal symbol = in the WHERE clause for the query string. Remove the @ symbol before the dynamic parameter (@Param1) in the Add() method of the Parameters property of the Command class. Also change the data type of the dynamic parameters in the Add() method of the Parameters property of the Command class from SqlDbType to OleDbType.

Change the prefixes of all data classes from Sql to OleDb and the prefixes of all data objects from sql to acc for these three methods. Another modification to this form is to change the data type of the passed argument FacultyReader from the SqlDataReader to the OleDbDataReader in the user-defined method FillFacultyReader().

7.8.1.1.4 Other Modifications Change the prefixes of all data classes from Sql to OleDb and the prefixes of all data object from sql to acc for the following methods:

- Cancel button's Click method in the LogIn Form
- User-defined method UpdateFaculty() in the Faculty Form
- Exit button's Click method in the Selection Form

Because we will not use other forms in this project such as the Course, Student, Insert Faculty Form, and the SP Forms, we do not need to make modifications to these forms. One possible problem is that you may encounter some compiling errors when you build this project because of some unmodified codes in these forms. To solve this problem, just comment out those codes that have not been modified. Also comment out the following statement inside the constructors of the SP Form and the Student Form classes since we will not uses these two forms:

```
cmdSelect_Click(this.cmdSelect, null);
```

Now let's create our stored procedure in the Microsoft Access 2007 database.

7.8.1.2 Create Stored Procedures in Microsoft Access Database

As we mentioned at the beginning of this section, the data updating operation updates all columns of one existing faculty record except the `faculty_id` column since it is

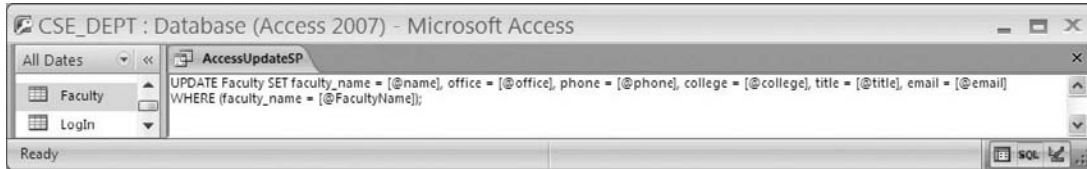


Figure 7.20 Stored procedure in Microsoft Access database.

unnecessary to update a primary key from the Faculty table. A better way to update a `faculty_id` is to insert a new faculty record with a new `faculty_id`, which is a common way used in most real data-driven applications.

First, let's create the updating stored procedure to update one faculty record. Open our sample database `CSE_DEPT.accdb`, which can be found from the folder `Database\Access` located at the accompanying ftp site (see Chapter 1). You had better copy that database and paste it in a folder in your root drive, such as `C:\database`.

On the opened database, select the Faculty table from the list and click on the **Create** tab from the menu tab, then click on the **Query Design** button to open the Query Builder dialog box. Then click on the **Close** button to close the **Show Table** dialog box.

Right-click on the top pane and select the item **SQL View** from the popup menu to open the SQL window, which is shown in Figure 7.20. Enter the Update statement that is shown below into this SQL View window as our stored procedure:

```
UPDATE Faculty SET faculty_name = [@name], office = [@office],
    phone = [@phone],
    college = [@college], title = [@title], email = [@email]
WHERE (faculty_name = [@FacultyName]);
```

Your finished statement of the updating stored procedure should match the one shown in Figure 7.20.

Click on the **Save** button to save this stored procedure as `AccessUpdateSP`. To confirm this stored procedure, we can run this query inside the Access 2007 environment. Right-click on the new created stored procedure `AccessUpdateSP` from the list and select the **Open** item from the pop-up menu to begin to run this query.

Enter `Ying Bai` into the `@FacultyName` input box to open this record from our Faculty table, and then update this faculty by entering the following values into the associated columns for this record:

- Peter Bai faculty_name column
- MTC-228 office column
- 750-378-1220 phone column
- University of Miami college column
- Associate Professor title column
- pbai@college.edu email column

After finish entering these updated values, double-click on the Faculty table from the left pane to open it to confirm our data updating. You can find that the old record for the faculty named `Ying Bai` has been replaced by our updated record, which is shown in Figure 7.21.

faculty_id	faculty_name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.edu
A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@college.edu
B78880	Peter Bai	MTC-228	750-378-1220	University of Miami	Associate Professor	pbai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	iking@college.edu

Figure 7.21 Confirmation of the data updating.

Now recover this record to the original one for this faculty with the following information since we want to keep our data unchanged:

- Ying Bai faculty_name column
- MTC-211 office column
- 750-378-1148 phone column
- Florida Atlantic University college column
- Associate Professor title column
- ybai@college.edu email column

In a similar way, we can create our deleting stored procedure AccessDeleteSP. Open the SQL View window and enter the following statement as the content of our deleting stored procedure:

```
DELETE FROM Faculty
WHERE (faculty_name = [@FacultyName]);
```

Click on the Save button to save this stored procedure as AccessDeleteSP.

At this point, we have finished creating our data updating and deleting stored procedures in Microsoft Access 2007 database. Close our sample database by exiting the Access 2007. Now let's develop the codes in Visual C#.NET to call these stored procedures to perform the data updating and deleting actions against the database.

7.8.1.3 Call Stored Procedure to Update Faculty Information

Open the Faculty Form and its Update button's Click method, and add the codes shown in Figure 7.22 into this method.

Only three modifications are made to this method and the user-defined method UpdateParameters(). All three modifications have been highlighted in bold. Let's take a close look at these modifications.

- The content of the query string now should be equal to the name of the stored procedure we developed in the last section since we need to call it to perform the data updating action. This name must be identical with the name we used when we developed this stored

```

AccessUpdateDeleteSP.FacultyForm  cmdUpdate_Click()
private void cmdUpdate_Click(object sender, EventArgs e)
{
A   string cmdString = "AccessUpdateSP";
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    OleDbCommand accCommand = new OleDbCommand();
    int intUpdate = 0;

B   accCommand.Connection = logForm.accConnection;
    accCommand.CommandType = CommandType.StoredProcedure;
    accCommand.CommandText = cmdString;
    UpdateParameters(ref accCommand);
    intUpdate = accCommand.ExecuteNonQuery();
    accCommand.Dispose();
    ComboName.Items.Clear();
    UpdateFaculty();
    if (intUpdate == 0)
        MessageBox.Show("The data updating is failed");
}
private void UpdateParameters(ref OleDbCommand cmd)
{
C   cmd.Parameters.Add("name", OleDbType.Char).Value = txtName.Text;
    cmd.Parameters.Add("office", OleDbType.Char).Value = txtOffice.Text;
    cmd.Parameters.Add("phone", OleDbType.Char).Value = txtPhone.Text;
    cmd.Parameters.Add("college", OleDbType.Char).Value = txtCollege.Text;
    cmd.Parameters.Add("title", OleDbType.Char).Value = txtTitle.Text;
    cmd.Parameters.Add("email", OleDbType.Char).Value = txtEmail.Text;
    cmd.Parameters.Add("FacultyName", OleDbType.Char).Value = ComboName.Text;
}

```

Figure 7.22 Coding for the Update button's Click method.

procedure in the Microsoft Access 2007 database. Otherwise, the project cannot find the stored procedure as the project runs.

- B.** When the Command object was initialized and built, the CommandType property should have been to StoredProcedure to tell the project that the query to be executed is a stored procedure not a normal query. Also the name of the stored procedure should be assigned to the CommandText property to allow the project to locate the stored procedure as the project runs.
- C.** The nominal dynamic parameter, which is an existing faculty name before the data updating and located in the Add() method of the Parameters property, should be equal to the name of the dynamic parameter we used in our stored procedure. Refer to Section 7.8.1.2 to make sure that both dynamic parameters are identical.

Now we have finished all coding for the data updating using the stored procedures in the Microsoft Access database. Before we can run the project to test this data updating function, we prefer to complete the coding for the data deleting action for our sample database. In that way, we can run the project to perform both the data updating and the data deleting functions at the same time.

7.8.1.4 Call Stored Procedure to Delete Faculty Information

Open the Faculty Form and its Delete button's Click method, and add the codes shown in Figure 7.23 into this method.

```

AccessUpdateDeleteSP.FacultyForm  cmdDelete_Click()
private void cmdDelete_Click(object sender, EventArgs e)
{
A   string cmdString = "AccessDeleteSP";
    MessageBoxButtons vbButton = MessageBoxButtons.YesNo;
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    OleDbCommand accCommand = new OleDbCommand();
    DialogResult Answer;
    int intDelete = 0;
    Answer = MessageBox.Show("Do you want to delete this record?", "Delete", vbButton);
    if (Answer == System.Windows.Forms.DialogResult.Yes)
    {
B       accCommand.Connection = logForm.accConnection;
        accCommand.CommandType = CommandType.StoredProcedure;
        accCommand.CommandText = cmdString;
C       accCommand.Parameters.Add("FacultyName", OleDbType.Char).Value = ComboName.Text;
        intDelete = accCommand.ExecuteNonQuery();
        accCommand.Dispose();

        if (intDelete == 0)
            MessageBox.Show("The data Deleting is failed");

        for (intDelete = 0; intDelete < 7; intDelete++) // clean up the Faculty textbox array
            FacultyTextBox[intDelete].Text = string.Empty;
    }
}

```

Figure 7.23 Coding for the Delete button's Click method.

Let's take a close look at these modifications.

- A.** The content of the query string now should be equal to the name of the stored procedure we developed in the last section since we need to call it to perform the data deleting action. This name must be identical with the name we used when we developed the data deleting stored procedure in the Microsoft Access 2007 database. Otherwise, the project cannot find the stored procedure as the project runs, and a running time error will be encountered.
- B.** When the Command object was initialized and built, the CommandType property should have been set to StoredProcedure to tell the project that the query to be executed is a stored procedure not a normal query. Also the name of the stored procedure should be assigned to the CommandText property to allow the project to locate the stored procedure as the project runs.
- C.** The nominal dynamic parameter, which is an existing faculty name before the data updating and located in the Add() method of the Parameters property, should be equal to the name of the dynamic parameter we used in our stored procedure. Refer to Section 7.8.1.2 to make sure that both dynamic parameters are identical.

Now we can run the project to test the stored procedures to perform the data updating and deleting actions in our sample database. Click on the Start Debugging button to start our project. Enter the suitable username and password to the LogIn Form and select the Faculty Information item from the Selection Form window to open the Faculty Form. Select the faculty member Ying Bai from the combobox control and click on the Select button to display the information for the selected faculty.

Figure 7.24 Running status of the data validation process.

To update this faculty information, enter the following information into the associated textboxes to perform this data updating:

- Distinguished Professor Title textbox
- MTC-228 Office textbox
- 750-378-1220 Phone textbox
- University of Main College textbox

Click on the Update button to call the stored procedure to update this faculty information in the Faculty table in our sample database.

To confirm this updating, first let's click on the drop-down arrow of the combobox control ComboName and select any other faculty from the box. Then click on the Select button to display the information related to that selected faculty. Then re-open the combobox control ComboName and select our new updated faculty Taylor Bai from the box. Click on the Select button to retrieve that updated faculty information from the database and display it in this form. Immediately you can find that the updated faculty information is returned and displayed, as shown in Figure 7.24. Our data updating action using the stored procedure is successful.

To test the data deleting action, keep the updated faculty Taylor Bai selected at the ComboName box and click on the Delete button. Click on Yes to the MessageBox to try to delete this faculty record from our sample database. Immediately, all textboxes that stored the related faculty information are clean up. To confirm this data deleting, click on the Select button to try to retrieve the deleted faculty record and display it in this form. A warning message “**No matched faculty found!**” is displayed, and this means that the selected faculty has been successfully deleted from our sample database and our data deleting action is successful!

In order to keep our database neat and complete, we need to recover that deleted faculty record. Remember, to delete a record from the parent table, all records related to that record in the child tables should be deleted first. Two child tables, LogIn and Course, are related to the faculty member to be deleted from the Faculty table, and the

other child table, StudentCourse, is related to the course_id to be deleted from the Course table. Therefore we have four tables to recover. Open our sample database CSE_DEPT.acddb and refer to Section 7.6.2.2 to recover the deleted faculty information for four tables. One point to be noted when recovering the LogIn table is that you need to remove the NULL item in the student_id column for the recovered faculty login record because the Access 2007 database uses a blank column as a NULL column.

A complete project AccessUpdateDeleteSP can be found from the folder DBProjects\Chapter 7 located at the accompanying ftp site (see Chapter 1).

7.8.2 Update and Delete Data in SQL Server Database Using Stored Procedures

To update and delete data using stored procedures developed in the SQL Server database is very similar to the data updating and deleting we performed in the last section. With a small modification to the existing project SQLUpdateDeleteRTOObject, we can easily create our new project SQLUpdateDeleteSP to perform the data updating and deleting by calling stored procedures developed in the SQL Server database.

To develop our new project in this section, we divide it into three sections:

1. Modify the existing project SQLUpdateDeleteRTOObject to create our new project SQLUpdateDeleteSP.
2. Develop the data updating and deleting stored procedures in the SQL Server database.
3. Call stored procedures to perform the data updating and deleting for the faculty information using the Faculty Form window.

Now let's start with the first step.

7.8.2.1 Modify Existing Project to Create New Project

Open Windows Explorer and create a new folder such as Chapter 7 if you have not, and then browse to the folder DBProjects\Chapter 7 located in the accompanying ftp site (see Chapter 1). Copy the project SQLUpdateDeleteRTOObject to the new folder C:\Chapter 7. Change the name of the project from SQLUpdateDeleteRTOObject to SQLUpdateDeleteSP. Double-click on the SQLUpdateDeleteSP.csproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to Project\SQLUpdateDeleteSP Properties menu item to open the project's property window. Change the Assembly name and the Default namespace from SQLUpdateDeleteRTOObject to SQLUpdateDeleteSP, respectively.
- Click the Assembly Information button to open the Assembly Information dialog box, change the Title and the Product to SQLUpdateDeleteSP. Click on OK to close this dialog box.
- Change the project namespace for all files from SQLUpdateDeleteRTOObject to SQLUpdateDeleteSP using the Find and Replace dialog box.

Go to the File|Save All to save these modifications. Now we are ready to modify our codes based on our new project SQLUpdateDeleteSP.

The code modifications include the following parts:

1. Replace the query string in the Update button's Click method in the Faculty Form with the name of the data updating stored procedure that will be developed in the next section to allow the method to call the related stored procedure to perform the data updating action.
2. Replace the query string in the Delete button's Click method in the Faculty Form with the name of the data deleting stored procedure that will be developed in the next section to allow the method to call the related stored procedure to perform the data deleting action.

Normally, these two modifications should be performed after the stored procedure has been created in the SQL Server database since we need some information from the created stored procedure to execute these modifications, such as the name of the stored procedure and the names of the input parameters to the stored procedure. Because of the similarity between this project and the last one, we assumed we knew those pieces of information and we can put those pieces of information into these two methods in advance. This assumed information includes:

1. The name of the data updating stored procedure—`dbo.UpdateFacultySP`.
2. The names of the input parameters—identical with the columns' names in the Faculty table in our sample database.
3. The name of the input dynamic parameter—`@FacultyName`.
4. The name of the data deleting stored procedure—`dbo.DeleteFacultySP`.
5. The names of the input parameters—identical with the columns' names in the Faculty table in our sample database.
6. The name of the input dynamic parameter—`@FacultyName`.

Based on these assumptions, we can first modify our coding in the Update button's Click method. The key point is that we need to remember the names of these parameters and the name of the stored procedure and put them into our stored procedure later when we developed it in the next section.

Open the Update button's Click method and modify its coding. Your finished modifications to this method should match those codes shown in Figure 7.25.

The modified parts have been highlighted in bold. Let's see how this piece of modified code works.

- A. The content of the query string now should be equal to the name of the stored procedure.
- B. The `CommandType` property of the `Command` object should be set to `StoredProcedure` to tell the project that a stored procedure should be called as the project runs to perform the data updating job.

The modifications to the user-defined `UpdateParameters()` method is shown in Figure 7.26.

The only modification is to change the name of the dynamic parameter from `@Param1` to `@FacultyName` since we must keep all names of the input parameters to the stored procedure identical with those parameters we used in our codes in this method.

Next let's modify the codes in the Delete button's Click method. Open this method and perform the following modifications to this method. Your finished method should match that shown in Figure 7.27.

The modified parts have been highlighted in bold. Let's see how this piece of modified code works.


```

SQLUpdateDeleteSP.FacultyForm | cmdUpdate_Click()
private void cmdUpdate_Click(object sender, EventArgs e)
{
A   string cmdString = "dbo.UpdateFacultySP";
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    SqlCommand sqlCommand = new SqlCommand();
    int intUpdate = 0;

B   sqlCommand.Connection = logForm.sqlConnection;
    sqlCommand.CommandType = CommandType.StoredProcedure;
    sqlCommand.CommandText = cmdString;
    UpdateParameters(ref sqlCommand);
    intUpdate = sqlCommand.ExecuteNonQuery();
    sqlCommand.Dispose();
    ComboName.Items.Clear();
    UpdateFaculty();
    if (intUpdate == 0)
        MessageBox.Show("The data updating is failed");
}

```

Figure 7.25 Modified codes for the Update button's Click method.

```

SQLUpdateDeleteSP.FacultyForm | UpdateParameters()
private void UpdateParameters(ref SqlCommand cmd)
{
    cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text;
    cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text;
    cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text;
    cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text;
    cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text;
    cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text;
    cmd.Parameters.Add("@FacultyName", SqlDbType.Char).Value = ComboName.Text;
}

```

Figure 7.26 Modified codes for the UpdateParameters method.

- A. The content of the query string now should be equal to the name of the stored procedure.
- B. The CommandType property of the Command object should be set to StoredProcedure to tell the project that a stored procedure should be called as the project runs to perform the data updating job.
- C. The name of the dynamic parameter, which is located in the Add() method of the Parameters property, @Param1, has been changed to @FacultyName since we must keep all names of the input parameters to the stored procedure identical with those parameters we used in this method.

Now we have finished all coding modifications in the Visual C#.NET environment. Let's start to create our stored procedures in the SQL Server database. There are two ways you can create a stored procedure: one way is to create it in the SQL Server Management Studio Express, and the other way is to create it in the Server Explorer in the Visual Studio.NET environment. Since we are working on a Visual C#.NET project, we prefer to use the second method to create our stored procedure.


```

SQLUpdateDeleteSP.FacultyForm  cmdDelete_Click()
private void cmdDelete_Click(object sender, EventArgs e)
{
A   string cmdString = "dbo.DeleteFacultySP";
    MessageBoxButtons vbButton = MessageBoxButtons.YesNo;
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    SqlCommand sqlCommand = new SqlCommand();
    DialogResult Answer;
    int intDelete = 0;
    Answer = MessageBox.Show("Do you want to delete this record?", "Delete", vbButton);
    if (Answer == System.Windows.Forms.DialogResult.Yes)
    {
B       sqlCommand.Connection = logForm.sqlConnection;
        sqlCommand.CommandType = CommandType.StoredProcedure;
C       sqlCommand.CommandText = cmdString;
        sqlCommand.Parameters.Add("@FacultyName", SqlDbType.Char).Value = ComboName.Text;
        intDelete = sqlCommand.ExecuteNonQuery();
        sqlCommand.Dispose();
        if (intDelete == 0)
            MessageBox.Show("The data Deleting is failed");
        for (intDelete = 0; intDelete < 7; intDelete++) // clean up the Faculty textbox array
            FacultyTextBox[intDelete].Text = string.Empty;
    }
}

```

Figure 7.27 Modified codes for the Delete button's Click method.

7.8.2.2 Develop Stored Procedure in SQL Server Database

Open the Server Explorer in the Visual Studio environment, and click on the small plus icon before our sample database CSE_DEPT.mdf to expand it. Then right-click on the Stored Procedures folder and select the item Add New Stored Procedure to open the default procedure window. Change the name of the default stored procedure from `dbo.StoredProcedure1` to `dbo.UpdateFacultySP`, which is identical with the name of the stored procedure we used in our coding in the last section. Then add the codes that are shown in Figure 7.28 into this stored procedure as the body of our new stored procedure.

Refer to Section 2.10.2 in Chapter 2 for the data types of those input parameters, and the data types of those input parameters should be identical with those data types of the associated columns defined in the Faculty table. Go to the menu item File|Save StoredProcedure1 to save our new stored procedure.

To test our stored procedure, right-click on our new created stored procedure `UpdateFacultySP`, which is located under the Stored Procedure folder, and select the item Execute from the pop-up menu to open the Run Stored Procedure dialog box. Enter the following updated information into each field on the Value column of this dialog box:

- Tailor Bai Name value
- MTC-228 Office value
- 750-378-1221 Phone value
- University of Miami College value
- Associate Professor Title value

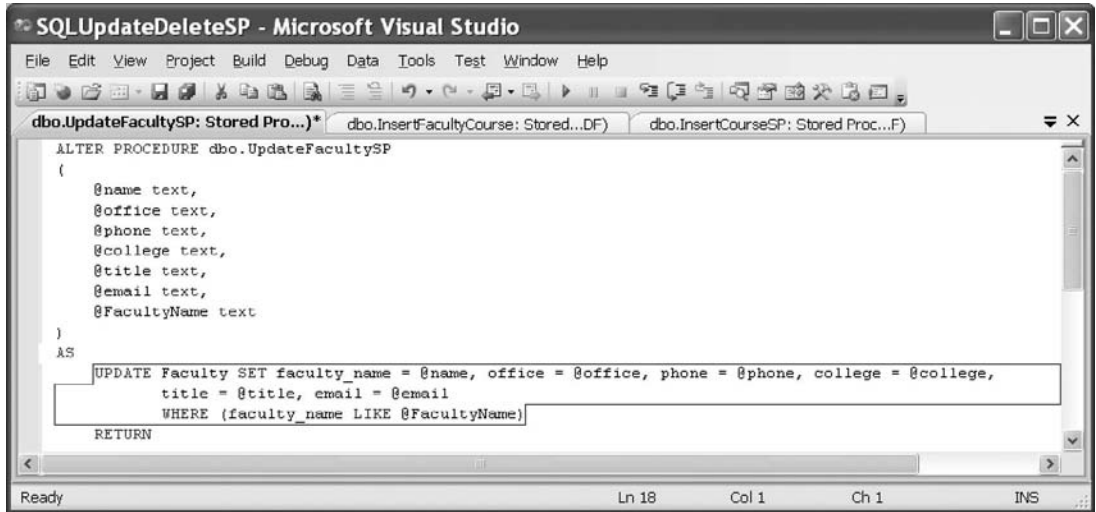


Figure 7.28 Created data updating stored procedure.

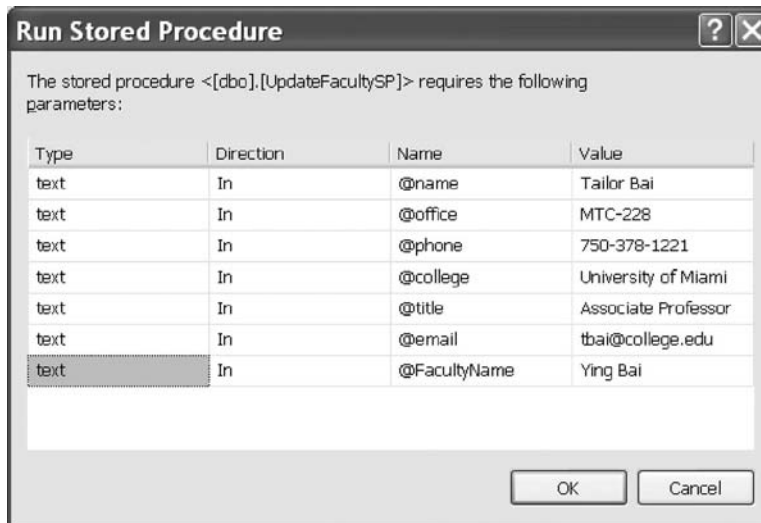


Figure 7.29 Finished Run Stored Procedure dialog box.

- tbai@college.edu Email value
- Ying Bai FacultyName value

Your finished information dialog box should match the one shown in Figure 7.29.

Click on the OK button to run this stored procedure. To confirm the execution of this stored procedure, you can open the Faculty table to check it. Go to the Server Explorer window, expand the Tables folder, and right-click on the Faculty table; then

The screenshot shows the Microsoft Visual Studio SQL UpdateDeleteSP window. The table 'Faculty' is displayed with the following data:

faculty_id	faculty_name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.edu
A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@college.edu
B78880	Tailor Bai	MTC-228	750-378-1221	University of Miami	Associate Professor	tbai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Profe...	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

Figure 7.30 Updated Faculty table.

select Show Table Data to open the Faculty table. You can find that our updated record is in there, which is shown as a highlighted row in Figure 7.30. Our stored procedure is successful.

In order to keep our database neat, we prefer to recover this updated faculty record with the original data. To do that, enter the following information into this updated row in the Faculty table to recover it:

- Ying Bai Name column
- MTC-211 Office column
- 750-378-1148 Phone column
- Florida Atlantic University College column
- Associate Professor Title column
- ybai@college.edu Email column

Save and close the database. Next let's create our data deleting stored procedure `dbo.DeleteFacultySP` in the SQL Server database.

In the opened Server Explorer window, right-click on the Stored Procedures folder and select the item Add New Stored Procedure to open the default procedure window. Change the name of the default stored procedure from `dbo.StoredProcedure2` to `dbo.DeleteFacultySP`, which is identical with the name of the stored procedure we used in our coding in the last section. Then add the codes shown in Figure 7.31 into this stored procedure as the body of our new data deleting stored procedure.

Go to the menu item File!Save StoredProcedure2 to save our new stored procedure. To test our stored procedure, right-click on our new created stored procedure `DeleteFacultySP`, which is located under the Stored Procedure folder, and select the item Execute from the pop-up menu to open the Run Stored Procedure dialog box. Enter the faculty name Ying Bai into the Value column of this dialog box, which is shown in Figure 7.32.

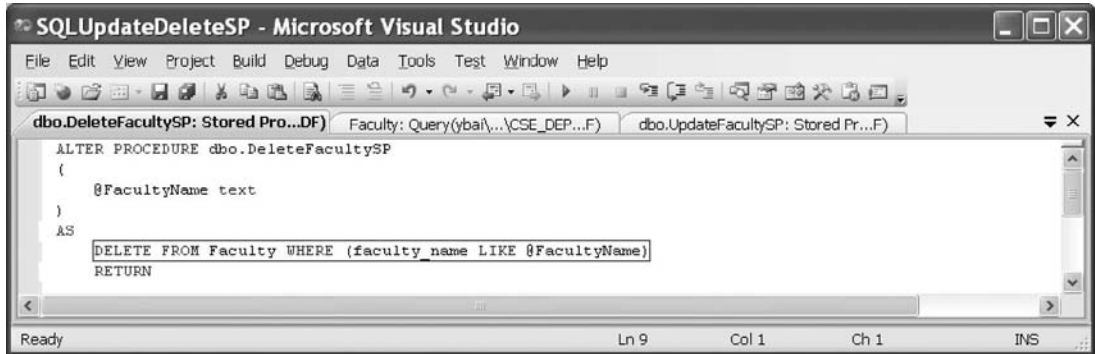


Figure 7.31 Created data deleting stored procedure.

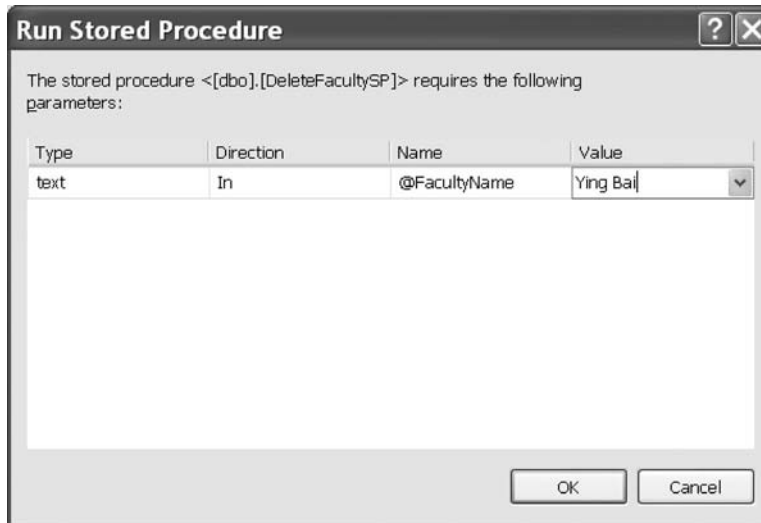


Figure 7.32 Finished Run Stored Procedure dialog box.

Click on the OK button to run this stored procedure. To confirm the execution of this stored procedure, you can open the Faculty table from the Server Explorer window to check it. When checking this data deleting action against the Faculty table in our sample database, remember that you need first close the Visual Studio.NET to disconnect the connection between our project and our sample database, and then reopen our project and the Server Explorer to perform that deleting confirmation. Otherwise the deleted data cannot be reflected in our sample database. Our stored procedure is successful.

In order to keep our database neat, we prefer to recover these deleted faculty records with the original data. Recall in Section 7.6.2.2, when a faculty member is deleted from the Faculty table (parent table), all records related to that faculty member in the Course and LogIn tables (child tables) will also be deleted. Similarly, as a course is deleted from the Course table (parent table), all records related to that course in the StudentCourse table (child table) will be deleted, too. In total there are 11 records deleted from four tables:

- One faculty record from the Faculty table (parent table)
- One login record from the LogIn table (child table)
- Four course records from the Course table (child table)
- Five student course records from the StudentCourse table (child table)

Open those tables in the Server Explorer window and add those deleted records to each associated table one by one. Refer to Section 7.6.2.2 to complete this data recovery. You can use the copy/paste functions to first copy all rows from each table in Section 7.6.2.2, and then paste them at the end of each table in our sample database.

Another important point in recovering these deleted records is the order in which you performed that copy/paste actions. You must first recover the faculty member deleted from the parent table, Faculty table, and then you can recover all other related records in all other child tables. The reason for this is that the Faculty is a parent table with the `faculty_id` as a primary key, you cannot recover any other record without first recovering the deleted record from the parent table. Click on **File|Save All** menu item when you finished these recoveries to save those recovered records.

Now that we have built our stored procedures, let's call these stored procedures from our Visual C#.NET project to test the data updating and deleting functions.

7.8.2.3 Call Stored Procedure to Perform Data Updating and Deleting

Start our project by clicking on the Start Debugging button, enter the suitable username and password to the LogIn Form, and then select the Faculty Information item from the Selection Form to open the Faculty Form window. Select the faculty name Ying Bai from the combobox control, and click on the Select button to display the information for the selected faculty.

To update this faculty information, enter the following data into the associated textboxes to perform this data updating:

- | | |
|---------------------------|----------------------|
| • Peter Bai | Faculty Name textbox |
| • Distinguished Professor | Title textbox |
| • MTC-228 | Office textbox |
| • 750-378-1222 | Phone textbox |
| • University of Main | College textbox |
| • pbai@college.edu | Email textbox |

Click on the Update button to call the stored procedure to update this faculty information in the Faculty table in the database.

To confirm this data updating, let's click on the drop-down arrow of the combobox control `ComboName` to try to find and select our updated faculty name **Peter Bai** from the box. Click on the Select button to retrieve that updated faculty information from the database and display it in this form. Click on OK to the pop-up `MessageBox` since we did not include a updated faculty image for this data updating. Immediately you will find that the updated faculty information is returned and displayed, which is shown in Figure 7.33.



Figure 7.33 Confirmation of the data updating.

Now let's test the data deleting action by clicking on the Delete button to try to delete this updated record. Click on Yes to the MessageBox to perform this deleting. Immediately all information stored in textboxes is removed. To confirm this data deleting, click on the Select button to try to retrieve the deleted faculty information. A warning message "No matched faculty found" is displayed, which means that the selected faculty member has been deleted from our database. Our data updating and deleting actions using the stored procedures in the SQL Server database is very successful.

In order to keep our database neat and complete, refer to Section 7.6.2.2 to recover those deleted records one by one in the Server Explorer window. A complete project SQLUpdateDeleteSP can be found from the folder DBProjects\Chapter 7 located at the accompanying ftp site (see Chapter 1).

7.8.3 Update and Delete Data in Oracle Database Using Stored Procedures

To update and delete data using stored procedures developed in the Oracle database is very similar to the data updating and deleting actions we performed in the project developed in Section 7.7. With a small modification to an existing project OracleUpdateDeleteRTOObject, we can easily create our new project OracleUpdateDeleteSP to perform the data updating and deleting by calling stored procedures developed in the Oracle database.

To develop our new project in this section, we need to:

1. Modify the existing project OracleUpdateDeleteRTOObject to create our new project OracleUpdateDeleteSP.
2. Develop stored procedures in the Oracle database.
3. Call the stored procedures to perform the data updating and deleting using the Faculty Form window.

Now let's start with the first step.

7.8.3.1 *Modify Existing Project to Create New Project*

Open Windows Explorer and create a new folder Chapter 7 if you have not, and then browse to our project OracleUpdateDeleteRTObject from the folder DBProjects\Chapter 7 located at the accompanying ftp site (see Chapter 1). Copy the project OracleUpdateDeleteRTObject to our new folder C:\Chapter 7. Change the name of the project from OracleUpdateDeleteRTObject to OracleUpdataDeleteSP. Double-click on our new project OracleUpdataDeleteSP.csproj to open it.

On the opened project, perform the following modifications to get our desired project:

- Go to Project\OracleUpdataDeleteSP Properties menu item to open the project's property window. Change the **Assembly name** and the **Default namespace** from OracleUpdateDeleteRTObject to the OracleUpdataDeleteSP.
- Click on the **Assembly Information** button to open the Assembly Information dialog box, change the **Title** and the **Product** to OracleUpdataDeleteSP. Click on OK to close this dialog box.
- Change the project namespace for all files from OracleUpdateDeleteRTObject to OracleUpdataDeleteSP using the **Find and Replace** dialog box.

Go to the **File\Save All** to save these modifications. Now we are ready to modify our codes based on our new project OracleUpdataDeleteSP.

The code modifications include the following parts:

1. Replace the query string in the Update button's Click method in the Faculty Form with the name of the data updating stored procedure that will be developed in the next section to allow the method to call the related stored procedure to perform the data updating action.
2. Replace the query string in the Delete button's Click method in the Faculty Form with the name of the data deleting stored procedure that will be developed in the next section to allow the method to call the related stored procedure to perform the data deleting action.

Normally, these modifications should be performed after stored procedures have been created in the Oracle database since we need some information from these created stored procedures, such as the name of the stored procedure and the names of the input parameters to the stored procedure. Because of the similarity between this project and the last one, we assumed that we knew that information and we can put them into our methods in advance when we perform the coding modifications. This assumed information includes:

1. The name of the data updating stored procedure—UpdateFaculty_SP.
2. The names of the input parameters—prefix an **in** before each parameter's name that is equal to each column's name in the Faculty table in our sample database.
3. The name of the input dynamic parameter—FacultyName.
4. The name of the data deleting stored procedure—DeleteFaculty_SP.
5. The name of the input dynamic parameter—FacultyName.

Based on these assumptions, we can first modify our coding in the Update button's Click method. The key point is that we need to remember the names of these parameters and the name of the stored procedure, and put them into our stored procedure later when we developed it in the next section.


```

OracleUpdateDeleteSP.FacultyForm cmdUpdate_Click()
private void cmdUpdate_Click(object sender, EventArgs e)
{
    string cmdString = "UpdateFaculty_SP";
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    OracleCommand oraCommand = new OracleCommand();
    int intUpdate = 0;
    oraCommand.Connection = logForm.oraConnection;
    oraCommand.CommandType = CommandType.StoredProcedure;
    oraCommand.CommandText = cmdString;
    UpdateParameters(ref oraCommand);
    intUpdate = oraCommand.ExecuteNonQuery();
    oraCommand.Dispose();
    ComboName.Items.Clear();
    UpdateFaculty();
    if (intUpdate == 0)
        MessageBox.Show("The data updating is failed");
}

```

Figure 7.34 Modified coding for the Update button's Click method.

```

OracleUpdateDeleteSP.FacultyForm UpdateParameters()
private void UpdateParameters(ref OracleCommand cmd)
{
    cmd.Parameters.Add("inName", OracleType.Char).Value = txtName.Text;
    cmd.Parameters.Add("inOffice", OracleType.Char).Value = txtOffice.Text;
    cmd.Parameters.Add("inPhone", OracleType.Char).Value = txtPhone.Text;
    cmd.Parameters.Add("inCollege", OracleType.Char).Value = txtCollege.Text;
    cmd.Parameters.Add("inTitle", OracleType.Char).Value = txtTitle.Text;
    cmd.Parameters.Add("inEmail", OracleType.Char).Value = txtEmail.Text;
    cmd.Parameters.Add("FacultyName", OracleType.Char).Value = ComboName.Text;
}

```

Figure 7.35 Modified coding for the UpdateParameters method.

Open the Update button's Click method and modify its coding. The modified parts have been highlighted in bold. Your finished modifications to this method should match those codes shown in Figure 7.34.

Let's see how this piece of modified code works.

- A.** The content of the query string now should be equal to the name of the stored procedure, UpdateFaculty_SP, which we will create later.
- B.** The CommandType property of the Command object should be set to StoredProcedure to tell the project that a stored procedure should be called as the project runs to perform the data updating job.

The modifications to the codes of the user-defined UpdateParameters() method are shown in Figure 7.35.

Two modifications are performed for this method. The first modification is that the name of the dynamic parameter has been changed from Param1 to FacultyName since


```

OracleUpdateDeleteSP.FacultyForm  cmdDelete_Click()
private void cmdDelete_Click(object sender, EventArgs e)
{
A   string cmdString = "DeleteFaculty_SP";
    MessageBoxButtons vbButton = MessageBoxButtons.YesNo;
    LogInForm logForm = new LogInForm();
    logForm = logForm.getLogInForm();
    OracleCommand oraCommand = new OracleCommand();
    DialogResult Answer;
    int intDelete = 0;

    Answer = MessageBox.Show("Do you want to delete this record?", "Delete", vbButton);
    if (Answer == System.Windows.Forms.DialogResult.Yes)
    {
B       oraCommand.Connection = logForm.oraConnection;
        oraCommand.CommandType = CommandType.StoredProcedure;
        oraCommand.CommandText = cmdString;
C       oraCommand.Parameters.Add("FacultyName", OracleType.Char).Value = ComboName.Text;
        intDelete = oraCommand.ExecuteNonQuery();
        oraCommand.Dispose();

        if (intDelete == 0)
            MessageBox.Show("The data Deleting is failed");

        for (intDelete = 0; intDelete < 7; intDelete++) // clean up the Faculty textbox array
            FacultyTextBox[intDelete].Text = string.Empty;
    }
}

```

Figure 7.36 Modified coding for the Delete button's Click method.

we must keep all names of the input parameters to the stored procedure identical with those parameters we used in this coding in the Visual C#.NET project. The second modification is to add a prefix **in** before each input parameter. The reason for this is that the PL-SQL is a case-insensitive language. In order to distinguish between each columns' name of the Faculty table in our sample database and each input parameter's name to our stored procedure, we must add this prefix.

Next let's perform the code modification to the Delete button's Click method. Open that method and make the following modifications to this method, which are shown in Figure 7.36.

The modified parts have been highlighted in bold. Let's see how this piece of modified code works.

- A.** The content of the query string now should be equal to the name of the stored procedure, DeleteFaculty_SP, which we will create later.
- B.** The CommandType property of the Command object should be set to StoredProcedure to tell the project that a stored procedure should be called as the project runs to perform the data deleting job.
- C.** The name of the dynamic parameter that is located in the Add() method of the Parameters property, Param1, has been changed to FacultyName since we must keep all names of the input parameters to the stored procedure identical with those parameters we used in this method.

Now we have finished the coding modifications in the Visual C#.NET environment. Next let's start to create our stored procedures in the Oracle database. Refer to Sections

5.20.3.6 and 5.20.3.7 in Chapter 5 to get a detailed discussion about the stored procedure and package in the Oracle database. In this section, since we want to perform the data updating and deleting functions, we do not need any query to return any data from the database, and only stored procedure is good enough for our applications. There are many ways you can create the stored procedure in the Oracle database. One way is to create it using the Object Browser page in Oracle Database 10g XE, and another way is to create it using the SQL Command page. Since we used the Object Browser to create our sample database in Chapter 2, we prefer to use the second method to create our stored procedures in this section.

7.8.3.2 Develop Stored Procedure in Oracle Database

Open the Oracle Database 10g XE home page by going to Start|All Programs|Oracle Database 10g Express Edition|Go To Database Home Page items. Enter CSE_DEPT and reback into the Username and Password boxes, click on the Login button to login to the Oracle database. This time we want to use the SQL Command page to create our stored procedure. The reason for that is because we can run and test our stored procedure directly in the Oracle Database 10g XE environment as soon as the stored procedure is done. This is very convenient for us, and we do not need to wait to test it by calling the finished stored procedure later from the Visual C#.NET project.

To open the SQL Command page, click on the SQL icon and select the item SQL Commands|Enter Command to open this page. First, let's create the data updating stored procedure. Enter the codes shown in Figure 7.37 into this page as the body of our stored procedure.

Your finished stored procedure should match the one shown in Figure 7.38.

Now highlight the whole code body of this stored procedure and click on the Run button to create our stored procedure. Immediately you can find a message displayed in the bottom pane in the Results tab to indicate that the stored procedure is created, which is shown below:

```
Procedure created.
0.72 seconds
```

To run this stored procedure to test it in the Oracle Database 10g XE environment, type the codes that are shown below under the code body of the stored procedure:

```
create or replace PROCEDURE UpdateFaculty_SP
(inName IN VARCHAR2,
inOffice IN VARCHAR2,
inPhone IN CHAR,
inCollege IN VARCHAR2,
inTitle IN VARCHAR2,
inEmail IN VARCHAR2,
FacultyName IN VARCHAR2) AS
begin
UPDATE Faculty
SET faculty_name=inName, office=inOffice, phone=inPhone, college=inCollege,
title=inTitle, email=inEmail
WHERE faculty_name = FacultyName;
end;
```

Figure 7.37 Code body of the data updating stored procedure.

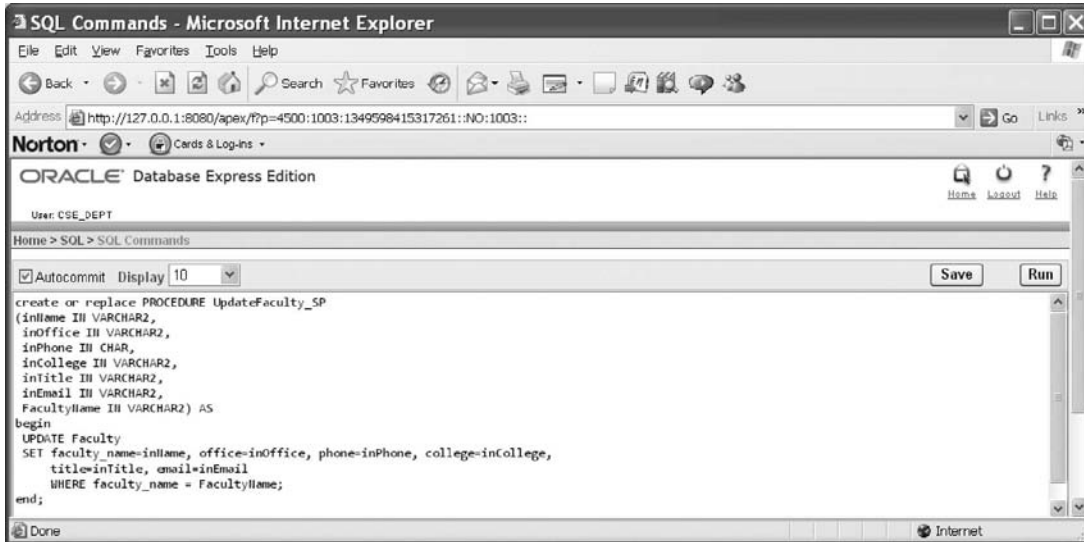


Figure 7.38 Finished data updating stored procedure.

```
begin
UpdateFaculty_SP('Peter Bai', 'MTC-228', '750-378-1222',
                 'University of Miami', 'Distinguished
                 Professor', 'pbai@college.edu', 'Ying Bai');
end;
```

Your finished test codes should match the one shown in Figure 7.39. Then highlight those codes and click on the Run button to run the stored procedure.

If the stored procedure is correctly created and executed, the running result, which is shown below, is displayed in the bottom pane under the Results tab:

```
Statement processed.
0.05 seconds
```

Now let's open our Faculty table to confirm that the selected row has been updated after the stored procedure UpdateFaculty_SP is executed. Click on the Home button, which is located at the upper-right corner of the page, to return to the Home page. Click on the drop-down arrow on the Object Browser icon and select the item Browse/Tables to open the Tables page.

Click on the Faculty table from the table list, and then click on the Data tab to open the Faculty table, which is shown in Figure 7.40.

It can be found that the fourth row, to which the arrow points, of the Faculty table has been updated. This confirmed that our stored procedure works fine.

Next let's create our data deleting stored procedure using the SQL Command page. Click on the Home button, which is located at the upper-right corner of the page, to return to the Home page. Click on the SQL icon and select the item SQL Commands/Enter Command to open the SQL Command page. Enter the codes shown in Figure 7.41 into this page as the body of our data deleting stored procedure.

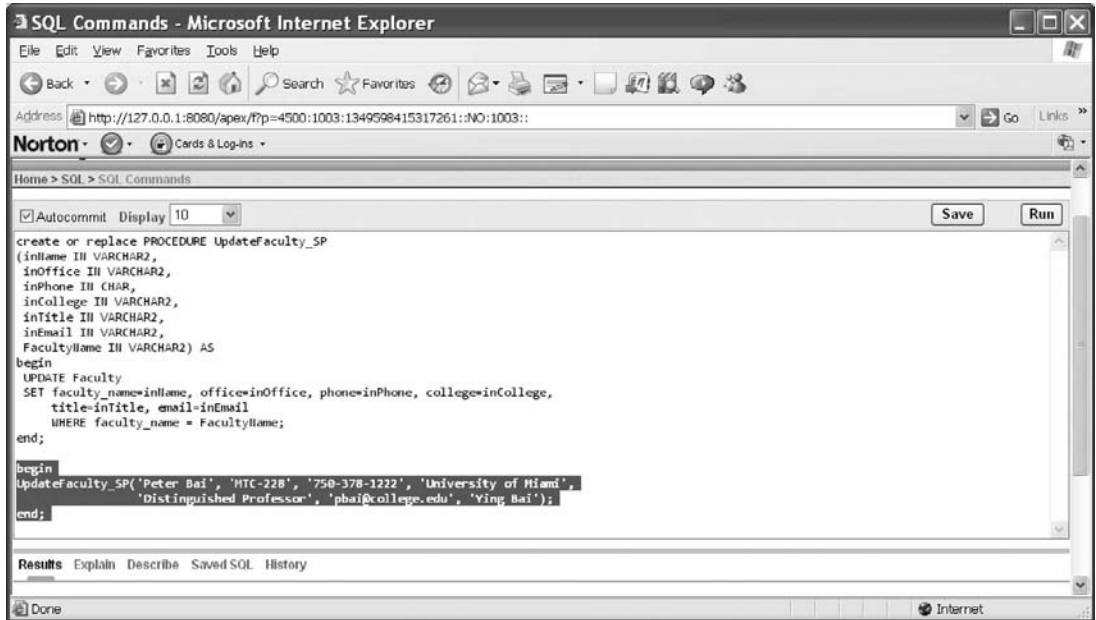


Figure 7.39 Codes to run the data updating stored procedure.

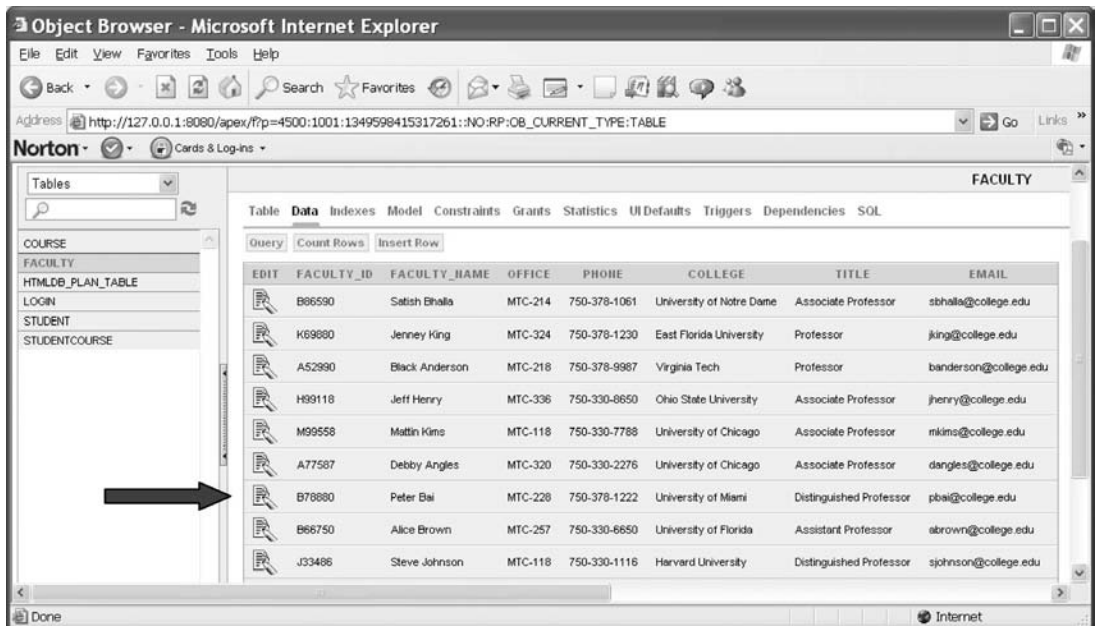


Figure 7.40 Updated Faculty table.

```

create or replace PROCEDURE DeleteFaculty_SP
 ( FacultyName IN VARCHAR2) AS
begin
  DELETE FROM Faculty WHERE faculty_name = FacultyName;
end;

```

Figure 7.41 Code body of the data deleting stored procedure.

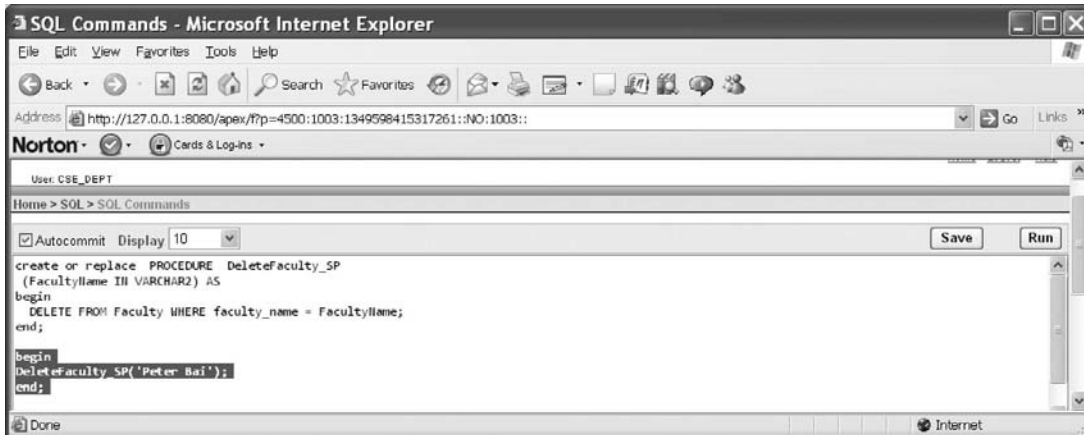


Figure 7.42 Codes to run the data deleting stored procedure.

Now highlight the whole code body of this stored procedure and click on the Run button to create our stored procedure. Immediately you can find a message displayed in the bottom pane in the Results tab to indicate that the stored procedure is created, which is shown below:

```

Procedure created.
0.05 seconds

```

To run this stored procedure to test it in the Oracle Database 10g XE environment, type the codes that are shown below under the code body of the stored procedure:

```

begin
DeleteFaculty_SP('Peter Bai');
end;

```

Your finished test codes should match the one shown in Figure 7.42. Then highlight those codes and click on the Run button to run the stored procedure.

If the stored procedure is correctly created and executed, the running result, which is shown below, is displayed in the bottom pane under the Results tab:

```

Statement processed.
0.18 seconds

```

Now let's open our Faculty table to confirm that the selected row has been deleted after the stored procedure DeleteFaculty_SP is executed. Click on the Home button, which is located at the upper-right corner of the page, to return to the Home page. Click on the drop-down arrow on the Object Browser icon and select the item Browse/Tables to open the Tables page.

Click on the Faculty table from the table list, and then click on the **Data** tab to open the Faculty table. It can be found that the faculty member **Peter Bai** with a `faculty_id = B78880` has been deleted from this Faculty table. The creation and execution of our data deleting stored procedure `DeleteFaculty_SP` are successful!

It is highly recommend to recover all deleted records related to this faculty member deletion using the **Insert Row** button. Refer to Section 7.6.2.2 to complete this data recovery. The order used to recover the deleted records is important. The correct order is: You must first recover the deleted faculty member from the Faculty table since it is a parent table, and then you can recover all other deleted records for all other related child tables, such as the Course, the LogIn, and the StudentCourse tables.

Now Close the Oracle Database 10g XE after these stored procedures have been created and tested successfully. Next we need to call these stored procedures from the Visual C#.NET project to perform the data updating and deleting functions.

7.8.3.3 Call Stored Procedures to Perform Data Updating and Deleting

Since we have finished the modifications to our new project in Section 7.8.3.1, now let's run our project to test the data updating and deleting functions by calling the stored procedures we developed in the last section.

Click on the **Start Debugging** button to run our project, enter the suitable username and password to the LogIn Form, and then select the Faculty Information item from the Selection form to open the Faculty Form window. Select the faculty member **Ying Bai** from the combobox control, and click on the **Select** button to display the information for the selected faculty.

To update this faculty member, enter the following information into the associated textboxes to perform this data updating:

Taylor Bai	Name textbox
Distinguished Professor	Title textbox
MTC-228	Office textbox
750-378-1222	Phone textbox
University of Miami	College textbox
tbai@college.edu	Email textbox

Click on the **Update** button to call the stored procedure to update this faculty information in the Faculty table in our sample database.

To confirm this updating, click on the drop-down arrow of the combobox control `ComboName` and select our new updated faculty member **Taylor Bai** from the box. Click on the **Select** button to retrieve that updated faculty information from the database and display it in this form. Click on **OK** to the `MessageBox` since we did not include any updated faculty photo for this faculty updating. Immediately you can find that the updated faculty information is returned and displayed, which is shown in Figure 7.43.

Next let's test the data deleting action by clicking on the **Delete** button to try to delete this updated record. Click on **Yes** to the `MessageBox` to perform this deleting. Immediately all information stored in textboxes is removed. To confirm this data deleting, click on the **Select** button to try to retrieve the deleted faculty information. A warning message "**No matched faculty found**" is displayed, which means that the selected faculty member has

Figure 7.43 Confirmation of the data updating.

been deleted from our database. Our data updating and deleting actions using the stored procedures in Oracle database is very successful.

In order to keep our database neat and complete, refer to Section 7.6.2.2 to recover those deleted records one by one using the **Insert Row** button in the Object Browser page. The point is to recover the deleted records. The correct order is: You must first recover the deleted faculty member from the Faculty table since it is a parent table, and then you can recover all other deleted records for all other related child tables, such as the Course, the LogIn, and the StudentCourse tables.

A complete project OracleUpdateDeleteSP can be found from the folder DBProjects\Chapter 7 located at the accompanying ftp site (see Chapter 1).

7.9 UPDATE AND DELETE DATA IN DATABASES USING LINQ TO SQL QUERY

As we discussed in Chapter 4, LINQ to SQL queries can perform not only the data selections but also the data insertion, updating, and deletion. The standard LINQ to SQL queries include:

- Select
- Insert
- Update
- Delete

To perform any of these operations or queries, we need to use entity classes and DataContext we discussed in Section 4.6.1 in Chapter 4 to do LINQ to SQL actions against our sample database. We have already created a Console project QueryLINQSQL in that section to illustrate how to use LINQ to SQL to perform data queries, such as data selection, insertion, updating, and deleting, against our sample database CSE_DEPT.mdf. However, in this section, we want to create a Windows-based project LINQSQLQuery by adding a graphic user interface to perform the data selection and data updating and deleting actions against our sample database CSE_DEPT.mdf using

the LINQ to SQL query. We leave the data insertion coding as homework to readers. Refer to Section 4.6.2.2 in Chapter 4 to complete the coding for this Insert button's Click method. Now let's perform the following steps to create our new project LINQSQLQuery:

1. Create a new Visual C# Windows-based project and name it as LINQSQLQuery.
2. Change the File Name to `Faculty Form.cs`.
3. Rename the Name and the Text of the default form window to `FacultyForm` and `CSE DEPT Faculty Form`, respectively.
4. Copy all controls from the Faculty Form from the project `SQLUpdateDeleteSP` we developed in this chapter and paste them into this new form.
5. Add `System.Data.Linq` reference to this new project by right-clicking on our new project from the Solution Explorer window, selecting the `Add Reference` item and scroll down the list, and selecting the item `System.Data.Linq` from the list and clicking the OK button.
6. Add the following directives at the top of the Faculty Form file:
 - using `System.Data.Linq`;
 - using `System.Data.Linq.Mapping`;
7. Follow the steps listed in Section 4.6.1 to create entity classes using the Object Relational Designer. The database used in this project is `CSE_DEPT.mdf` and it is located at the folder `C:\database\SQLServer`. Open the Server Explorer window and add this database by right-clicking on the Data Connections item and select `Add Connection` if it has not been added into our project.
8. We need to create five entity classes and each of them is associated with a data table in our sample database. Drag each table from the Server Explorer window and place it on the Object Relational Designer canvas. The mapping file's name is `CSE_DEPT.dbml`. Make sure that you enter this name into the Name box in the Object Relational Designer.
9. Right click on the mapping file `CSE_DEPT.dbml` from the Solution Explorer window and select the `View Code` item to create C# code file for our sample database, `CSE_DEPT.cs`.

Now let's begin the coding process for this project. Since we need to use the Select button's Click method to validate our data updating and deleting actions, we need to divide our coding process into the following four parts:

1. Create a new object of the `DataContext` class and do some initialization coding.
2. Develop the codes for the Select button's Click method to retrieve the selected faculty information using the LINQ to SQL query.
3. Develop the codes for the Update button's Click method to update the selected faculty member using the LINQ to SQL query.
4. Develop the codes for the Delete button's Click method to delete the selected faculty member using the LINQ to SQL query.

Now let's start with part 1.

7.9.1 Create New Object of DataContext Class

We need to create this new object of the `DataContext` class since we need to use this object to connect to our sample database to perform data queries. We have connected this `DataContext` class to our sample database `CSE_DEPT.mdf` in step 7 above, and the

connection string has been added into our `app.config` file when step 7 is done. Therefore, we do not need to indicate the special connection string when we create this object.

Some initialization coding includes retrieving all updated faculty members from the Faculty table in our sample database using the LINQ to SQL query and display them in the ComboName combobox control.

Open the code window and the constructor of the FacultyForm class, and enter the codes shown in Figure 7.44 into this constructor.

Let's have a close look at this piece of code to see how it works.

- A. A new field-level object of the DataContext class, `cse_dept`, is created first since we need to use this object to connect our sample database to this project to perform the data actions later.
- B. A user-defined method `UpdateFaculty()` is executed to retrieve all updated faculty members from our sample database and display them in the ComboName combobox control to allow users to select a desired faculty.
- C. The LINQ query is created and initialized with three clauses: `from`, `let`, and `select`. The range variable `fi` is selected from the Faculty entity in our sample database. All current faculty members (`faculty_name`) will be read back using the `let` clause.
- D. The LINQ query is executed to pick up all queried faculty members and add them into the ComboName combobox control in our Faculty Form.

Let's continue to develop the coding for the second part.

7.9.2 Develop Codes for Select Button Click Method

Double-click on the Select button to open its Click method and enter the codes shown in Figure 7.45 into this method. The function of this piece of code is to retrieve all current

```

LINQSQLQuery.FacultyForm | FacultyForm()
public partial class FacultyForm : Form
{
A   CSE_DEPTDataContext cse_dept = new CSE_DEPTDataContext();
    public FacultyForm()
    {
B       InitializeComponent();
        UpdateFaculty();
        ComboName.SelectedIndex = 0;
    }

    void UpdateFaculty()
    {
C       var faculty = (from fi in cse_dept.Faculties
                       let fields = "faculty_name"
                       select fi);
D       foreach (var f in faculty)
        {
            ComboName.Items.Add(f.faculty_name);
        }
    }
    .....

```

Figure 7.44 Initialization codes for the constructor of the Faculty Form.

```

LINQSQLQuery.FacultyForm  cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
A   string strName = ShowFaculty(ComboName.Text);
   if (strName == "No Match")
       MessageBox.Show("No Matched Faculty Image found!");
B   var faculty = (from fi in cse_dept.Faculties
                  where fi.faculty_name == ComboName.Text
                  select fi);
C   foreach (var f in faculty)
   {
       txtName.Text = f.faculty_name;
       txtTitle.Text = f.title;
       txtOffice.Text = f.office;
       txtPhone.Text = f.phone;
       txtCollege.Text = f.college;
       txtEmail.Text = f.email;
   }
}

```

Figure 7.45 Codes for the Select button's Click method.

faculty members from the Faculty table in our sample database and display them in the ComboName combobox control in the Faculty Form window as this Select button is clicked on by the user.

Let's have a close look at this piece of code to see how it works.

- A.** The user-defined ShowFaculty() method is executed to identify and display a matched faculty image for the selected faculty member. You can copy this piece of code from the Faculty Form class in the previous projects we developed in this chapter.
- B.** The LINQ query is created and initialized with three clauses: from, where, and select. The range variable fi is selected from the Faculty entity in our sample database based on a matched faculty members (faculty_name).
- C.** The LINQ query is executed to pick up all columns for the selected faculty member and display them on the associated textbox in our Faculty Form.

It is recommended to copy the body of the user-defined ShowFaculty() method from any Faculty Form class in the previous projects we developed in this chapter, and paste it into this FacultyForm class.

Now let's concentrate on the coding for our data updating and deleting actions.

7.9.3 Develop Codes for Update Button Click Method

Double-click on the Update button from our Faculty Form window to open its Click method and enter the codes shown in Figure 7.46 into this method.

Let's have a close look at this piece of code to see how it works.

- A.** A selection query is executed using the Standard Query Operator methods with the faculty_name as the query criterion. The First() method is used to return only the first matched record. It does not matter for our application since we have only one record that is associated with this specified faculty_name.

```

LINQSQLQuery.FacultyForm  cmdUpdate_Click()
private void cmdUpdate_Click(object sender, EventArgs e)
{
    Faculty fi = cse_dept.Faculties.Where(f => f.faculty_name == ComboName.Text).First();
    // updating the existing faculty information
    fi.faculty_name = txtName.Text;
    fi.title = txtTitle.Text;
    fi.office = txtOffice.Text;
    fi.phone = txtPhone.Text;
    fi.college = txtCollege.Text;
    fi.email = txtEmail.Text;
    cse_dept.SubmitChanges();
    ComboName.Items.Clear();
    UpdateFaculty();
}

```

Figure 7.46 Codes for the Update button's Click method.

- B.** All six columns for the selected faculty member are updated by assigning the current value stored in the associated textbox to each column in the Faculty table in our sample database.
- C.** This data updating can occur only after the SubmitChanges() method is executed.
- D.** The ComboName combobox control is cleaned up and the user-defined UpdateFaculty() method is executed to refresh the updated faculty members stored in that control.

You can run the project now to perform the data updating for the selected faculty member such as Ying Bai with the following updated information:

Peter Bai	Faculty Name textbox
Distinguished Professor	Title textbox
MTC-228	Office textbox
750-378-1223	Phone textbox
University of Main	College textbox
pbai@college.edu	Email textbox

After this updating, you can click on the Select button to confirm whether this data updating is successful or not. Remember, you need to recover the updated information to the originals for the selected faculty member in order to keep our sample database neat and complete. A good way to do that recovery is to run this project again and perform another updating with the following updated information for the updated faculty member Ying Bai:

Ying Bai	Faculty Name textbox
Associate Professor	Title textbox
MTC-211	Office textbox
750-378-1148	Phone textbox
Florida Atlantic University	College textbox
ybai@college.edu	Email textbox

Next let's perform the coding for the data deleting action.

7.9.4 Develop Codes for Delete Button Click Method

Double-click on the Delete button from the Faculty Form window to open its Click method and enter the codes shown in Figure 7.47 into this method.

Let's take a close look at this piece of code to see how it works.

- A. The LINQ query is created and initialized with three clauses: **from**, **where**, and **select**. The range variable `fi` is selected from the Faculty, which is exactly an instance of our entity class Faculty, and the `faculty_name` works as the query criterion for this query. All information related to the selected faculty members (`faculty_name`) will be retrieved and stored in the query variable `faculty`. The `Single` means that only a single record is queried.
- B. The system method `DeleteOnSubmit()` is executed to issue a deleting action to the faculty instance, `Faculties`.
- C. Another system method `SubmitChanges()` is executed to exactly perform this deleting action against data tables in our sample database. Only after this method is executed is the deleting action actually performed on our database.
- D. All textboxes stored information related to the deleted faculty are cleaned up by assigning an empty string to each of them.

Now we can build and run our project to test the data deleting action against our sample database. However, note that before we can run the project we must make sure that all faculty image files have been stored in the default folder in which our executable file `LINQSQLQuery.exe` is located. In this application, it should be `C:\Chapter 7\LINQSQLQuery\bin\Debug`.

It is highly recommended to recover all deleted records related to that deleted faculty member after you finished testing the data deleting action. Remember, this deleting action not only deletes a single faculty record from the Faculty table but it also deletes some related records from the Course, LogIn, StudentCourse tables (child tables). Therefore, you need to recover all of those related records in the associated tables. Refer to section 7.6.2.2 to recover all of those related records if you deleted a faculty member Ying Bai by running this test.

```

LINQSQLQuery.FacultyForm cmdDelete_Click()
private void cmdDelete_Click(object sender, EventArgs e)
{
  A   var faculty = (from fi in cse_dept.Faculties
                    where fi.faculty_name == ComboName.Text
                    select fi).Single<Faculty>());
  B   cse_dept.Faculties.DeleteOnSubmit(faculty);
  C   cse_dept.SubmitChanges();

  D   // clean up all textboxes
      txtName.Text = string.Empty;
      txtOffice.Text = string.Empty;
      txtTitle.Text = string.Empty;
      txtPhone.Text = string.Empty;
      txtCollege.Text = string.Empty;
      txtEmail.Text = string.Empty;
}

```

Figure 7.47 Codes for the Delete button's Click method.

A complete project LINQSQLQuery can be found from the folder DBProjects\Chapter 7 located at the accompanying ftp site (see Chapter 1).

7.10 CHAPTER SUMMARY

Data updating and deleting queries are discussed in this chapter with three popular databases: Microsoft Access, SQL Server, and Oracle.

Four popular data updating and deleting methods are discussed and analyzed with eight real project examples:

1. Use TableAdapter DBDirect methods such as TableAdapter.Update() and TableAdapter.Delete() to update and delete data directly against the databases.
2. Use TableAdapter.Update() method to update and execute the associated TableAdapter's properties such as UpdateCommand or DeleteCommand to save changes made for the table in the DataSet to the table in the database.
3. Use the runtime objects method to develop and execute the Command's method ExecuteNonQuery() to update or delete data against the database directly.
4. Use LINQ to SQL query method to update and delete data against our sample SQL Server database CSE_DEPT.mdf.

Both methods 1 and 2 need to use the Visual Studio.NET Design Tools and Wizards to create and configure suitable TableAdapters, build the associated queries using the Query Builder, and call those queries from the Visual C#.NET applications. The difference between methods 1 and 2 is that method 1 can be used to directly access the database to perform the data updating and deleting in a single step. Method 2 needs two steps to finish the data updating or deleting. First, the data updating or deleting are performed to the associated tables in the DataSet, and then those updated or deleted data are updated to the tables in the database by executing the TableAdapter.Update() method.

This chapter is divided into two parts: Part I provides discussions on data updating and deleting using methods 1 and 2, or in other words, using the TableAdapter.Update() and TableAdapter.Delete() methods developed in the Visual Studio.NET Design Tools and Wizards. Part II presents the data updating and deleting using the runtime objects method to develop command objects to execute the ExecuteNonQuery() method dynamically. Updating and deleting data against our sample database using the LINQ to SQL query method are also discussed in the second part.

Eight real sample projects are provided in this chapter to help readers to understand and design the professional data-driven applications to update or delete data against three types of databases: Microsoft Access 2007, SQL Server 2005 SP2, and Oracle Database 10g XE R2. The stored procedures are discussed in the last section of each part to help readers to perform the data updating or deleting more efficiently and conveniently.

HOMEWORK

I. True/False Selections

- ___1. Three popular data updating methods are: the TableAdapter DBDirect method and the tableAdapter.Update() and ExecuteNonQuery() methods of the Command class.
- ___2. Unlike the Fill() method, a valid database connection must be set before data can be updated in the database.

- ___3. One can directly update data or delete records against the database using the TableAdapter.Update() method.
- ___4. When executing an UPDATE query, the order of the input parameters in the SET list can be different with the order of the data columns in the database.
- ___5. To update data against the Oracle database using stored procedures, an Oracle Package must be developed to include stored procedures.
- ___6. One can directly delete records from the database using the TableAdapter DBDirect method such as TableAdapter.Delete() method.
- ___7. When performing the data updating, the same data can be updated in the database multiple times.
- ___8. To delete data from the database using the TableAdapter.Update() method, the data should be first deleted from the table in the DataSet, and then the Update() method is executed to update that deletion to the table in the database.
- ___9. To update data in the SQL Server database using the stored procedures, one can create and test the created stored procedure in the Server Explorer window.
- ___10. To call stored procedures to update data against a database, the input parameters' names must be identical with those names of the input parameters defined in the stored procedures.

II. Multiple Choices

- 1. To update data in the database using the TableAdapter.Update() method, one needs to use the _____ to build the _____.
 - a. Data Source, Query Builder
 - b. TableAdapter Query Configuration Wizard, Update query
 - c. Runtime object, Insert query
 - d. Server Explorer, Data Source
- 2. To delete data from the database using the TableAdapter.Update() method, one needs first to delete data from the _____, and then update that data into the database.
 - a. Data table
 - b. Data table in the database
 - c. DataSet
 - d. Data table in the DataSet
- 3. To delete data from the database using the TableAdapter.Update() method, one can delete _____.
 - a. One data row only
 - b. Multiple data rows
 - c. The whole data table
 - d. Any of the above
- 4. Because the ADO.NET provides a disconnected mode to the database, to update or delete a record against the database, a valid _____ must be established.
 - a. DataSet
 - b. TableAdapter
 - c. Connection
 - d. Command

5. The _____ operator should be used as an assignment operator for the WHERE clause with a dynamic parameter for a data query in Oracle database.
 - a. =:
 - b. LIKE
 - c. =
 - d. @
6. To test a data deleting stored procedure built in the Object Browser page in Oracle database, one can _____ the stored procedure to make sure it works.
 - a. Build
 - b. Test
 - c. Debug
 - d. Compile
7. To test a data updating stored procedure built in the Server Explorer window for the SQL Server database, one can _____ the stored procedure to make sure it works.
 - a. Build
 - b. Execute
 - c. Debug
 - d. Compile
8. To update data in an Oracle database using the UPDATE command, the data types of the parameters in the SET list should be _____.
 - a. OleDbType
 - b. SqlDbType
 - c. OracleDbType
 - d. OracleType
9. To update data using stored procedures, the CommandType property of the Command object must be equal to _____.
 - a. CommandType.InsertCommand
 - b. CommandType.StoredProcedure
 - c. CommandType.Text
 - d. CommandType.Insert
10. To update data using stored procedures, the CommandText property of the Command object must be equal to _____.
 - a. The content of the CommandType.InsetCommand
 - b. The content of the CommandType.Text
 - c. The name of the Insert command
 - d. The name of the stored procedure

III. Exercises

1. A stored procedure developed in the SQL Server database is shown in Figure 7.48. Please develop a piece of code in Visual C#.NET to call this stored procedure to update a record in the database.
2. A piece of code developed in Visual C#.NET is shown in Figure 7.49. This coding is used to call a stored procedure in the Oracle database to update a record in the database. Please create the associated stored procedure in the Oracle database using the PL- SQL language.

3. Using the tools and wizards provided by Visual Studio.NET and ADO.NET to perform the data updating for the Student Form in the AccessUpdateDeleteWizard project (the project file is located at the folder DBProjects\Chapter 7 that can be found from the accompanying ftp site (see Chapter 1)).
4. Using the runtime objects method to complete the update data query for the Student Form by using the project SQLUpdateDeleteRTOObject (the project file is located at the folder DBProjects\Chapter 7 found at the accompanying ftp site (see Chapter 1)).
5. Using the stored procedure to complete the data updating query for the Student Form to the Student table by using the project OracleUpdateDeleteSP (the project file is located at the folder DBProjects\Chapter 7 found at the accompanying ftp site (see Chapter 1)).
6. Using the stored procedure to complete the data deleting query for the Student Form to the Student table by using the project OracleUpdateDeleteSP (the project file is located at the folder DBProjects\Chapter 7 found at the accompanying ftp site (see Chapter 1)). It is highly recommended to recover those deleted records after they are deleted.

Hints: Four tables are involved in this data deleting action: Student, LogIn, Course, and the StudentCourse tables. The recovery order is, first, recover the record from the parent table (Student table), and then recover all other records for all other tables.

```

CREATE OR REPLACE PROCEDURE dbo.UpdateStudent
(@Name IN VARCHAR(20),
 @Major IN text,
 @SchoolYear IN int,
 @Credits IN float,
 @Email IN text
 @StudentName IN VARCHAR(20))
```

AS
UPDATE Student SET name=@Name, major=@Major, schoolYear=@SchoolYear,
credits=@Credits, email=@Email
WHERE (name=@StudentName)
RETURN

Figure 7.48

```

string cmdString = "UpdateCourse";
int intInsert = 0;
OracleCommand oraCommand = new OracleCommand();
oraCommand.Connection = oraConnection;
oraCommand.CommandType = CommandType.StoredProcedure;
oraCommand.CommandText = cmdString;
oraCommand.Parameters.Add("Name", OracleType.Char).Value = ComboName.Text;
oraCommand.Parameters.Add("CourseID", OracleType.Char).Value = txtCourseID.Text;
oraCommand.Parameters.Add("Course", OracleType.Char).Value = txtCourse.Text;
oraCommand.Parameters.Add("Schedule", OracleType.Char).Value = txtSchedule.Text;
oraCommand.Parameters.Add("Classroom", OracleType.Char).Value = txtClassRoom.Text;
oraCommand.Parameters.Add("Credit", OracleType.Char).Value = txtCredits.Text;
oraCommand.Parameters.Add("StudentID", OracleType.Char).Value = txtID.Text;
intInsert = oraCommand.ExecuteNonQuery();
```

Figure 7.49

Chapter 8

Accessing Data in ASP.NET

We have provided a very detailed discussion on database programming with Visual C#.NET using the Windows-based applications in the previous chapters. Starting with this chapter, we will concentrate on database programming with Visual C#.NET using Web-based applications. To develop the Web-based application and allow users to access the database through the Internet, you need to understand an important component: Active Server Page.NET or ASP.NET.

Essentially, ASP.NET allows users to write software to access databases through a Web browser rather than a separate program installed on their computers. With the help of ASP.NET, the users can easily create and develop an ASP.NET Web application and run it on the server as a server-side project. The user then can send requests to the server to download any Web page and to access the database to retrieve, display, and manipulate data via the Web browser. The actual language used in the communications between the client and the server is Hypertext Markup Language (HTML).

When finished this chapter, you will:

- Understand the structure and components of ASP.NET Web applications.
- Understand the structure and components of .NET Framework.
- Select data from the database and display data in a Web page.
- Understand the Application state structure and implement it to store global variables.
- Understand the AutoPostBack property and implement it to communicate with the server effectively.
- Insert, update, and delete data from the database through a Web page.
- Use the stored procedure to perform the data actions against the database via a Web application.
- Use LINQ to SQL query to perform the data actions against the database via a Web application.
- Perform client-side data validation in Web pages.

In order to help readers to successfully complete this chapter, first we need to provide a detailed discussion about the ASP.NET. But the prerequisite to understanding the ASP.NET is the .NET Framework since the ASP.NET is a part of .NET Framework, or in other words, the .NET Framework is the foundation of the ASP.NET. So we need first to give a detailed discussion about the .NET Framework.

8.1 WHAT IS .NET FRAMEWORK?

The .NET Framework is a model that provides a foundation to develop and execute different applications at an integrated environment such as Visual Studio.NET. In other words, the .NET Framework can be considered as a system to integrate and develop multiple applications such as Windows applications, Web applications, or XML Web Services by using a common set of tools and codes such as Visual C#.NET or Visual Basic.NET.

The current version of the .NET Framework is 3.5. Basically, the .NET Framework consists of the following components:

- The Common Language Runtime—CLR (called runtime). The runtime handles runtime services such as language integration, security, and memory management. During the development stage, the runtime provides features that are needed to simplify the development.
- Class Libraries. Class libraries provide reusable codes for most common tasks such as data access, XML Web service development, and Web and Windows forms.

The main goal to develop the .NET Framework is to overcome several limitations on Web applications since different clients may provide different client browsers. To solve these limitations, .NET Framework provides a common language called Microsoft Intermediate Language (MSIL) that is language independent and platform independent, and allows all programs developed in any .NET-based language to be converted into this MSIL. The MSIL can be recognized by the Common Language Runtime (CLR), and the CLR can compile and execute the MSIL codes by using the Just-In-Time compiler located at the local machines or clients.

You access the .NET Framework by using the class libraries provided by the .NET Framework, and you implement the .NET Framework by using the tools such as Visual Studio.NET provided by the .NET Framework, too. All class libraries provided by the .NET Framework are located at the different namespaces. All .NET-based languages access the same libraries. A typical .NET Framework model is shown in Figure 8.1.

The .NET Framework supports three types of user interfaces:

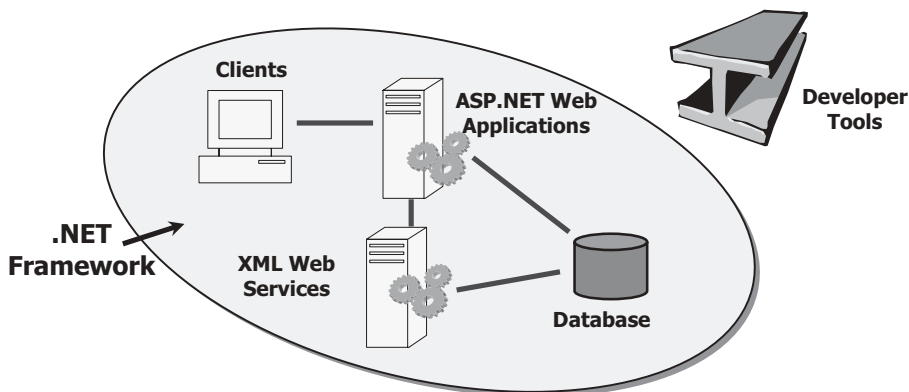


Figure 8.1 A .NET Framework model.

- Windows Forms that run on Windows 32 client computers. All projects we developed in the previous chapters used this kind of user interface.
- Web Forms that run on Server computers through ASP.NET and the Hypertext Transfer Protocol (HTTP).
- The Command Console.

The advantages of using the .NET Framework to develop Windows-based and Web-based applications include but are no limited to:

- The .NET Framework is based on Web standards and practices, and it fully supports Internet technologies, including HTML, HTTP, XML, Simple Object Access Protocol (SOAP), XML Path Language (XPath), and other Web standards.
- The .NET Framework is designed using unified application models, so the functional of any class provided by the .NET Framework is available to any .NET-compatible language or programming model. The same piece of code can be implemented in Windows applications, Web applications, and XML Web Services.
- The .NET Framework is easy for developers to use since the code in the .NET Framework is organized into hierarchical namespaces and classes. The .NET Framework provides a common type system, which is called the unified type system, and it can be used by any .NET-compatible language. In the unified type system, all language elements are objects that can be used by any .NET application written in any .NET-based language.

Now let's take a closer look at the ASP.NET.

8.2 WHAT IS ASP.NET AND ASP.NET 3.5?

ASP.NET is a programming framework built on the .NET Framework, and it is used to build Web applications. Developing ASP.NET Web applications in the .NET Framework is very similar to developing Windows-based applications. An ASP.NET Web application is composed of many different parts and components, but the fundamental component of ASP.NET is the Web Form. A Web Form is the Web page that users view in a browser, and an ASP.NET Web application can contain one or more Web Forms. A Web Form is a dynamic page that can access server resources.

The current version of the ASP.NET is 3.5, which is combined with .NET Framework 3.5 to provide a professional and convenient way to help users build and develop a variety of data-driven applications in .NET programming languages. Compared with the progression from ASP.NET 2.0, the features in ASP.NET 3.5 are additive, which means that the core assemblies installed from the .NET Framework 2.0 are still used by the 3.5 version. In short, ASP.NET 3.5 does not change, take away, or break any function, concepts, or code present in 2.0, but it simply adds new types and features and capabilities to the framework. Therefore, ASP.NET 3.5 is a rather minor upgrade from ASP.NET 2.0; that is, there are not many new ASP.NET-specific features in the .NET Framework 3.5.

There are three new features worth noting in ASP.NET 3.5:

- Integrated ASP.NET AJAX support
- The ListView control
- The DataPager control

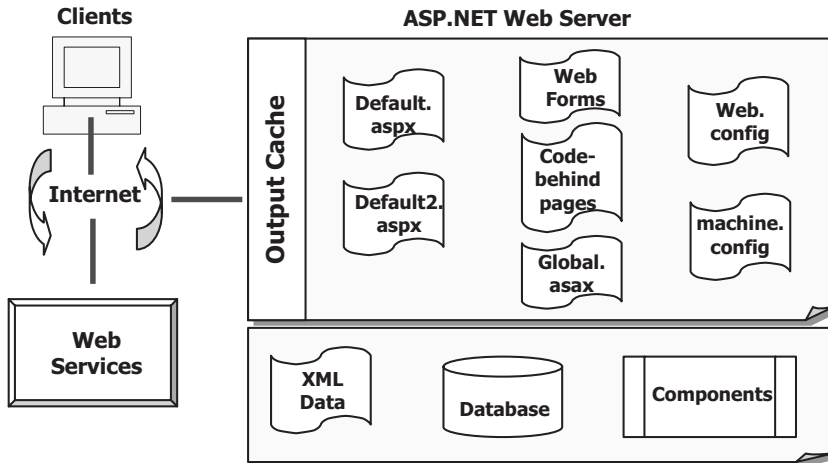


Figure 8.2 Structure of an ASP.NET Web application.

Besides these new features, one of the most significant differences between ASP.NET 3.5 and ASP.NET 2.0 is that the LINQ support is added to ASP.NET 3.5. LINQ provides a revolutionary solution between the different query syntaxes used in the different databases and bridges the gap between the world of objects and the world of data.

A completed structure of an ASP.NET Web application is shown in Figure 8.2.

Unlike a traditional Web page that can run scripts on the client, an ASP.NET Web Form can also run server-side codes to access databases, to create additional Web Forms, or to take advantage of built-in security of the server. In addition, since an ASP.NET Web Form does not rely on client-side scripts, it is independent on the client's browser type or operating system. This independence allows users to develop a single Web Form that can be viewed on any device that has Internet access and a Web browser.

Because ASP.NET is part of the .NET Framework, the ASP.NET Web application can be developed in any .NET-based language.

The ASP.NET technology also supports XML Web Services. XML Web Services are distributed applications that use XML for transferring information between clients, applications, and other XML Web Services.

The main parts of an ASP.NET Web application include:

- Web Forms or `Default.aspx` pages. The Web Forms or `Default.aspx` pages provide the user interface for the Web application, and they are very similar to the Windows Forms in the Windows-based application. The Web Forms files are indicated with an extension of `.aspx`.
- Code-behind pages. The so-called code-behind pages are related to the Web Forms and contain the server-side codes for the Web Form. This code-behind page is very similar to the code window for the Windows Forms in a Windows-based application we discussed in the previous chapters. Most event methods or handlers associated with controls on the Web Forms are located in this code-behind page. The code-behind pages are indicated with an extension of `.aspx.cs`.
- Web Services or `.asmx` pages. Web services are used when you create dynamic sites that will be accessed by other programs or computers. ASP.NET Web Services may be supported by a code-behind page designed by the extension of `.asmx.cs`.

- Configuration files. The Configuration files are XML files that define the default settings for the Web application and the Web server. Each Web application has one **Web.config** configuration file, and each Web server has one **machine.config** file.
- **Global .aspx** file. The **Global.aspx** file, also known as the ASP.NET application file, is an optional file that contains code for responding to application-level events that are raised by ASP.NET or by HttpModules. At runtime, **Global.aspx** is parsed and compiled into a dynamically generated .NET Framework class that is derived from the **HttpApplication** base class. This dynamic class is very similar to the **Application** class or main thread in Visual C++, and this class can be accessed by any other objects in the Web application.
- XML Web service links. These links are used to allow the Web application to send and receive data from an XML Web service.
- Database connectivity. The Database connectivity allows the Web application to transfer data to and from database sources. Generally, it is not recommended to allow users to access the database from the server directly because of security issues. Instead, in most industrial and commercial applications, the database can be accessed through the application layer to strengthen the security of the databases.
- Caching. Caching allows the Web application to return Web Forms and data more quickly after the first request.

8.2.1 ASP.NET Web Application File Structure

When you create an ASP.NET Web application, Visual Studio.NET creates two folders to hold the files that relate to the application. When the project is compiled, a third folder is created to store the terminal dll file. In other words, the final or terminal file of an ASP.NET Web application is a dynamic linked library file (.dll). Figure 8.3 shows a typical file structure of an ASP.NET Web application.

The folders on the left side in Figure 8.3 are very familiar to us since they are created by the Windows-based applications. But the folders on the right side are new to us, and the functions of those folders are:

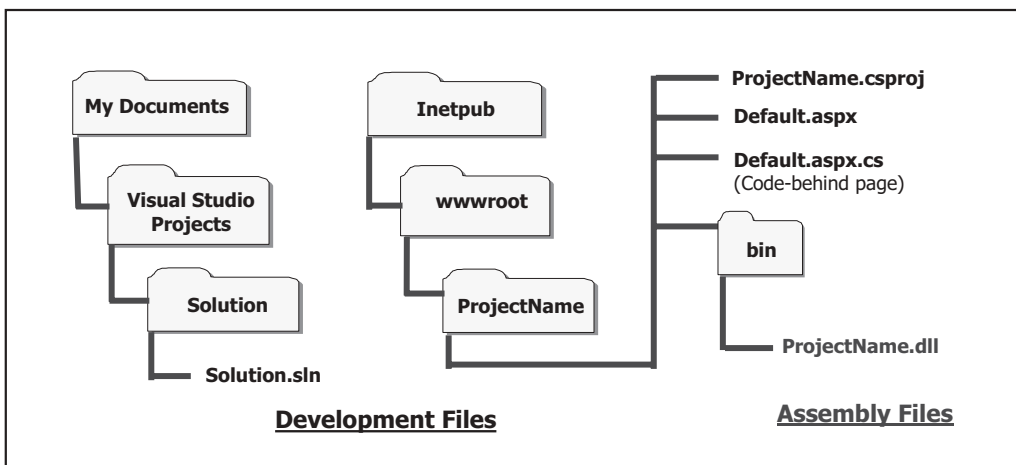


Figure 8.3 ASP.NET Web application file structure.

- The `inetpub` folder contains another folder named `wwwroot`, and it is used to hold the root address of the Web project whose name is defined as `ProjectName`. The project file `ProjectName.csproj` is an XML file that contains references to all project items, such as forms and classes.
- The `bin` folder contains the assembly file or the terminal file of the project with the name of `ProjectName.dll`. All ASP.NET Web applications will be finally converted to a `dll` file and stored in the server's memory.

8.2.2 ASP.NET Execution Model

When you finished an ASP.NET Web application, the Web project is compiled and two terminal files are created:

1. Project Assembly files (`.dll`). All code-behind pages (`.aspx.cs`) in the project are compiled into a single assembly file that is stored as `ProjectName.dll`. This project assembly file is placed in the **bin** directory of the Web site and will be executed by the Web server as a request is received from the client at running time.
2. `AssemblyInfo.cs` file. This file is used to write the general information, specially assembly version and assembly attributes, about the assembly.

As a Web project runs and the client requests a Web page for the first time, the following events occur:

1. The client browser issues a GET HTTP request to the server.
2. The ASP.NET parser interprets the course code.
3. Based on the interpreting result, ASP.NET will direct the request to the associated assembly file (`.dll`) if the code has been compiled into the `dll` files. Otherwise, the ASP.NET invokes the compiler to convert the code into the `dll` format.
4. Runtime loads and executes the Microsoft Intermediate Language (MSIL) codes and sends back the required Web page to the client in the HTML file format.

For the second time when the user requests the same Web page, no compiling process is needed, and the ASP.NET can directly call the `dll` file and execute the MSIL code to speed up this request.

From this execution sequence, it looks like the execution or running of a Web application is easy and straightforward. However, in practice, a lot of data round trips occurred between the client and the server. To make it clear, let's continue the discussion and analysis of this issue and see what really happens between the client and the server as a Web application is executed.

8.2.3 What Really Happens When a Web Application Is Executed?

The key point is that a Web Form is built and run on the Web server. When the user sends a request from the user's client browser to request that Web page, the server needs to build that form and send it back to the user's browser in the HTML format. Once the

Web page is received by the client's browser, the connection between the client and the server is terminated. If the user wants to request any other page or information from the server, additional requests must be submitted.

To make this issue more clear, we can use our LogIn Form as an example. The first time the user sends a request to the server to ask to start a logon process, the server builds the LogIn Form and sends it back to the client in the HTML format. After that, the connection between the client and the server is gone. After the user receives the LogIn Web page and enters the necessary logon information such as the username and password to the LogIn Form, the user needs to send another request to the server to ask the server to process those pieces of logon information. After the server receives and processes the logon information, if the server finds that the logon information is invalid, the server needs to rebuild the LogIn Form and resend it to the client with some warning message. Therefore, you can see how many round trips occurred between the client and the server as a Web application is executed.

A good solution to try to reduce these round trips is to make sure that all the information entered from the client side is as correct as possible. In other words, try to make as much validation as possible on the client side to reduce the burden of the server.

Now we have finished the discussion about the .NET Framework and ASP.NET as well as the ASP.NET Web applications. Next we will create and develop some actual Web projects using the ASP.NET Web Forms to illustrate how to access the database through the Web browser to select, display, and manipulate data on Web pages.

8.2.4 Requirements to Test and Run a Web Project

Before we can start to create our real Web project using the ASP.NET, we need the following requirements to test and run our Web project:

1. **Web server:** To test and run our Web project, you need a Web server either on your local computer or on your network. By default, if you installed the Internet Information Services (IIS) on your local computer before the .NET Framework is installed on your computer, the FrontPage Server Extension 2000 should have been installed on your local computer. This software allows your Web development tools such as Visual Studio.NET to connect to the server to upload or download pages from the server.
2. In this chapter, in order to make our Web project simple and easy, we always use our local computer as a pseudoserver. In other words, we always use the localhost, which is the IP name of our local computer, as our Web server to communicate with our browser to perform the data accessing and manipulating.

If you have not installed the IIS on your computer, follow the steps below to install this component on your computer:

- Click on **Start**, then click on **Control Panel**, and click on **Add or Remove Programs**.
- Click on **Add/Remove Windows Components**. The **Windows Components Wizard** appears, which is shown in Figure 8.4.
- Check the checkbox for the **Internet Information Services (IIS)** from the list to add the IIS to your computer. To confirm that this installation contains the installation of the **FrontPage 2000 Server Extensions**, click on the **Details** button to open the IIS dialog box.

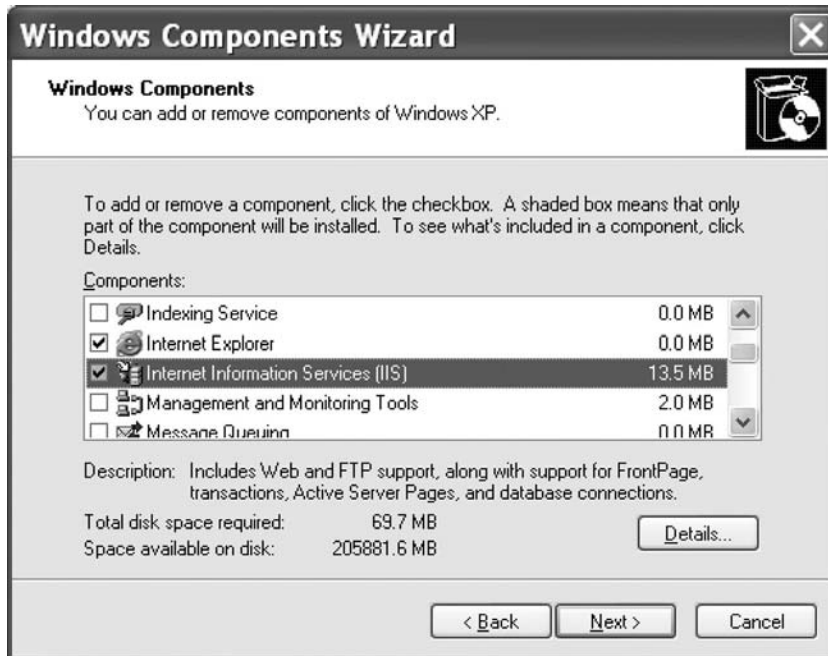


Figure 8.4 Opened Windows Components dialog box.

Check on the checkbox for the **FrontPage 2000 Server Extensions** to select it if it is not checked. Although Microsoft has stopped supporting this version of the server and the current version is **FrontPage 2002 Server Extensions**, you can still use it without any problem.

- Click on the **OK** button to close the **IIS** dialog box.
- Click on the **Next** button to begin to install the **IIS** and the **FrontPage 2000 Server Extensions** to your computer.

You may be asked to insert the **Windows XP SP2 Operating System CD** into your **CD** drive since this installation needs some files in that system disk. Just follow the instructions to do that to complete this installation. Click on the **Finish** button to close the **Windows Components Wizard** when this installation is finished. You may need to contact the administrator at your college to get this system **CD** if you do not have one. You must reboot your computer to make this installation complete.

As you know, the **.NET Framework** includes two **Data Providers** for accessing enterprise databases: the **.NET Framework Data Provider for OLE DB** and the **.NET Framework Data Provider for SQL Server**. Because there is no significant difference between the **Microsoft Access** database and the **SQL Server** database, in this chapter we only use the **SQL Server** database and the **Oracle** database as our target databases to illustrate how to select, display, and manipulate data against our sample database through the **Web** pages.

This chapter is organized as follows:

1. Develop **ASP.NET Web** application to select and display data from the **Microsoft SQL Server** database.

2. Develop ASP.NET Web application to insert data into the Microsoft SQL Server database.
3. Develop ASP.NET Web application to update and delete data against the Microsoft SQL Server database.
4. Develop ASP.NET Web application to select and manipulate data against the Microsoft SQL Server database using LINQ to SQL query.
5. Develop ASP.NET Web application to select and display data from the Oracle database.
6. Develop ASP.NET Web application to insert data into the Oracle database.
7. Develop ASP.NET Web application to update and delete data against the Oracle database.

Let's start with the first one in this list to create and build our ASP.NET Web application.

8.3 DEVELOP ASP.NET WEB APPLICATION TO SELECT DATA FROM SQL SERVER DATABASES

Let's start a new ASP.NET Web application project SQLWebSelect to illustrate how to access and select data from the database via the Internet. Open the Visual Studio.NET and click on the File|New Web Site to create a new ASP.NET Web application project. On the opened New Web Site dialog box, which is shown in Figure 8.5, keep the default template ASP.NET Web Site selected and the default content of Location box unchanged. Select Visual C# from the Language box and then enter the project name SQLWebSelect into the box that is next to the Browse button, as shown in Figure 8.5.

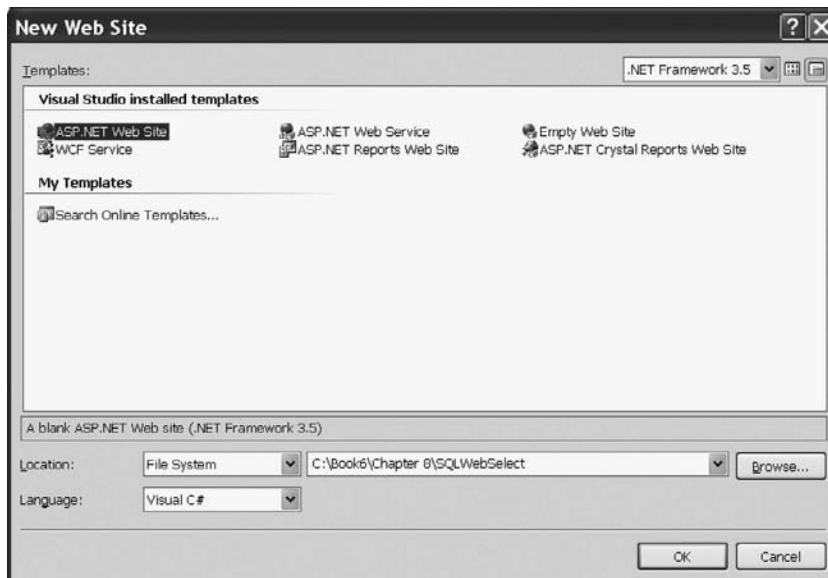


Figure 8.5 Opened Template dialog box.

You can place your new project in any folder you like on your computer. In our case, we place it in the folder `C:\Book 6\Chapter 8`. Click on the OK button to create this new Web application project.

On the opened new project, the default Web form is named `Default.aspx`, and it is located at the Solution Explorer window. This is the Web form that works as a user interface on the server side. Now let's perform some modifications to this form to make it our LogIn form page.

8.3.1 Create the User Interface—LogIn Form

Right-click on this `Default.aspx` item and select the **Rename** item from the pop-up menu and change the name of this Web Form to `LogIn.aspx` since we want to use this default page as our LogIn page.

Three buttons are located at the bottom of this window: **Design**, **Split**, and **Source**. The **Design** button is used to open the Web form page to allow users to insert any control onto that page. The **Source** button is used to open the Source file that basically is an HTML file that contains the related codes for all the controls you added into this Web Form in the HTML format. The **Split** button is used to divide the full window into two parts: the **Design** view and **Source** view. Compared with the codes in the code-behind page, the difference between them is that the Source file is used to describe all controls you added into the Web Form in HTML format, but the code-behind page is used to describe all controls you added into the Web form in Visual C#.NET code format.

Note that the code line is inside the code body: `<form id="form1" runat="server">`. This coding line indicates that this Web form will be run at the server side as the project runs.

Now let's click on the **View Designer** button from the Solution Explorer window to open the **Design** view to design our Web form window.

Unlike the Windows-based application, by default the user interface in the Web-based application has no background color. You can modify the Web form by adding another Style Sheet and format the form as you like. Also if you want to change some styles such as the header and footer of the form applied to all of your pages, you can add a Master Page to do that. But in this project we prefer to use the default window as our user interface and each page in our project has a different style.

We need to add the controls into our LogIn user interface or Web page as shown in Table 8.1. Note that there is no **Name** property available for any control in the Web form object, instead the property **ID** is used to replace the **Name** property and it works as a unique identifier for each control you added into the Web form.

Another difference with the Windows-based form is that when you add these controls into our Web form, first you must locate a position for the control to be added using the **Space** key and the **Enter** key on your keyboard in the Web form, and then pick up a control from the Toolbox window and drag it to that location. You cannot pick and drag a control to a random location in this Web form, and this is a significant difference between the Windows-based form and the Web-based form windows. Your finished user interface should match the one shown in Figure 8.6.

Table 8.1 Controls for the LogIn Form

Type	ID	Text	TabIndex	BackColor	TextMode	Font
Label	Label1	Welcome to CSE DEPT	0	#E0E0E0		Bold/Large
Label	Label2	User Name	1			Bold/Small
Textbox	txtUserName		2			
Label	Label3	Pass Word	3			Bold/Small
Textbox	txtPassWord		4		Password	
Button	cmdLogIn	LogIn	5			Bold/Medium
Button	cmdCancel	Cancel	6			Bold/Medium

**Figure 8.6** Finished LogIn Web form.

Before we can add the codes into the code-behind page in response to the controls to perform the logon process, first we must run the project to allow the web.config file to recognize those controls added into the Web form. Click on the **Start Debugging** button on the toolbar to run our project. Click on OK to a prompted window to add a Web.config file with debugging enabled. Your running Web page should match the one shown in Figure 8.6. Click on the Close button located at the upper-right corner of the form to close this page.

Now let's develop the codes to access the database to perform the logon process.

8.3.2 Develop Codes to Access and Select Data from Database

Open the code-behind page by clicking on the View Code button from the Solution Explorer window. First, we need to add an SQL Server data provider-related namespace as we did for those projects in the previous chapters. Add the following namespace to the top of this code window to involve the namespace of the SQL Server Data Provider:

```

.....
A using System.Data.SqlClient;
public partial class _Default : System.Web.UI.Page
B {
    public SqlConnection sqlConnection;
    protected void Page_Load(object sender, EventArgs e)
C {
    string sqlString = "Server=localhost;Data Source=.\SQLEXPRESS;" +
        "Database=C:\\database\\SQLServer\\CSE_DEPT.mdf;Integrated Security=SSPI";
D
    sqlConnection = new SqlConnection(sqlString);
E Application["sqlConnection"] = sqlConnection; //define a global connection object
F
    if (sqlConnection.State == ConnectionState.Open)
        sqlConnection.Close();
G
    sqlConnection.Open();
H
    if (sqlConnection.State != ConnectionState.Open)
        Response.Write("<script>alert('Database connection is Failed')</script>");
    }
}

```

Figure 8.7 Coding for the Page_Load method.

using System.Data.SqlClient;

Next we need to create a class or field-level variable, `sqlConnection`, for our connection object. Enter the following code under the class header:

public SqlConnection sqlConnection;

This connection object will be used by all Web forms in this project later.

Now we need to perform the coding for the `Page_Load()` method, which is similar to the `Form_Load()` method in the Windows-based application. Open this event method and enter the codes shown in Figure 8.7 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** An SQL Server data provider related namespace is added into this project since we need to use those data components to perform data actions against our sample SQL Server database later.
- B.** A class-level Connection object is declared first, and this object will be used by all Web forms in this project later to connect to our sample database.
- C.** As we did for the `Form_Load()` method in the Windows-based applications, we need to perform the database connection job in this `Page_Load()` method. A connection string is created with the database server name, database name, and security mode.
- D.** A new database Connection object is created with the connection string as the argument.
- E.** The Connection object `sqlConnection` is added into the Application state function, and this object can be used by any pages in this application by accessing this Application state function later. Unlike global variables in the Windows-based applications, one cannot access a class variable by prefixing the form's name before the class variable declared in that form from other pages. In the Web-based application, the Application state function is a good place to store any global variable. In ASP.NET Web application, the Application state is stored in an instance of the `HttpApplicationState` class, which can be accessed

through the Application property of the HttpContext class in the server side and is faster than storing and retrieving information in a database.

- F.** First, we need to check whether this database connection has been done. If it has, we need first to disconnect this connection by using the Close() method.
- G.** Then we can call the Open() method to set up the database connection.
- H.** By checking the database connection state property, we can confirm the connection we did. If the connection state is not equal to Open, which means that the database connection has failed, a warning message is displayed and the procedure is exited.

One significant difference in using the Message box to display some debugging information in the Web form is that you cannot use a Message box as you did in the Windows-based applications. In the Web form development, no Message box is available, and you can only use the Javascript alert() method to display a Message box in ASP.NET. Two popular objects are widely utilized in the ASP.NET Web applications: The Request and the Response objects. The ASP Request object is used to get information from the user, and the ASP Response object is used to send output to the user from the server. The Write() method of the Response object is used to display the message sent by the server. You must add the script tag <script>.....</script> to indicate that the content is written in Javascript language.

Now let's perform the coding for the LogIn button's Click method. The function of this piece of coding is to access the LogIn table located in our sample SQL Server database based on the username and password entered by the user to try to find the matched logon information. Currently, since we have not created our next page—Selection page—we just display a Message box to confirm the success of the logon process if it is. Click on the View Design button from the Solution Explorer window and then double-click on the LogIn button to open its Click method. Enter the codes shown in Figure 8.8 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** An SQL query statement is declared first since we need to use this query statement to retrieve the matched username and password from the LogIn table. Because this query statement is relatively long, we split it into two substrings.
- B.** All data objects related to the SQL Server Data Provider are created here, such as the Command object and DataReader object.
- C.** The Command object is initialized and built by assigning it with the Connection object, CommandType, and Parameters collection properties of the Command class. The Add() method is utilized to add two actual dynamic parameters to the Parameters collection of the Command class.
- D.** The ExecuteReader() method of the Command class is executed to access the database, retrieve the matched username and password, and return them to the DataReader object.
- E.** If the HasRows property of the DataReader is true, at least one matched username and password has been found and retrieved from the database. A successful message is created and sent back from the server to the client to display it in the client browser.
- F.** Otherwise, no matched username or password has been found from the database, and a warning message is created and sent back to the client and displayed in the client browser.
- G.** The used objects such as the Command and the DataReader are released.

_Default	cmdLogIn_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p> <p>F</p> <p>G</p>	<pre>protected void cmdLogIn_Click(object sender, EventArgs e) { string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "; cmdString += "WHERE (user_name=@name) AND (pass_word=@word)"; SqlCommand sqlCommand = new SqlCommand(); SqlDataReader sqlReader; sqlCommand.Connection = sqlConnection; sqlCommand.CommandType = CommandType.Text; sqlCommand.CommandText = cmdString; sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = txtUserName.Text; sqlCommand.Parameters.Add("@word", SqlDbType.Char, 8).Value = txtPassWord.Text; sqlReader = sqlCommand.ExecuteReader(); if (sqlReader.HasRows == true) { Response.Write("<script>alert('LogIn is successful!')</script>"); } else Response.Write("<script>alert('No matched username/password found!')</script>"); sqlCommand.Dispose(); sqlReader.Close(); }</pre>

Figure 8.8 Coding for the LogIn button's Click method.

_Default	cmdCancel_Click()
<p>A</p> <p>B</p>	<pre>protected void cmdCancel_Click(object sender, EventArgs e) { if (sqlConnection.State == ConnectionState.Open) sqlConnection.Close(); Response.Write("<script>window.close()</script>"); }</pre>

Figure 8.9 Coding for the Cancel button's Click method.

Next let's make the coding for the Cancel button's Click method. The function of this method is to close the current Web page if one clicks on this Cancel button, which means that the user wants to terminate the ASP.NET Web application. Double-click on the Cancel button from the Design View of the LogIn Form to open this method and enter the codes shown in Figure 8.9 into this method.

The function of this piece of code is:

- A.** First, we need to check whether the database is still connected to our Web form. If it is, we need to close this connection before we can terminate our Web application.
- B.** The server sends back a command with the Response object's method Write() to issue a Javascript statement window.close() to close the Web application.

At this point, we have finished developing the codes for the LogIn Web form. Before we can run the project to test our Web page, we need to add some data validation functions in the client side to reduce the burden of the server.

Table 8.2 Validation Controls

Validation Control	Functionality
RequiredFieldValidator	Validate whether the required field has valid data (not blank).
RangeValidator	Validate whether a value is within a given numeric range. The range is defined by the MaximumValue and MinimumValue properties provided by users.
CompareValidator	Validate whether a value fits a given expression by using the different Operator property such as 'equal', 'greater than', 'less than' and the type of the value, which is setting by the Type property.
CustomValidator	Validate a given expression using a script function. This method provides the maximum flexibility in data validation but one needs to add a function to the Web page and sends it to the server to get the feedback from it.
RegularExpressionValidator	Validate whether a value fits a given regular expression by using the ValidationExpression property, which should be provided by the user.

8.3.3 Validate Data on Client Side

As we mentioned in Section 8.2.3, in order to reduce the burden of the server, we should make every effort to perform the data validation on the client side. In other words, before we can send requests to the server, we need to make sure that the information to be sent to the server should be as correct as possible. ASP.NET provides some tools to help us to complete this data validation. These tools include five validation controls that are shown in Table 8.2. All of these five controls are located at the Validation tab in the Toolbox window in Visual Studio.NET environment.

Here we want to use the first control, **RequiredFieldValidator**, to validate our two textboxes, `txtUserName` and `txtPassWord`, in the LogIn page to make sure that both are not empty when the user clicks on the LogIn button as the project runs.

Open the Design View of the LogIn Web form, go to the Toolbox window, and click on the Validation tab to expand it. Drag the **RequiredFieldValidator** control from the Toolbox window and place it next to the username textbox. Set the following properties to this control in the property window:

- ErrorMessage UserName is required
- ControlToValidate txtUserName

Perform the similar dragging and placing operations to place the second **RequiredFieldValidator** just next to the password textbox. Set the following properties for this control in the property window:

- ErrorMessage PassWord is required
- ControlToValidate txtPassWord

Your finished LogIn Web form should match the one shown in Figure 8.10.

Now run our project to test this data validation by clicking on the Start Debugging button, without entering any data into two textboxes, and then click on the LogIn button. Immediately two error messages, which are created by the **RequiredFieldValidators**, are displayed to ask users to enter these two pieces of information. After entering the user-name and password, click on the LogIn button again; a successful login message is displayed. So you can see how the **RequiredFieldValidator** works to reduce the processing load for the server.

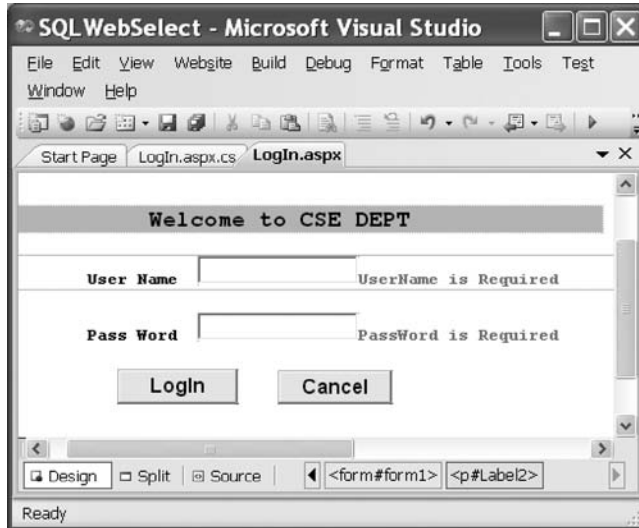


Figure 8.10 Adding the data validation—RequiredFieldValidator.

One good thing always brings some bad thing, which is true to our project, too. After the `RequiredFieldValidator` is added into our Web page, the user cannot close the page by clicking on the `Cancel` button if both username and password textboxes are empty. This is because the `RequiredFieldValidator` is performing the validation checking and no further action can be taken by the Web page until both textboxes are filled with some valid information. Therefore, if you want to close the Web page now, you have to enter a valid username and password, and then you can close the page by clicking on the `Cancel` button.

8.3.4 Create Second User Interface—Selection Page

Now let's continue to develop our Web application by adding another Web page, the Selection page. As we did in previous chapters, after the logon process is finished, the next step is to allow users to select different functions from the Selection form to perform the associated database actions.

The function of this Selection page is to allow users to visit different pages to perform the different database actions such as selecting, inserting, updating, or deleting data in the the database via the different tables by selecting the different items. Therefore, this Selection page needs to perform the following operations:

1. Provide and display all available selections to allow users to select them.
2. Open the associated page based on the users' selection.

Now let's build this page. To do that, we need to add a new Web page. Right-click on the project icon from the Solution Explorer window and select the `Add New Item` from the pop-up menu. On the opened window, keep the default `Template Web Form` selected, and enter `Selection.aspx` into the `Name` box as the name for this new page,

Table 8.3 Controls for the Selection Form

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	Make Your Selection:	0	#E0E0E0	Bold/Large
DropDownList	ComboSelection		1		
Button	cmdSelect	Select	2		Bold/Medium
Button	cmdExit	Exit	3		Bold/Medium

**Figure 8.11** Finished Selection page.

and then click on the Add button to add this page into our project. On the opened Web form, add the controls listed in Table 8.3 into this page.

As we mentioned in the last section, before you pick up those controls from the Toolbox window and drag them into the page, you must first use the Space or the Enter keys on the keyboard to locate the positions on the page for those controls. Your finished Selection page should match the one shown in Figure 8.11.

Next let's create the codes for this Selection page to allow users to select the different page to perform the associated data actions.

8.3.5 Develop Codes to Open Other Page

First, let's run the Selection page to build the Web configuration file. Click on the Start Debugging button to run this page, and then click on the Close button located at the upper-right corner of the page to close it.

Click on the View Code button from the Solution Explorer window to open the code page for the Selection Web form. First, let's add an SQL Data Provider–related namespace to the top of this page to provide a reference to all data components of the SQL Data Provider:

```
using System.Data.SqlClient;
```

Then enter the codes shown in Figure 8.12 into the Page_Load() method to add all selection items into the combobox control ComboSelection to allow users to make their selection as the project runs.

Selection	Page_Load()
<pre>protected void Page_Load(object sender, EventArgs e) { ComboSelection.Items.Add("Faculty Information"); ComboSelection.Items.Add("Course Information"); ComboSelection.Items.Add("Student Information"); }</pre>	

Figure 8.12 Coding for the Page_Load method of the Selection page.

Selection	cmdSelect_Click()
<pre>protected void cmdSelect_Click(object sender, EventArgs e) { if (ComboSelection.Text == "Faculty Information") Response.Redirect("Faculty.aspx"); else if (ComboSelection.Text == "Student Information") Response.Redirect("Student.aspx"); else if (ComboSelection.Text == "Course Information") Response.Redirect("Course.aspx"); }</pre>	

Figure 8.13 Coding for the Select button's Click method.

The function of this piece of code is straightforward. Three pieces of CSE Dept-related data are added into the combobox `ComboSelection` by using the `Add()` method, and these pieces of data will be selected by the user as the project runs.

Next we need to create the codes for two buttons' Click methods. First, let's do the coding for the Select button. Click on the View Designer button from the Solution Explorer window to open the Selection Web form, and then double-click on the Select button to open its method. Enter the codes shown in Figure 8.13 into this method.

The function of this piece of code is easy. Based on the information selected by the user, the related Web page is opened by using the server's `Response` object, that is, by using the `Redirect()` method of the server's `Response` object. These three pages will be created and discussed in the following sections.

Finally let's take care of the coding for the Exit button's Click method. The function of this piece of code is to close the database connection and close the Web application. Double-click on the Exit button from the Design View of the Selection page to open this method. Enter the codes shown in Figure 8.14 into this method.

First, we need to check if the database is still connected to our application. If it is, the global connection object stored in the Application state is activated with the `Close()` method to close the database connection. Then the `Write()` method of the server `Response` object is called to close the Web application. A key point is that the Application state function stores an object, the Connection object in this case. In order to access and use that Connection object stored in the Application global function, a casting (`SqlConnection`) must be clearly prefixed before that object; therefore, a two-layer parenthesis is used to complete this casting. Otherwise a compiling error will be encountered since the compiler cannot recognize and convert the general object stored in the Application state function to a specific Connection object.

Selection	▼	cmdExit_Click()	▼
<pre>protected void cmdExit_Click(object sender, EventArgs e) { if (((SqlConnection)Application["sqlConnection"]).State == ConnectionState.Open) ((SqlConnection)Application["sqlConnection"]).Close(); Response.Write("<script>window.close()</script>"); } </pre>			

Figure 8.14 Coding for the Exit button's Click method.

Now we have finished the coding for the Selection page. Before we can run the project to test this page, we need to do some modifications to the coding in the LogIn button's Click method in the LogIn page to allow the application to switch from the LogIn page to the Selection page as the LogIn process is successful.

Open the LogIn page and the LogIn button's Click method, and replace the code body located inside the `if` block:

```
Response.Write("<script>alert('LogIn is successful!')</script>");
```

with the following code:

```
Response.Redirect("Selection.aspx");
```

In this way, as long as the logon process is successful, the next page, the Selection page, will be opened by executing the `Redirect()` method of the server `Response` object. The argument of this method is the URL address of the Selection page. Since the Selection page is located at the same application as the LogIn page, a direct page name is used.

Now let's run the application to test two pages. Make sure that the LogIn page is the starting page for our application. To do that, right-click on the `LogIn.aspx` from the Solution Explorer window and select the item **Set As Start Page** from the pop-up menu. Click the Start Debugging button to run our project. Enter the suitable username and password such as `ybai` and `reback` to the username and password boxes; then click on the LogIn button. The Selection page is displayed if the logon process is successful, as shown in Figure 8.15. Click on the Exit button to close our application. Now let's begin to develop our next page, the Faculty page.

8.3.6 Create Third User Interface—Faculty Page

Right-click on our project folder from the Solution Explorer window and select the **Add New Item** from the pop-up menu. On the opened dialog box, keep the default **Template Web Form** selected, and then enter `Faculty.aspx` into the **Name** box as the name for this new page, and click on the **Add** button to add this new page into our project. On the opened Web form, add the controls shown in Table 8.4 into this page.

As we mentioned in the last section, before you can drag those controls from the Toolbox window and place them into the page, you must first use the **Space** or the **Enter** keys on the keyboard to locate the positions on the page for those controls. You cannot just place a control in a random position on the form as you did in the Windows-based applications since the Web-based applications have special layout requirements.



Figure 8.15 Running status of the second page—Selection page.

Table 8.4 Controls for the Faculty Form

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	CSE DEPT Faculty Page	0	#E0E0E0	Bold/ Large
Image	PhotoBox		22		
Label	Label2	Faculty Name	1		Bold/ Small
DropDownList	ComboName		2		
Label	Label3	Faculty ID	3		Bold/ Small
TextBox	txtID		4		
Label	Label4	Name	5		Bold/ Small
TextBox	txtName		6		
Label	Label5	Title	7		Bold/ Small
TextBox	txtTitle		8		
Label	Label6	Office	9		Bold/ Small
TextBox	txtOffice		10		
Label	Label7	Phone	11		Bold/ Small
TextBox	txtPhone		12		
Label	Label8	College	13		Bold/ Small
TextBox	txtCollege		14		
Label	Label9	Email	15		Bold/ Small
TextBox	txtEmail		16		
Button	cmdSelect	Select	17		Bold/ Small
Button	cmdInsert	Insert	18		Bold/ Small
Button	cmdUpdate	Update	19		Bold/ Small
Button	cmdDelete	Delete	20		Bold/ Small
Button	cmdBack	Back	21		Bold/ Small

One important point to note is the position of the Image control on the page form. After you drag an Image control from the Toolbox window and place it into the page window, you should set the `ImageAlign` property to `Left`. In this way, we can continue to add all other controls such as labels and textboxes to this Web page without any overlapping problem.

Now you can enlarge this Image and place it on the left side of this Web page. Your finished Faculty page should match the one shown in Figure 8.16.

Although we have added five buttons into this Faculty page, in this section we only take care of the Select and the Back buttons, that is, two buttons' Click methods, since

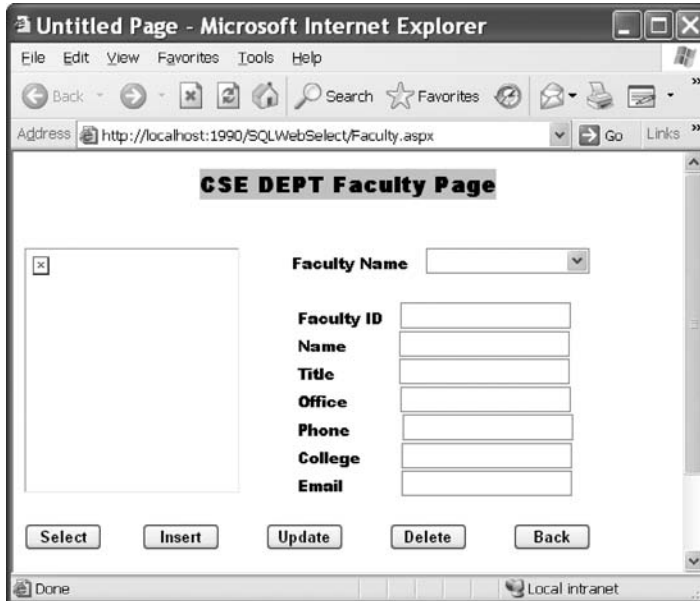


Figure 8.16 Finished Faculty page.

we want to discuss how to retrieve the queried data from our database and display them in this Faculty page. Other buttons will be used in the following sections later. Now let's begin to develop the codes for the Faculty page.

8.3.7 Develop Codes to Select Desired Faculty Information

First, let's run the project to build the configuration file `web.config` to configure all controls we just added into the Faculty page. Click on the Start Debugging button to run the project, and enter the suitable username and password to open the Selection page. Select the Faculty Information item from this page to open the Faculty page. Click on the Close button located at the upper-right corner of this page to close the project.

Open the code page of the Faculty form, and, as we did before, let's first add an SQL Data Provider–related namespace to the top of this page to provide a reference to all data components of the SQL Data Provider:

```
using System.Data.SqlClient;
```

The coding for this page can be divided into three parts: Coding for the `Page_Load()` method, coding for the Select button's Click method, and coding for other methods. First, let's take care of the coding for the `Page_Load()` method.

8.3.7.1 Develop Codes for Page_Load Method

In the opened code page, open the `Page_Load()` method and enter the codes shown in Figure 8.17 into this method. Let's take a closer look at this piece of code to see how it works.

```

Faculty
Page_Load()
.....
A using System.Data.SqlClient;
public partial class Faculty : System.Web.UI.Page
{
B private TextBox[] FacultyTextBox = new TextBox[7];
protected void Page_Load(object sender, EventArgs e)
{
C if (((SqlConnection)Application["sqlConnection"]).State != ConnectionState.Open)
((SqlConnection)Application["sqlConnection"]).Open();
D if (!IsPostBack)
{
ComboName.Items.Add("Ying Bai");
ComboName.Items.Add("Satish Bhalla");
ComboName.Items.Add("Black Anderson");
ComboName.Items.Add("Steve Johnson");
ComboName.Items.Add("Jenney King");
ComboName.Items.Add("Alice Brown");
ComboName.Items.Add("Debby Angles");
ComboName.Items.Add("Jeff Henry");
}
}
}

```

Figure 8.17 Coding for the Page_Load method.

- A.** An SQL Server Data Provider–related namespace is added into this namespace area since we need to use some SQL Server data components located in that namespace.
- B.** A field-level textbox array is created first since we need this array to hold six pieces of faculty information and display them in these six textboxes later.
- C.** Before we can perform the data actions against the database, we need to make sure that a valid database connection is set to allow us to transfer data between our project and the database. An Application state, which is used to hold our global connection object variable, is utilized to perform this checking and connecting to our database if it has not been connected.
- D.** As the project runs, each time as the user clicks on the Select button to perform a data query, a request is sent to the database server and the Web server (it can be the same server as the database server). Then the Web server will post back a refreshed Faculty page to the client when it received this request (`IsPostBack = true`). Each time this happens, the `Page_Load()` method will be activated again, and the duplicated eight faculty members are attached to the end of the combobox control `ComboName` again. To avoid this duplication, we need to check the `IsPostBack` property of the page and add eight faculty members into the combobox control only one time when the project starts (`IsPostBack = false`). Refer to Section 8.3.8.1 for more detailed discussion about the `AutoPostBack` property.

Next we need to develop the coding for the Select button's Click method to perform the data query actions against the database.

8.3.7.2 Develop Codes for Select Button Method

The function of this coding is to make queries to the database to retrieve the faculty information based on the selected faculty member by the user from the combobox control

```

Faculty
cmdSelect_Click()
protected void cmdSelect_Click(object sender, EventArgs e)
{
A   string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty ";
B   cmdString += "WHERE faculty_name LIKE @name";
   SqlCommand sqlCommand = new SqlCommand();
   SqlDataReader sqlDataReader;
C   sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
   sqlCommand.CommandType = CommandType.Text;
   sqlCommand.CommandText = cmdString;
D   sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = ComboName.Text;
E   string strName = ShowFaculty(ComboName.Text);
F   sqlDataReader = sqlCommand.ExecuteReader();
G   if (sqlDataReader.HasRows == true)
   FillFacultyReader(sqlDataReader);
H   else
   Response.Write("<script>alert('No matched faculty found!')</script>");
I   sqlDataReader.Close();
   sqlCommand.Dispose();
}

```

Figure 8.18 Coding for the Select button's Click method.

ComboName, and then display that retrieved information in the six textbox controls on the Faculty page.

Open this Select button's Click method by double-clicking on this button from the Design View of the Faculty Form, and enter the codes shown in Figure 8.18 into this method.

Let's take a look at this piece of code to see how it works.

- A.** The query string that contains a SELECT statement is declared here since we need to use this string as our command text. The dynamic parameter of this query is the @name, which is the selected faculty name by the user as the project runs.
- B.** All data components such as the Command and DataReader objects are declared here since we need to use them to perform the data query later.
- C.** The Command object is initialized by assigning the associated components to it. These components include the global Connection object that is stored in the Application state function, the CommandType, and the CommandText properties.
- D.** The Parameter object is initialized by assigning the dynamic parameter's name and value to it.
- E.** The user-defined ShowFaculty() method is called to display the selected faculty photo in the Image control on the Faculty page. We will develop this method in the next section.
- F.** The ExecuteReader() method of the Command object is called to execute the query command to retrieve the selected faculty information, and assign it to the DataReader object.
- G.** By checking the HasRows property of the DataReader, we can determine whether this query is successful or not. If this property is greater than zero, which means that at least one row is retrieved from the Faculty table in the database and therefore the query is successful, a user-defined method FillFacultyReader(), which we will develop in the next part, is called to fill the six textboxes on the Faculty page with the retrieved faculty information.

- H. Otherwise if the `HasRows` property is equal to zero, which means that no row has been retrieved from the database, the query has failed. A warning message is displayed to the client by calling the `Write()` method of the server `Response` object.
- I. All data components used for this data query are released after this query.

At this point we have finished the coding for the `Select` button's `Click` method.

8.3.7.3 Develop Codes for Other Methods

Next let's take care of the coding for other methods in this `Faculty` page, which includes the coding for the following methods:

1. User-defined `ShowFaculty()` method
2. User-defined `FillFacultyReader()` method
3. User-defined `MapFacultyTable()` method
4. `Back` button's `Click` method

The third method, `MapFacultyTable()`, is used and called by the second user-defined `FillFacultyReader()` method in our project. Now let's discuss the coding for these methods one by one.

First, let's see the coding for the `ShowFaculty()` method. The function of this method is to get the matched faculty photo from the default location based on the input faculty name and display it in the `Image` control on the `Faculty` page. The so-called default location for the photo file is exactly the current ASP.NET Web application folder. In our case, it is `C:\Book6\Chapter 8\SQLWebSelect`. You must place all faculty photo files in this location before you can run this project to pick up the desired faculty information from the database and display it in the `Faculty` page. Of course, you can place your faculty photo files in any folder in your computer. In that case, you must provide the full name for the faculty photo, including the drive, path, and the name of the photo file. In this project, to make it simple, we used the default folder to store our faculty photo files.

Open the code page of the `Faculty` page, and type and create this method, which is shown in Figure 8.19. Let's take a look at the coding for this method to see how it works.

- A. A local string variable `FacultyImage` is created, and it is used to hold the name of the matched faculty photo file.
- B. A `switch...case` structure is utilized to find the matched faculty photo file based on the input faculty name. The name of the matched faculty photo file is assigned to the local string variable `FacultyImage` if it is found.
- C. The name of the matched faculty photo file is assigned to the `ImageUrl` property of the `Image` control to display that photo if the `FacultyImage` is not equal to "No Match".
- D. Otherwise, a warning message is displayed in the client using the `Write()` method of the server `Response` object.
- E. The `FacultyImage` is returned to the calling method.

One significant difference in displaying an image between the Windows-based and the Web-based application is that the `ImageUrl` property, which belongs to the control `System.Web.UI.WebControls.Image`, is utilized to access the matched faculty photo file and only the name of an image file is needed to display the associated image in the

Faculty	ShowFaculty()
<div style="font-family: monospace; font-size: 12px; padding: 5px;"> <pre> private string ShowFaculty(string fName) { string FacultyImage; switch (fName) { case "Black Anderson": FacultyImage = "Anderson.jpg"; break; case "Ying Bai": FacultyImage = "Bai.jpg"; break; case "Satish Bhalla": FacultyImage = "Satish.jpg"; break; case "Steve Johnson": FacultyImage = "Johnson.jpg"; break; case "Jenney King": FacultyImage = "King.jpg"; break; case "Alice Brown": FacultyImage = "Brown.jpg"; break; case "Debby Angles": FacultyImage = "Angles.jpg"; break; case "Jeff Henry": FacultyImage = "Henry.jpg"; break; default: FacultyImage = "No Match"; break; } if (FacultyImage != "No Match") PhotoBox.ImageUrl = FacultyImage; else Response.Write("<script>alert('No matched faculty image found!')</script>"); return FacultyImage; } </pre> </div>	

Figure 8.19 Coding for the ShowFaculty method.

Web-based application. In the Windows-based application, a `System.Drawing()` method must be used to display an image based on the image file's name.

The next method is the `FillFacultyReader()`. Open the code page of the Faculty Web form, type the codes shown in Figure 8.20 to create this user-defined method inside the Faculty class.

The function of this method is to pick up each data column from the retrieved data that is stored in the `DataReader` and assign it to the associated textbox on the Faculty page to display it.

Let's see how this piece of code works.

- A.** A loop counter `intIndex` is declared.
- B.** Seven instances of the object array, textbox array, are created and initialized. These seven objects are mapped to seven columns in the Faculty table in our sample database.

```

Faculty FillFacultyReader()
private void FillFacultyReader(SqlDataReader FacultyReader)
{
A   int intIndex = 0;
B   for (intIndex = 0; intIndex <= 6; intIndex++) //Initialize the object array
      FacultyTextBox[intIndex] = new TextBox();
C   MapFacultyTable(FacultyTextBox);
D   while (FacultyReader.Read())
      {
E       for (intIndex = 0; intIndex <= FacultyReader.FieldCount - 1; intIndex++)
          FacultyTextBox[intIndex].Text = FacultyReader.GetString(intIndex);
      }
}

```

Figure 8.20 Coding for the FillFacultyReader method.

- C. Another user-defined method MapFacultyTable() is called to set up the correct mapping between the seven textbox controls on the Faculty page window and seven columns in the query string cmdString.
- D. A while loop is executed as long as the loop condition Read() method is true, which means that a valid data is read out from the DataReader. This method will return a false if no valid data can be read out from the DataReader, which means that all data has been read out. In this application, in fact, this while loop is only executed once since we have only one row (one record) read out from the DataReader.
- E. A for loop is utilized to pickup each data read out from the DataReader object, and assign each of them to the associated textbox control on the Faculty page window. The intIndex is used here to identify each item from the DataReader.

The detailed codes for the user-defined MapFacultyTable() method are shown in Figure 8.21. The function of this method, as we mentioned, is to set up a correct mapping relationship between seven textboxes in the textbox array on the Faculty page and the seven data columns in the query string since the order in which the textboxes are displayed in the Faculty page may not be identical with the order of the data columns in the query string cmdString we created at the beginning of the Select button's Click method.

Open the code page of the Faculty Web form, type the codes shown in Figure 8.21 to create this method inside the Faculty class.

The order of the seven textboxes on the right-hand side of the equals operator should be equal to the order of the queried columns in the query string—cmdString. By performing this assignment, the seven textbox controls on the Faculty page window have a correct one-to-one relation with the queried columns in the query string cmdString.

Finally, let's take care of the coding for the Back button's click method. The function of this piece of coding is to return to the Selection page as this button is clicked. Double-click on the Back button from the Faculty page window to open this method, and enter the coding line into this method, which is shown in Figure 8.22.

This coding is straightforward and easy to understand. The Redirect() method of the server Response object is executed to direct the client from the current Faculty page back to the Selection page when this button is clicked on by the user; that is, the server resends the Selection page to the client when this button is clicked on and a request is sent to the server.

Faculty	▼	MapFacultyTable()	▼
<pre>private void MapFacultyTable(Object[] fText) { fText[0] = txtID; fText[1] = txtName; fText[2] = txtOffice; //The order must be identical fText[3] = txtPhone; //with the real order in the query string fText[4] = txtCollege; fText[5] = txtTitle; fText[6] = txtEmail; } </pre>			

Figure 8.21 Coding for the MapFacultyTable method.

Faculty	▼	cmdBack_Click()	▼
<pre>protected void cmdBack_Click(object sender, EventArgs e) { Response.Redirect("Selection.aspx"); } </pre>			

Figure 8.22 Coding for the Back button's method.

Now we have finished all coding development for the Faculty page. It is time for us to run the project to test our pages. However, before you can run the project, make sure that you have stored all faculty photo files in the default location. Now click on the Start Debugging button to run our project. Enter the suitable username and password such as `jhenry` and `test` to the LogIn page, and select the Faculty Information from the Selection page to open the Faculty page. Select one faculty member from the combobox control such as `Ying Bai`, and then click on the Select button to retrieve the selected faculty information from the database. All information related to that selected faculty is retrieved and displayed on this Faculty page, as shown in Figure 8.23.

Click on the Back button to return to the Selection page, and then click on the Exit button to terminate our project. So far our Web application is successful.

Next we need to create our last Web page, Course page, and add it into our project to select and display all courses taught by the selected faculty member.

8.3.8 Create Fourth User Interface—Course Page

To create a new Web page and add it to our project, go to the Solution Explorer window and right-click on our project folder, select **Add New Item** from the pop-up menu to open the **Add New Item** dialog box. On the opened dialog box, keep the default template **Web Form** selected, enter `Course.aspx` into the **Name** box as the name for our new page, and click on the **Add** button to add it to our project. On the opened Web form, add the controls shown in Table 8.5 into this page.

As we mentioned before, you cannot place a control in any position on the form as you like. Now we introduce a technology to place a control at your desired position on the page window just as you did for your Windows-based application. The key element



Figure 8.23 Running SQL of the Faculty page.

Table 8.5 Controls on the Course Form

Type	ID	Text	TabIndex	BackColor	Font	AutoPostBack
Panel	Panel1		16	#COCOFF		
Label	Label1	Faculty Name	0		Bold/Small	
DropDownList	ComboName		1			
ListBox	CourseList		17		Bold/Small	True
Panel	Panel2		18	#COCOFF		
Label	Label2	Course Title	2		Bold/Small	
TextBox	txtCourse		3			
Label	Label3	Schedule	4		Bold/Small	
TextBox	txtSchedule		5			
Label	Label4	Classroom	6		Bold/Small	
TextBox	txtClassRoom		7			
Label	Label5	Credit	8		Bold/Small	
TextBox	txtCredit		9			
Label	Label6	Enrollment	10		Bold/Small	
TextBox	txtEnrollment		11			
Button	cmdSelect	Select	12		Bold/ Small	
Button	cmdInsert	Insert	13		Bold/ Small	
Button	cmdUpdate	Update	14		Bold/ Small	
Button	cmdDelete	Delete	15		Bold/ Small	
Button	cmdBack	Back	16		Bold/ Small	

is the Position property for each control you added into the page window. For example, in this Course page, we added two panel controls, one listbox control, five textbox controls, and five button controls. In order to locate those controls at the desired position on the page, you must set up the FormatPosition property to Absolute for all controls added to this page except the label controls.

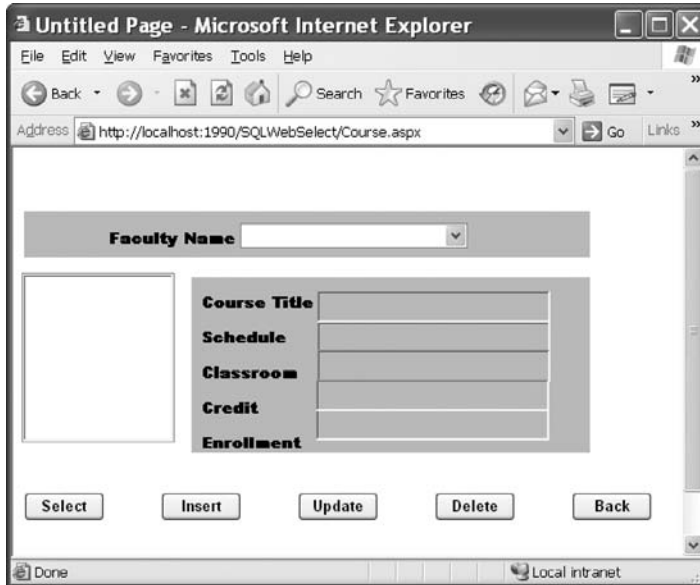


Figure 8.24 Finished Course Web page.

Another issue is the panel size and background color. As you add a new panel into the page, the background color of that new panel is transparent, which means that you cannot see the new added panel. To see that panel, you need to define the size of the panel by performing the following steps:

1. Click on the new added panel to select it and go to the Property window.
2. Set the Height and the Width properties to the desired dimensions in pixels (px).

In this application, set the Height and Width properties for panel1 and panel2 to:

- Panel1: Height 40px Width 500px
- Panel2: Height 148px Width 352px

Your finished Course page should match the one shown in Figure 8.24. Before we can continue to develop the codes for this page, we must emphasize one key point for the listbox control used in the Web-based applications. There is a significant difference between the Windows-based and Web-based applications for the listbox control.

8.3.8.1 AutoPostBack Property of Listbox Control

One important property is the AutoPostBack property for the listbox control CourseList. Unlike the listbox control used in the Windows-based application, a SelectedIndexChanged event will not be created on the server side if the user clicked on and selected an item from the listbox. The reason for that is because the default value for the AutoPostBack property of a listbox control is set to `false` when you add a new listbox to your Web form. This means that even if the user clicked on and changed the item from the listbox, a SelectedIndexChanged event can only be created on the client side, and it cannot be

sent to the server. As you know, the listbox is running on the server side when your project runs. Therefore, no matter how many times you clicked on and changed the items in the listbox at the client side, no event can be created on the server side. Therefore, the project will not respond to your clicking on the listbox control as the project runs.

However, in this project we need to use this `SelectedIndexChanged` event to trigger its event method to perform the course information query. To solve this problem, the `AutoPostBack` property should be set to `true`. In this way, each time you click on an item to select it from the listbox, this `AutoPostBack` property will be set to post back to the server to indicate that the user has interacted with that control.

In this section, we only discuss the coding for the `Select` and the `Back` buttons' click methods to perform the course data query. The operations for the other buttons such as `Insert`, `Update`, and `Delete` will be discussed in the following sections when we perform the data inserting or updating in the database using the Web pages.

Now let's develop the codes for the `Select` and the `Back` buttons' click methods to pick up the course data from the database using the Course Web page.

8.3.9 Develop Codes to Select Desired Course Information

The functions of the Course page are:

1. When the user selects the desired faculty member from the Faculty Name combobox control and clicks on the `Select` button, the IDs of all courses taught by the selected faculty should be retrieved from the database and displayed in the listbox control `CourseList` on the Course page.
2. When the user clicks any `course_id` from the listbox control `CourseList`, the detailed course information related to the selected `course_id` in the listbox will be retrieved from the database and displayed in five textboxes on the Course page form.

Based on the function analysis above, we need to concentrate on the coding for two methods: one is the `Select` button's click method and the second is the `SelectedIndexChanged` event method of the listbox control `CourseList`. The first coding is used to retrieve and display all `course_id` related to courses taught by the selected faculty in the listbox control `CourseList`, and the second coding is to retrieve and display the detailed course information such as course title, schedule, classroom, credit, and enrollment related to the selected `course_id` from the `CourseList`.

The above coding operations can be divided into four parts:

1. Coding for the Course page loading and ending event methods. These methods include the `Page_Load()` and the `Back` button's click method.
2. Coding for the `Select` button's click method.
3. Coding for the `SelectedIndexChanged` event method of the listbox control `CourseList`.
4. Coding for other user-defined methods.

Before we can start the first coding operation, we need to add an SQL Server Data Provider related namespace to the top of the Course page. Open the code window of the Course page and enter the following namespace to the top of that page:

```
using System.Data.SqlClient;
```

```

Course Page_Load()
.....
A using System.Data.SqlClient;
public partial class Course : System.Web.UI.Page
B {
private TextBox[] CourseTextBox = new TextBox[6];
protected void Page_Load(object sender, EventArgs e)
C {
if (((SqlConnection)Application["sqlConnection"]).State != ConnectionState.Open)
((SqlConnection)Application["sqlConnection"]).Open();
D
if (!IsPostBack) //these items can only be added into the combo box in one time
{
ComboName.Items.Add("Ying Bai");
ComboName.Items.Add("Satish Bhalla");
ComboName.Items.Add("Black Anderson");
ComboName.Items.Add("Steve Johnson");
ComboName.Items.Add("Jenney King");
ComboName.Items.Add("Alice Brown");
ComboName.Items.Add("Debby Angles");
ComboName.Items.Add("Jeff Henry");
}
}
}

```

Figure 8.25 Coding for the Page_Load method.

8.3.9.1 Coding for Course Page Loading and Ending Methods

Open the Page_Load() method and enter the codes shown in Figure 8.25 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** The SQL Server Data Provider–related namespace is added into this page since we need to use some data components stored in that namespace to perform the data actions against the Course table in our sample database.
- B.** This coding fragment is similar to one we did for the Faculty Form. Five textbox controls in the Course Form page are used to display the detailed course information that is related to the selected faculty from the Faculty Name combobox. The Course table has seven columns, although we only need five of them, we still set the size of this TextBox array to 6. Each element or each TextBox control in this array is indexed from 0 to 5.
- C.** The function of this code segment is: Before we can perform any data query, we need to check whether a valid database connection is available. Since we created a class-level connection instance in the LogIn page and stored it in the Application state, now we need to check this connection object and reconnect it to the database if our application has not been connected to the database.
- D.** This piece of codes is used to initialize the Faculty Name combobox control, and the Add() method is utilized to add all faculty members into this combobox control to allow users to select one to get the course information as the project runs. Here a potential bug exists for this piece of code. As we mentioned in Section 8.3.8.1, an AutoPostBack property will be set to true whenever the user clicked on and selected an item from the listbox control, and this property will be sent back to the server to indicate that an action has been taken by the user to this listbox. After the server received this property, it will resend a refreshed

Course page to the client. Therefore, the Page_Load() method of the Course page will be triggered and run again as a refreshed Course page is sent back. The result of execution of this Page_Load() method is to attach another copy of all faculty members to the end of those faculty members that have been already added into the combobox control ComboName when the Course page is displayed in the first time. As the number of the item you clicked on from the CourseList box increases, the number of copies of all faculty members will also increase and be displayed the ComboName box. To avoid this duplication, we only need to add all faculty members the first time as the Course page is displayed, but do nothing if any another AutoPostBack property occurs.

The coding for the Back button's click method is similar to that for the Back button in the Faculty page. When this button is clicked on by the user, the Course page should be switched back to the Selection page. The Redirect() method of the server Response object is used to fulfill this switching back function. Double-click on the Back button from the Course page window and enter the following codes into this method:

```
Response.Redirect("Selection.aspx");
```

Let's continue with the coding for the Select button's click method.

8.3.9.2 Coding for Select Button Click Method

As we mentioned at the beginning of this section, the function of this method is: When the user selects the desired faculty member from the Faculty Name combobox control and clicks on the Select button, all `course_id` related to courses taught by the selected faculty should be retrieved from the database and displayed in the listbox control CourseList on the Course page.

Double-click on the Select button from the Course page window to open its Click method, and enter the codes shown in Figure 8.26 into this method.

Let's take a look at this piece of code to see how it works.

```

Course
cmdSelect_Click()
protected void cmdSelect_Click(object sender, EventArgs e)
{
A   string strCourse = "SELECT Course.course_id, Course.course FROM Course JOIN Faculty ";
B   strCourse += "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.faculty_name LIKE @name)";
B   SqlCommand sqlCommand = new SqlCommand();
   SqlDataReader sqlDataReader;
C   sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
   sqlCommand.CommandType = CommandType.Text;
   sqlCommand.CommandText = strCourse;
   sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = ComboName.Text;
D   sqlDataReader = sqlCommand.ExecuteReader();
E   if (sqlDataReader.HasRows == true)
   FillCourseReader(sqlDataReader);
F   else
   Response.Write("<script>alert('No matched course found!')</script>");
G   sqlDataReader.Close();
   sqlCommand.Dispose();
}

```

Figure 8.26 Coding for the Select button's Click method.

- A.** The joined table query string is declared at the beginning of this method. Here two columns are queried. The first one is the `course_id` and the second is the course name. The reason for this is that we need to use the `course_id`, not course name, as the identifier to pick up each course' detailed information from the Course table when the user clicks on and selects the `course_id` from the CourseList box. We use the `course_id` with the course name together in this joined table query, and we will use that `course_id` later. The comparator LIKE is used to replace the original equals symbol for the criteria in the ON clause in the definition of the query string, and this is required by SQL Server database operation. For a more detailed description about the joined table query, refer to Section 5.19.2.5 in Chapter 5.
- B.** Some SQL data objects such as the Command and DataReader are created here. All of these objects should be prefixed by the keyword `sql` to indicate that all those components are related to the SQL Server database.
- C.** The `sqlCommand` object is initialized with the connection string, command type, command text, and command parameter. The parameter's name must be identical with the dynamic nominal name `@name`, which is defined in the query string and is located after the LIKE comparator in the ON clause. The parameter's value is the content of the Faculty Name combobox, which should be entered by the user as the project runs later.
- D.** The `ExecuteReader()` method of the Command class is executed to read back all courses (`course_id`) taught by the selected faculty and assign them to the DataReader object.
- E.** If the `HasRows` property of the DataReader is `true`, which means that at least one row data has been retrieved from the database, the `FillCourseReader()` method is called to fill the `course_id` into the CourseList box.
- F.** Otherwise, this joined query has failed and a warning message is displayed.
- G.** Finally some cleaning is preformed to release objects used for this query.

Now let's take care of the coding for the user-defined `FillCourseReader()` method, which is shown in Figure 8.27. Open the code page of the Course Web form and enter the codes shown in Figure 8.27 to create this method inside the Course class.

Let's see how this piece of code works.

- A.** A local string variable `strCourse` is created, and this variable can be considered an intermediate variable used to temporarily hold the queried data from the Course table.
- B.** The CourseList box is cleaned up before it can be filled by the `course_id`.

```

private void FillCourseReader(SqlDataReader CourseReader)
{
    string strCourse = string.Empty;
    CourseList.Items.Clear();
    while (CourseReader.Read())
    {
        strCourse = CourseReader.GetString(0);    //the 1st column is course_id
        CourseList.Items.Add(strCourse);
    }
}

```

Figure 8.27 Coding for the `FillCourseReader` method.

- C. A `while` loop is utilized to retrieve the first column for each row (`GetString(0)`), whose column index is 0 and the data value is the `course_id`. The queried data is first assigned to the intermediate variable `strCourse`, and then it is added into the `CourseList` box by using the `Add()` method.

Now let's start to develop the codes for the `SelectedIndexChanged` event method of the listbox control `CourseList`. The function of this method is: When the user clicks any `course_id` from the listbox control `CourseList`, the detailed course information related to the selected `course_id` in the listbox such as course title, schedule, credit, classroom, and enrollment will be retrieved from the database and displayed in five textboxes on the `Course` page form.

8.3.9.3 Coding for SelectedIndexChanged Method of Listbox Control

Double-click on the listbox control `CourseList` from the `Course` Web form to open this event method, and then enter the codes shown in Figure 8.28 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. The query string is created with six queried columns such as `course`, `credit`, `classroom`, `schedule`, `enrollment`, and `course_id`. The query criterion is `course_id`, which was selected by the user by clicking on a valid `course_id` from the `CourseList` box control as the project runs.
- B. Two SQL data objects are created, and these objects are used to perform the data operations between the database and our project. All of these objects should be prefixed by the keyword `sql` since in this project we used an SQL Server data provider.
- C. The `sqlCommand` object is initialized with the connection object, command type, command text, and command parameter. The parameter's name must be identical with the dynamic nominal name `@courseid`, which is defined in the query string, exactly after the `LIKE` comparator in the `WHERE` clause. The parameter's value is the `course_id` selected by the user from the `CourseList` box control.

```

protected void CourseList_SelectedIndexChanged(object sender, EventArgs e)
{
    A   string cmdString = "SELECT course, credit, classroom, schedule, enrollment, course_id FROM Course ";
    cmdString += "WHERE course_id LIKE @courseid";
    B   SqlCommand sqlCommand = new SqlCommand();
    SqlDataReader sqlDataReader;

    C   sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
    sqlCommand.CommandType = CommandType.Text;
    sqlCommand.CommandText = cmdString;
    sqlCommand.Parameters.Add("@courseid", SqlDbType.Char).Value = CourseList.SelectedItem;
    D   sqlDataReader = sqlCommand.ExecuteReader();
    E   if (sqlDataReader.HasRows == true)
        FillCourseReaderTextBox(sqlDataReader);
    F   else
        Response.Write("<script>alert('No matched course information found!')</script>");
    G   sqlDataReader.Close();
    sqlCommand.Dispose();
}

```

Figure 8.28 Coding for the `SelectedIndexChanged` event method.

- D. The `ExecuteReader()` method is executed to read back the detailed information for the selected course and assign it to the `DataReader` object.
- E. If the `HasRows` property of the `DataReader` is `true`, which means that at least one row data has been retrieved from the database, the user-defined method `FillCourseReaderTextBox()` is called to enter those pieces of data into five textboxes.
- F. Otherwise, this query has failed and a warning message is displayed.
- G. Finally some cleaning jobs are preformed to release objects used for this query.

The coding for other user-defined methods includes the coding for the user-defined `FillCourseReaderTextBox()` and `MapCourseTable()` methods. The function of the user-defined `MapCourseTable()` method is to set up a one-to-one relationship between each textbox in the Course Form page and each queried column from our database since the order of the textbox in the Course Form page may not be identical to the order of the data column queried from the database based on the query string.

8.3.9.4 Coding for Other User-Defined Methods

First, let's develop the coding for the `FillCourseReaderTextBox()` method. On the opened code page of the Course Web form, enter the codes shown in Figure 8.29 into the Course class to create this user-defined method.

Let's take a look at this piece of code to see how it works.

- A. A loop counter `intIndex` is first created, and it is used for the loop of creation of the textbox object array and the loop of retrieving data from the `DataReader` later.
- B. The first `for` loop is used to create the textbox object array and perform the initialization for those objects.
- C. Another user-defined method `MapCourseTable()` is executed to set up a one-to-one relationship between each textbox control on the Course page and each queried column in the query string. This step is necessary since the distribution order of five textbox controls on the Course page may differ from the column order in the query string.
- D. A `while` and the second `for` loop are used to pick up all five pieces of course-related information from the `DataReader` one by one. The `Read()` method is used as the `while` loop condition. A returned `true` means that a valid data is read out from the `DataReader`,

Course	FillCourseReaderTextBox()
<p>A</p> <p>B</p> <p>C</p> <p>D</p>	<pre>private void FillCourseReaderTextBox(SqlDataReader CourseReader) { int intIndex = 0; for (intIndex = 0; intIndex <= 5; intIndex++) //Initialize the object array CourseTextBox[intIndex] = new TextBox(); MapCourseTable(CourseTextBox); while (CourseReader.Read()) { for (intIndex = 0; intIndex <= CourseReader.FieldCount - 1; intIndex++) CourseTextBox[intIndex].Text = CourseReader.GetValue(intIndex).ToString(); } }</pre>

Figure 8.29 Coding for the `FillCourseReaderTextBox` method.

Course	▼	MapCourseTable()	▼
<pre>private void MapCourseTable(Object[] fCourse) { fCourse[0] = txtCourse; //The order must be identical with fCourse[1] = txtCredit; //the real order in the query string - fCourse[2] = txtClassroom; //cmdString fCourse[3] = txtSchedule; fCourse[4] = txtEnrollment; } </pre>			

Figure 8.30 Coding for the subroutine MapCourseTable.

and a returned `false` means that no valid data has been read out from the `DataReader`. In other words, no more data is available and all data has been read out. The second `for` loop uses the `FieldCount-1` as the termination condition since the index of the first data column is 0, not 1, in the `DataReader` object. Each read-out data is converted to a string and assigned to the associated textbox control in the textbox object array.

The coding for the user-defined `MapCourseTable()` method is shown in Figure 8.30.

The function of this piece of coding is straightforward with no tricks. The order of the textboxes on the right-hand side of the equals operator is aligned with the column order of the query string, `cmdString`, by assigning each column of queried data to each of its partner, the textbox in the textbox object array in this order. A one-to-one relationship between each column of queried data and the associated textbox control on the Course page is built, and the data retrieved from the `DataReader` can be mapped exactly to the associated textbox control and displayed there.

At this point, we have finished all coding developments for the Course Web form. Now let's run the project to test the functionality of this form. Click on the Start Debugging button to run the project. Enter the suitable username and password such as `jhenry` and `test` to the LogIn page, and select the Course Information item from the Selection page to open the Course page. On the opened page, select a faculty member from the combobox control and then click on the Select button to retrieve all courses taught by that faculty and display them in the CourseList box. Your running result should match the one shown in Figure 8.31.

Click on any `course_id` from the CourseList box to select it. Immediately the detailed information related to that selected `course_id` is displayed in the five textboxes, which is shown in Figure 8.32.

Click on the Back button to return to the Selection page, and you can click on any other item from the Selection page to perform the associated information query or you can click on the Exit button to terminate the application. Our Web application is successful.

A complete Web application project `SQLWebSelect`, which is used for data query from the SQL Server database, can be found from the folder `DBProjects\Chapter 8` located at the accompanying ftp site (see Chapter 1).

When running the project, a possible bug is the font size of those five textbox controls in this Course Form page. Sometimes it looks like it does not work to set the Font size property for these textboxes in the Properties window. Instead, you have to do that by going to the `Format!Font` menu item for each textbox.

In the next section, we will discuss how to insert a new record into the SQL Server database via Web page applications.

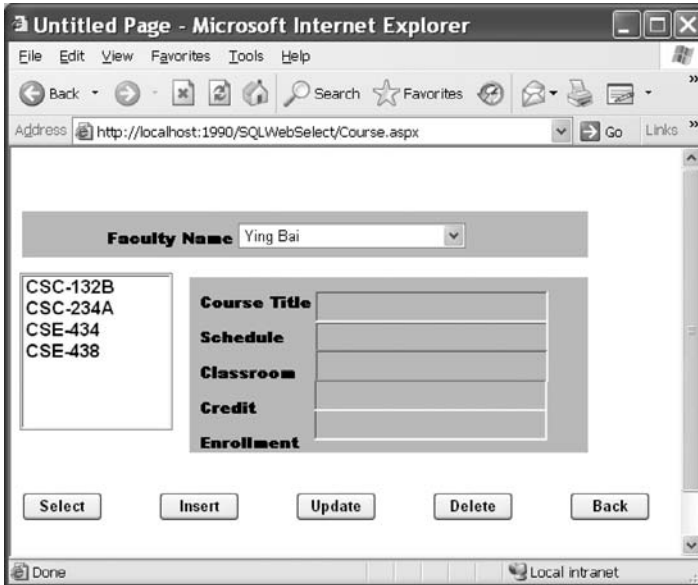


Figure 8.31 Running status of the Course page.

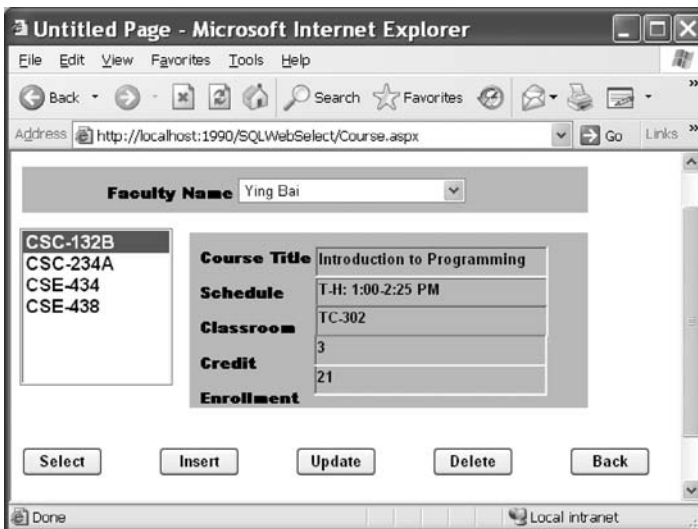


Figure 8.32 Detailed course information.

8.4 DEVELOP ASP.NET WEB APPLICATION TO INSERT DATA INTO SQL SERVER DATABASES

In this section we discuss how to insert a new faculty record into the SQL Server database from the Web page. To do that, we need to create a new Web page called Insert page and add it into our new project. To save time and space, we can modify an existing project,

Table 8.6 Controls for the Insert Form

Type	ID	Text	TabIndex	BackColor	Font
Panel	Panel1		19	#COCOFF	
Label	Label1	Faculty Photo	0		x-small
TextBox	txtPhoto		1		
Panel	Panel2		20	#COCOFF	
Label	Label2	Faculty ID	2		x-small
TextBox	txtID		3		
Panel	Panel3		21	#COCOFF	
Image	PhotoBox		4		
Label	Label3	Name	5		Small
TextBox	txtName		6		
Label	Label4	Title	7		Small
TextBox	txtTitle		8		
Label	Label5	Office	9		Small
TextBox	txtOffice		10		
Label	Label6	Phone	11		Small
TextBox	txtPhone		12		
Label	Label7	College	13		Small
TextBox	txtCollege		14		
Label	Label8	Email	15		Small
TextBox	txtEmail		16		
Button	cmdInsert	Insert	17		Bold/Small
Button	cmdBack	Back	18		Bold/Small

SQLWebSelect developed in the previous section, and make it our new Web application project named SQLWebInsert.

Open Windows Explorer and create a new folder Chapter 8 if you have not done that. Then copy the project SQLWebSelect from the folder DBProjects\Chapter8 located at the accompanying ftp site (see Chapter 1) and paste it to our new folder C:\Chapter 8. Rename this project to SQLWebInsert.

As we mentioned, to add a new record into the database, we need a user interface to do that job. So we need to create and add a new Web page `Insert.aspx` into our new project to perform the inserting data function.

8.4.1 Create New Web Page `Insert.aspx`

Right-click on our new project icon from the Solution Explorer window and select **Add New Item** from the pop-up menu to open the **Add New Item** dialog box. Keep the default template **Web Form** selected and then enter `Insert.aspx` into the **Name** box as the name for this new page. Click on the **Add** button to add this new page into our new project. Add the controls shown in Table 8.6 into this Web page.

The key points to add these controls to the page are: For three panels you need to set the **Height** and **Width** properties to the following dimensions:

- Panel1: **Height** 36px **Width** 230px
- Panel2: **Height** 36px **Width** 230px
- Panel3: **Height** 200px **Width** 474px

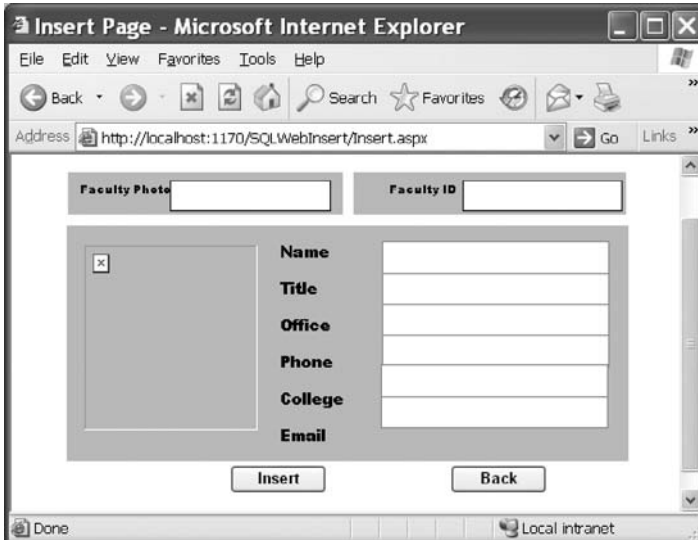


Figure 8.33 Insert Web page.

Also set the `Format/Position` properties for all Panel, Image, Label, and TextBox controls to `Absolute` position. You can move, copy, and paste those Label and TextBox controls one by one on the page form to save time. Your finished Insert page should match the one shown in Figure 8.33.

8.4.2 Develop Codes to Perform Data Insertion Function

The function of this Insert page are:

1. While the project is running, you need to open the Insert page by clicking on the Insert button from the Faculty page.
2. To insert a new faculty record into the database, you need to enter seven pieces of new information into seven textboxes on the insert page. The information includes the faculty_id, faculty name, title, office, phone, college, and email.
3. The Faculty Photo textbox is optional, which means that you can either enter a new faculty photo name with this new record or leave it blank. If you leave it blank, a default photo will be displayed when this new record is validated later.
4. After all information has been entered into all textboxes, you can click on the Insert button to insert this new record into the Faculty table in our sample database via the Web page.
5. The Back button is used to return to the Faculty page to perform the data validation to confirm this data insertion.

Now let's start creating the codes for this Insert page. The coding can be divided into three parts: the coding for the `Page_Load()` and Back button's click method, the coding for the Insert button's click method, and the coding for other user-defined methods.

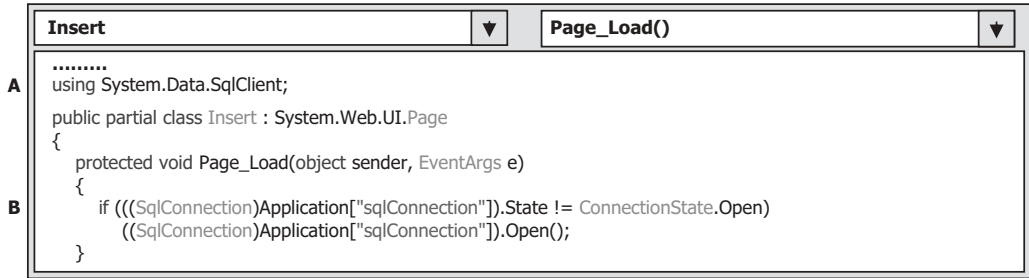


Figure 8.34 Coding for the Page_Load method.

8.4.3 Develop Codes for Page_Load and Back Button Click Methods

Open the code page window of the Insert Web form, and enter the following SQL Server Data Provider–related namespace at the top of this page:

```
using System.Data.SqlClient;
```

Open the Page_Load() method and enter the codes shown in Figure 8.34 into this method.

The function of this piece of coding is:

- A.** The SQL Server Data Provider–related namespace is added into the namespace area in this page since we need to use some data components stored in that namespace to perform the data insertion.
- B.** The global connection object sqlConnection stored in the Application state will be checked to make sure that a valid database connection exists before this data insertion. Redo this connection if no valid connection existed.

The coding for the Back button’s Click method is simple. As this button is clicked, the current page will be returned to the Faculty page to perform the data validation. Open this method by double-clicking on the Back button from the Insert Web form, and enter the following code into this method:

```
Response.Redirect("Faculty.aspx");
```

The Redirect() method of the server’s Response object is used to redirect the current page to the Faculty page.

8.4.4 Develop Codes for Insert Button Click Method

Open the Insert button’s Click method by double-clicking on the Insert button from the Insert Web form, and enter the codes shown in Figure 8.35 into this method.

Let’s take a closer look at this piece of code to see how it works.

- A.** The Insert query string is declared first, and it contains seven pieces of information related to seven columns in the Faculty table in the database.


```

protected void cmdInsert_Click(object sender, EventArgs e)
{
A   string cmdString = "INSERT INTO Faculty (faculty_id, faculty_name, office, phone, college, title, email) " +
   "VALUES (@faculty_id,@faculty_name,@office,@phone,@college,@title,@email)";
B   SqlCommand sqlCommand = new SqlCommand();
   string FacultyImage;
   int intInsert = 0;
C   FacultyImage = txtPhoto.Text;           //reserve the new inserted faculty photo location
   if (FacultyImage == string.Empty)
       FacultyImage = "Default.jpg";
D   Application["FacultyImage"] = FacultyImage;   //reserve faculty image for validation
   PhotoBox.ImageUrl = FacultyImage;
E   Application["FacultyName"] = txtName.Text;   //reserve faculty name for validation
F   sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
   sqlCommand.CommandType = CommandType.Text;
   sqlCommand.CommandText = cmdString;
G   InsertParameters(sqlCommand);
H   intInsert = sqlCommand.ExecuteNonQuery();
I   sqlCommand.Dispose();
J   if (intInsert == 0)
       Response.Write("<script>alert('The data insertion is failed')</script>");
K   cmdInsert.Enabled = false;           //disable the Insert button
}

```

Figure 8.35 Coding for the Insert button's method.

- B.** The data components and local variables used in the method are declared here. The FacultyImage string variable is used to hold the faculty photo name, and the local integer variable intInsert is used to hold the returned data from the execution of the data insertion command.
- C.** The faculty photo name is reserved to the FacultyImage string variable. If this variable contains an empty string, which means that the user did not enter any photo name to the txtPhoto box, a default faculty image will be assigned and displayed as the project runs. Otherwise, the selected faculty photo will be displayed based on the photo name entered by the user.
- D.** The name of the faculty photo file, which is entered by the user, is reserved to the FacultyImage string variable. Also this faculty photo file is stored into the Application state as a global variable since we need to use it later as we perform data validation in the Faculty page. The reason we use the Application state to store this global variable is that each time when the client sends a request to the server, the server will post back a refreshed page to the client after the server received that request. The values of all global variables created in the client side will be lost after this refreshed page is sent back. Therefore, in order to keep those values, we must use this Application state to reserve them.
- E.** The faculty name entered by the user is reserved to the Application state as a global variable since we need to use it later as we perform data validation in the Faculty page. The reason we use the Application state to store this global variable is same as that we discussed in step D for the FacultyImage.
- F.** The Command object is initialized by assigning it with connection object stored in the Application state, the command type, and the command text objects, respectively.
- G.** The user-defined subroutine InsertParameters() is executed to assign all seven input parameters to the Parameters collection of the command object.

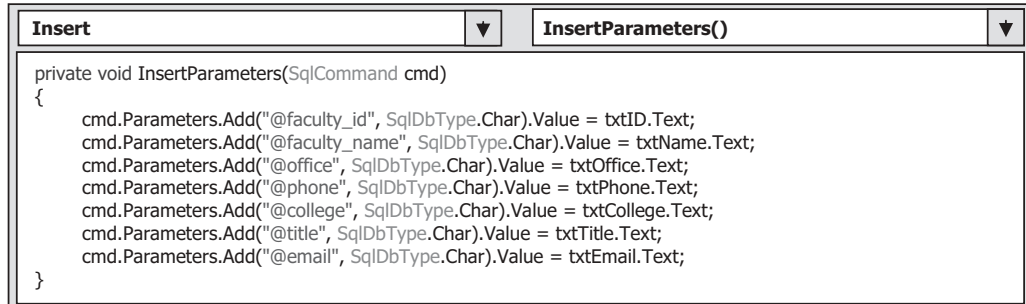


Figure 8.36 Coding for the subroutine InsertParameters.

- H.** The ExecuteNonQuery() method of the command object is called to run the insert query to perform this data insertion.
- I.** A cleaning job is performed to release all objects used in the method.
- J.** The ExecuteNonQuery() method will return a data value to indicate whether this data insertion is successful or not. The value of this returned data equals to the number of rows that have been successfully inserted into the Faculty table in the database. If a zero is returned, which means that no row has been inserted into the database, a warning message is displayed to indicate this situation. Otherwise, the data insertion is successful.
- K.** The Insert button is disabled after the current record is inserted into the database. This is to avoid the multiple insertions of the same record into the database. The Insert button will be enabled again when the content of the Faculty ID textbox is changed, which means that a new different faculty record will be inserted.

The detailed coding for the user-defined InsertParameters() method is shown in Figure 8.36.

This coding is straightforward and easy to understand. Each piece of new faculty information is assigned to the associated input parameter by using the Add() method of the Parameters collection of the Command object.

8.4.5 Develop Codes for Other Methods

The coding for other methods includes the coding for the Insert button's Click method in the Faculty page. The function of this piece of code is to direct the project from the Faculty page to the Insert page as the user clicks on this Insert button from the Faculty page.

Open the Insert button's Click method in the Faculty page by double-clicking on the Insert button from the Faculty Web form window, and enter the following code into this method to direct the project to the Insert page:

```
Response.Redirect("Insert.aspx");
```

Now we can run the project to test the data insertion function via the website. However, before we can start the project, make sure that a default faculty photo file named Default.jpg and a faculty photo file named Mhamed.jpg have been stored in the default folder in which our project is located since we need to use those photo files to

test our project. Also make sure that the start page is LogIn page by setting it up as the start page using the Start Options menu item.

Click on the Start Debugging button to run the project. Enter the suitable username and password such as jhenry and test to the LogIn page, and select the Faculty Information item from the Selection page to open the Faculty page. Click on the Insert button to open the Insert page and enter the following data as the information for inserting a new faculty member:

- Mhamed.jpg Faculty Photo textbox
- M56789 Faculty ID textbox
- Ali Mhamed Faculty Name textbox
- Professor Title textbox
- MTC-353 Office textbox
- 750-378-3355 Phone textbox
- University of Main College textbox
- amhamed@college.edu Email textbox

Your finished new faculty information page should match the one shown in Figure 8.37.

Click on the Insert button to insert this new record into the database. The Insert button is immediately disabled and the associated faculty image is displayed in the PhotoBox, which is shown in Figure 8.37.

Click on the Back button to return to the Faculty page, and next we need to perform data validation to confirm our data insertion is successful.



Figure 8.37 Running status of the Insert page.

8.4.6 Validate Data Insertion

To validate the new inserted data from the Faculty page, we need to do some modifications to the coding in the Faculty page and add some codes to this page to allow us to retrieve the new inserted record from the database and display it on this page. The following methods need to be modified:

1. Page_Load() method
2. ShowFaculty() method

Let's first take care of the first method, Page_Load() method. After a new faculty record is inserted into the database from the Insert page and returned to the Faculty page, the new inserted faculty name should be added into the combobox control ComboName to allow the user to select it to retrieve the new inserted information for the selected faculty. To do this, we need to add the codes shown in Figure 8.38 into the Page_Load() method of the Faculty page.

The codes we developed in the previous section have been highlighted with shading. The function of this piece of new added code is: Each time, when the server posts a refreshed Faculty page to the client, we need to inspect whether a new faculty record has been inserted into the database by checking the global variable FacultyName, which is stored in the Application state. If this global variable is empty, which means that no data insertion occurred, we need to do nothing. However, if this variable contains a valid faculty name, which means that a data insertion has occurred, we need to add this new inserted faculty name into the combobox control ComboName to allow users to select this new faculty from the control to perform the associated data actions against the database. After this new faculty name is added into the combobox control, we need to reset this global variable to avoid the multiple additions of the same faculty name into the ComboName control.

```

Faculty Page_Load()
protected void Page_Load(object sender, EventArgs e)
{
    if (((SqlConnection)Application["sqlConnection"]).State != ConnectionState.Open)
        ((SqlConnection)Application["sqlConnection"]).Open();

    if (!IsPostBack)
    {
        ComboName.Items.Add("Ying Bai");
        ComboName.Items.Add("Satish Bhalla");
        ComboName.Items.Add("Black Anderson");
        ComboName.Items.Add("Steve Johnson");
        ComboName.Items.Add("Jenney King");
        ComboName.Items.Add("Alice Brown");
        ComboName.Items.Add("Debby Angles");
        ComboName.Items.Add("Jeff Henry");
    }
    if ((string)Application["FacultyName"] != string.Empty)
    {
        ComboName.Items.Add((string)Application["FacultyName"]);
        Application["FacultyName"] = string.Empty;
    }
}

```

Figure 8.38 Modified coding for the Page_Load method.

During the data insertion process, the user may want to insert a faculty photo by entering the name of the faculty photo file into the Faculty Photo textbox. However, another possibility is that the user may not want to insert any faculty photo with that data insertion. In that case, the content of the Faculty Photo textbox should be empty. Recall that when we developed the coding for the Insert button's Click method in the Insert page (refer to Section 8.4.4), we used a global variable FacultyImage that is stored in the Application state to store the name of the new inserted faculty photo file. Now when we validate that data insertion, we need to confirm whether the user inserted a faculty photo or not by checking that global variable FacultyImage.

Open the user-defined ShowFaculty() method and add the codes shown in Figure 8.39 into this method. The modified codes have been highlighted in bold. The function

```

Faculty ShowFaculty()
private string ShowFaculty(string fName)
{
    string FacultyImage;
    switch (fName)
    {
        case "Black Anderson":
            FacultyImage = "Anderson.jpg";
            break;
        case "Ying Bai":
            FacultyImage = "Bai.jpg";
            break;
        case "Satish Bhalla":
            FacultyImage = "Satish.jpg";
            break;
        case "Steve Johnson":
            FacultyImage = "Johnson.jpg";
            break;
        case "Jenney King":
            FacultyImage = "King.jpg";
            break;
        case "Alice Brown":
            FacultyImage = "Brown.jpg";
            break;
        case "Debby Angles":
            FacultyImage = "Angles.jpg";
            break;
        case "Jeff Henry":
            FacultyImage = "Henry.jpg";
            break;
        default:
            FacultyImage = "No Match";
            break;
    }
    if (FacultyImage != "No Match")
        PhotoBox.ImageUrl = FacultyImage;
    else
        if ((string)Application["FacultyImage"] == string.Empty)
            FacultyImage = "Default.jpg";
        else
            FacultyImage = (string)Application["FacultyImage"];
        PhotoBox.ImageUrl = FacultyImage;
    return FacultyImage;
}

```

Figure 8.39 Modifications to the coding of ShowFaculty method.

of these new added codes is that a default faculty photo file `Default.jpg` will be assigned to the `FacultyImage` variable if the global variable `FacultyImage` is empty, which means that the user does not want to add a new faculty photo with that data insertion and the Faculty Photo textbox in the Insert page is blank. Otherwise the faculty photo file stored in the Application state will be assigned to the `FacultyImage` variable that will be displayed later in the PhotoBox image control in the Faculty page.

Compared with the original coding we did for this part, it can be found that the warning message box, which will be displayed if no matched faculty image can be found from this `ShowFaculty` method, has been removed. Right now the default faculty image will be displayed. Either the global variable `FacultyImage` is empty or no matched faculty image can be found. The reason we made this modification to this part is that we want our project more professional and neat. Of course, if you like to keep that warning message box to indicate the situation in which no matched faculty image can be found, you can add another global variable to monitor whether the Insert page has been executed or not. If the Insert page has not been executed and no matched faculty image can be found, the warning message should be displayed. Otherwise, the default faculty image should be displayed.

Now we have completed all modifications to the coding on our Faculty page, and we can run the project to test our data insertion function via the website. Recall that we have already inserted a new faculty record with the faculty name `Ali Mhamed` into the Faculty table in the last section. In order to validate this insertion, we need to run the project and insert this new record again. To avoid the duplicated insertion, we need first to open our sample database to delete that new inserted record from the Faculty table. Open the SQL Server 2005/SQL Server Management Studio Express, and then open our sample database `CSE_DEPT.mdf` that is located at the folder `C:\database\SQLServer` and the Faculty table, and delete that new inserted record.

Click on the Start Debugging button to run our project. Enter the suitable username and password to the LogIn page, and select the Faculty Information item from the Selection page to open the Faculty page. Click on the Insert button to open the Insert page and enter the following data as the information for a new faculty member:

- | | |
|-----------------------|-----------------------|
| • Mhamed.jpg | Faculty Photo textbox |
| • M56789 | Faculty ID textbox |
| • Ali Mhamed | Faculty Name textbox |
| • Professor | Title textbox |
| • MTC-353 | Office textbox |
| • 750-378-3355 | Phone textbox |
| • University of Main | College textbox |
| • amhamed@college.edu | Email textbox |

Click on the Insert button to insert this new record into the database. Then click on the Back button to return to the Faculty page to perform the data validation.

Go to the ComboName combobox control, and you can find that the new inserted faculty name `Ali Mhamed` is already in there. Click on it to select this faculty and then click on the Select button to retrieve this new inserted record from the database and display it in this page. The inserted record is displayed in this page, which is shown in Figure 8.40.

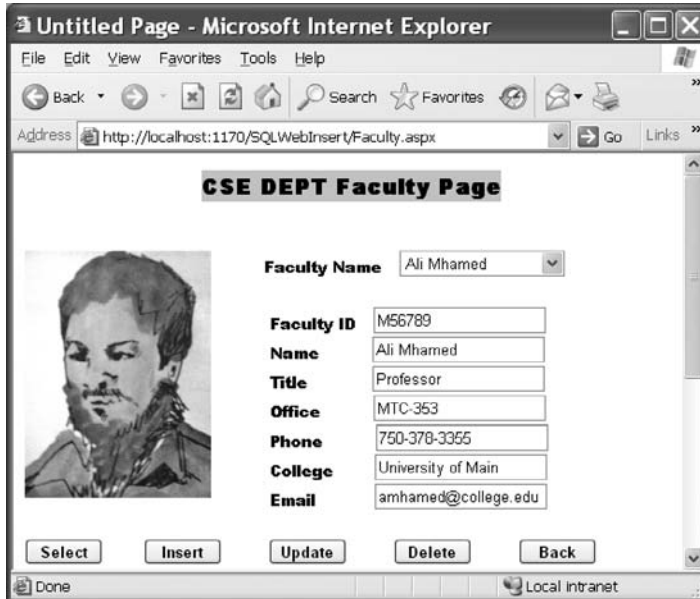


Figure 8.40 Data validation process.

Our data insertion is successful. Click on the Back button and then on the Exit button to close our project. A complete Web application project SQLWebInsert can be found at the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

In the next section, we will discuss how to perform data updating and deleting actions against our SQL Server database via the website.

8.5 DEVELOP WEB APPLICATIONS TO UPDATE AND DELETE DATA IN SQL SERVER DATABASES

To update or delete data against the relational databases is a challenging topic. We have provided a very detailed discussion and analysis of this topic in Section 7.1.1 in Chapter 7. Refer to that section to get more detailed information for these data actions. Here we want to emphasize some important points related to data updating and deleting.

1. When updating or deleting data against related tables in a database, it is important to update or delete data in the proper sequence in order to reduce the chance of violating referential integrity constraints. The order of command execution will also follow the indices of the DataRowCollection in the database. To prevent data integrity errors from being raised, the best practice is to update or delete data against the database in the following sequence:
 - a. Child table: delete records.
 - b. Parent table: insert, update, and delete records.
 - c. Child table: insert and update records.
2. To update existing data in the database, generally it is unnecessary to update the primary key for that record. It is much better to insert a new record with a new primary key into

the database than updating the primary key for an existing record because of the complicated table operations listed above. In practice, it is very rare to update a primary key for an existing record in the database in most real applications. Therefore, in this section, we concentrate our discussion on updating the existing records by modifying all data columns except the primary key column.

3. To delete a record from a relational database, the normal operation sequence listed above must be followed. For example, to delete a record from the Faculty table in our application, one must first delete those records that are related to the data to be deleted in the Faculty table from the child table such as the LogIn and Course tables, and then one can delete the record from the Faculty table. The reason for this deleting sequence is because the `faculty_id` is a foreign key in the LogIn and the Course tables, but it is a primary key in the Faculty table. One must first delete data with the foreign keys and then one can delete the data with the primary key from the database.

Keep these three points in mind. Now let's begin our project. We can modify one of our existing projects, SQLWebInsert, and make it our new project SQLWebUpdateDelete. To do that, open Windows Explorer and create a new folder Chapter 8 if you have not done that. Then copy the project SQLWebInsert from the folder DBProjects\Chapter 8 from the accompanying ftp site (see Chapter 1) and paste it to our new folder C:\Chapter 8. Rename this project SQLWebUpdateDelete.

8.5.1 Application User Interfaces

To update or delete an existing faculty record in our sample database, we don't need any new Web page as our user interface, and we can use the Faculty page as our user interface to perform these data actions. To meet our data actions' requirements, we need to perform some modifications to the Faculty page.

The first modification to the Faculty Web form is to clean up the Faculty ID textbox during the data updating process because we don't want users to modify this piece of information based on our discussion in step 2 in the last section.

8.5.2 Modify Coding for Faculty Page

Besides the coding development for the Update button's Click method we will discuss in the next section, the second modification to this page is to add one more statement into the Select button's Click method, which is shown in step A in Figure 8.41.

The new added statement has been highlighted in bold in Figure 8.41. The purpose of this statement is to store the current selected faculty name located at the combobox control ComboName into the Application state function as a global variable. During the data updating process, the faculty name may be updated by the user. If this happens, the updated faculty name stored in the txtName textbox will be added into the combobox control ComboName, and the original faculty name should be removed from that control. In order to remember the original faculty name, we must use this global variable to keep it since this is a Web application, and each time the server posts back a refreshed Faculty page based on the client's request, all contents in all controls on that page will be refreshed and all old data will be lost.

Faculty	cmdSelect_Click()
<pre> protected void cmdSelect_Click(object sender, EventArgs e) { string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty "; cmdString += "WHERE faculty_name LIKE @name"; SqlCommand sqlCommand = new SqlCommand(); SqlDataReader sqlDataReader; A Application["oldFacultyName"] = ComboName.Text; sqlCommand.Connection = (SqlConnection)Application["sqlConnection"]; sqlCommand.CommandType = CommandType.Text; sqlCommand.CommandText = cmdString; sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = ComboName.Text; string strName = ShowFaculty(ComboName.Text); sqlDataReader = sqlCommand.ExecuteReader(); if (sqlDataReader.HasRows == true) FillFacultyReader(sqlDataReader); else Response.Write("<script>alert('No matched faculty found!')</script>"); sqlDataReader.Close(); sqlCommand.Dispose(); } </pre>	

Figure 8.41 Modified Select button's Click method.

Faculty	ShowFaculty()
<pre> if (FacultyImage != "No Match") PhotoBox.ImageUrl = FacultyImage; else A if (((string)Application["FacultyImage"]) == string.Empty) (string)Application["FacultyImage"] == null) FacultyImage = "Default.jpg"; else FacultyImage = (string)Application["FacultyImage"]; PhotoBox.ImageUrl = FacultyImage; return FacultyImage; } </pre>	

Figure 8.42 Modified codes of the ShowFaculty method.

Another modification to this Faculty page is to add one more statement to the `if` condition in the `ShowFaculty()` method, which is shown in step **A** in Figure 8.42, to display a default faculty image if no data insertion action is performed. The new added instruction has been highlighted in bold.

The purpose of adding this `or` condition is to display a default image if no data insertion action is performed since this `ShowFaculty()` method will be used by different data actions, including data selection, insertion, and updating, as the project runs. Without this `or` condition, no faculty image will be displayed if no data insertion occurred, even when data updating is performed with a default faculty image selected by the user. Now let's develop the codes for the Update button's Click method.

Faculty	cmdUpdate_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p> <p>F</p> <p>G</p> <p>H</p> <p>I</p>	<pre>protected void cmdUpdate_Click(object sender, EventArgs e) { string cmdString = "UPDATE Faculty SET faculty_name = @name, office = @office, phone = @phone, " + "college = @college, title = @title, email = @email " + "WHERE (faculty_name LIKE @oldName)"; SqlCommand sqlCommand = new SqlCommand(); int intUpdate = 0; txtID.Text = string.Empty; //clean up the faculty_id textbox if (txtName.Text != (string)Application["oldFacultyName"]) //the faculty name is updated { ComboName.Items.Add(txtName.Text); ComboName.Items.Remove((string)Application["oldFacultyName"]); } sqlCommand.Connection = (SqlConnection)Application["sqlConnection"]; sqlCommand.CommandType = CommandType.Text; sqlCommand.CommandText = cmdString; UpdateParameters(ref sqlCommand); intUpdate = sqlCommand.ExecuteNonQuery(); sqlCommand.Dispose(); if (intUpdate == 0) Response.Write("<script>alert('The data updating is failed')</script>"); } </pre>

Figure 8.43 Coding for the Update button's Click method.

8.5.3 Develop Codes for Update Button Click Method

Open this method by double-clicking on the Update button from the Faculty Web form window and enter the codes shown in Figure 8.43 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** An updating query string is declared first with the `oldName` as the name of the dynamic parameter. This is because when you want to update the faculty name, the original name stored in the combobox control `ComboName` becomes the old name, and we need to distinguish this old name from the updated name.
- B.** The data component, Command object, used in this method is created here. A local integer variable `intUpdate` is also created, and it is used as a value holder to keep the returned data from executing the `ExecuteNonQuery()` method later.
- C.** Before we can perform data updating, first we need to clean up the Faculty ID textbox since we don't want to update this piece of information.
- D.** Now we need to check whether the user wants to update the faculty name or not by comparing the global variable `oldFacultyName` that is stored in the Application state function during the data selection process in the Select button's click method with the current faculty name stored in the textbox control `txtName`. If both are different, that means the user has updated the faculty name. In that case, we need to add the updated faculty name into the combobox control `ComboName` and remove the old faculty name from that control to allow users to select this updated faculty from the combobox list to perform the data actions in the database in the future.
- E.** The Command object is initialized with the Connection object, Command type and Command text.

Faculty	▼	UpdateParameters()	▼
<pre>private void UpdateParameters(ref SqlCommand cmd) { cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text; cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text; cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text; cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text; cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text; cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text; cmd.Parameters.Add("@oldName", SqlDbType.Char).Value = ComboName.Text; } }</pre>			

Figure 8.44 Coding for the UpdateParameters method.

- E.** The user-defined UpdateParameters() method, whose detailed coding is shown in Figure 8.44, is called to assign all input parameters to the command object.
- G.** The ExecuteNonQuery() method of the Command class is called to execute the data updating operation. This method returns a feedback value to indicate whether this data updating is successful or not, and this returned value is stored into the local integer variable intUpdate.
- H.** A cleaning job is performed to release all data objects used in this method.
- I.** The data value returned from calling the ExecuteNonQuery() is exactly equal to the number of rows that have been successfully updated in the database. If this value is zero, which means that no row has been updated and this data updating has failed, a warning message is displayed. Otherwise if this value is nonzero, this data updating is successful.

The detailed coding for the method UpdateParameters() is shown in Figure 8.44.

Seven input parameters are assigned to the Parameters collection property of the command object using the Add() method. One important point for this parameter assignment is the last input parameter or the dynamic parameter oldName. The original or the old faculty name oldFacultyName stored in the Application state function must be used as the value for this parameter. Some readers may disagree with me: The original or the old faculty name is located at the combobox control ComboName, and we can directly get it from that control without using this global variable. Well, this statement is correct for the Windows-based application without any problem. However, for the Web-based application, it is absolutely wrong. Recall that when the users clicked on the Update button to perform a data updating action, this updating request will be sent to the server, and the server will post back a refreshed Faculty page to the client. All old or the original data stored in all textboxes or comboboxes in the previous page will be gone. In other words, the contents stored in all textboxes and comboboxes in this refreshed page are different with the contents stored in the previous pages. A wrong updating may occur if you still use the faculty name stored in the combobox control ComboName in the current or refreshed page.

At this point we have finished all coding jobs for the data updating actions against the SQL Server database in the Faculty page. Before we can run the project to test this data updating function, we must make sure that the starting page is the LogIn page, and a default faculty image file Default.jpg has been stored in our default folder. To check the starting page, right-click on our project icon from the Solution Explorer window, select the Start Options item from the pop-up menu, and then check the Specific page radio button and select the LogIn.aspx as the starting page.

Now let's run the project to test the data updating actions. Click on the Start Debugging button to run the project, enter the suitable username and password to the LogIn page, and select the Faculty Information item from the Selection page to open the Faculty page. Select the faculty name Ying Bai from the combobox control ComboName and click on the Select button to retrieve the information for this selected faculty from the database and display it on this page.

Now let's test the data updating actions in two steps: First, we update the faculty information without touching the faculty name, and second we update the faculty information with changing the faculty name.

Let's start from the first step now. Enter the following information into the associated textboxes to update this faculty record:

- Professor Title textbox
- MTC-353 Office textbox
- 750-378-3300 Phone textbox

Click on the Update button to perform this data updating. To confirm this data updating, first select another faculty from the combobox control ComboName and click on the Select button to retrieve and display that faculty information. Then select the faculty Ying Bai whose information has just been updated from the combobox control and click on the Select button to retrieve and display it. You can see that the selected faculty information has been updated, which is shown in Figure 8.45.

Next let's perform data updating with the second method: Include updating of the faculty name. Still keep the current page unchanged, and then modify the faculty information from the associated textboxes by entering the following data:



Figure 8.45 Data updating process.

- Peter Bai Faculty Name textbox
- Associate Professor Title textbox
- MTC-555 Office textbox
- 750-378-3355 Phone textbox
- pbai@college.edu Email textbox

Click on the Update button to update this faculty information. Immediately you will find that the original faculty name Ying Bai has disappeared from the combobox control ComboName. To confirm this data updating, in a similar way, let's first select another faculty from the combobox control ComboName and click on the Select button to retrieve and display that faculty information. Then select the faculty Peter Bai from the combobox control and click on the Select button to retrieve and display it. You can see that the selected faculty information including the faculty name has been updated, which is shown in Figure 8.46.

One point to note is the faculty photo. When you updated the faculty name, you did not place an updated faculty photo file in our default folder. Therefore, a default faculty photo is displayed for this situation. You can change this situation by placing an updated faculty photo file in our default folder before the project runs if you like the correct faculty photo to be displayed with this data updating. Our data updating action is very successful.

Next let's take care of the data deleting action in the SQL Server database. Similarly to data updating, for data deleting we don't need any new Web page as our user interface, and we can still use the Faculty page to perform data deleting actions.



Figure 8.46 Data updating process—including the faculty name updating.

8.5.4 Develop Codes for Delete Button Click Method

Since deleting a record from a relational database is a complex issue, we divide this discussion into five sections:

1. Relationships between five tables in our sample database
2. Data deleting sequence
3. Using the Cascade deleting option to simplify the data deleting
4. Creating the stored procedure to perform the data deleting
5. Calling the stored procedure to perform data deleting

Let's start with the first section.

8.5.4.1 Relationships Between Five Tables in Our Sample Database

As we discussed at the beginning of this section, to delete a record from a relational database, one must follow the correct sequence. In other words, one must first delete the records related to the record to be deleted in the parent table from the child tables. In our sample database, five tables are related together by using the primary and foreign keys. In order to make these relationships clear, we redraw Figure 2.5 in Chapter 2, which is Figure 8.47 in this section, to illustrate this issue.

If you want to delete a record from the Faculty table, you must first delete the related records from the LogIn, Course, StudentCourse, and Student tables, and then you can delete the desired record from the Faculty table. The reason for that is because the relationships existed between five tables.

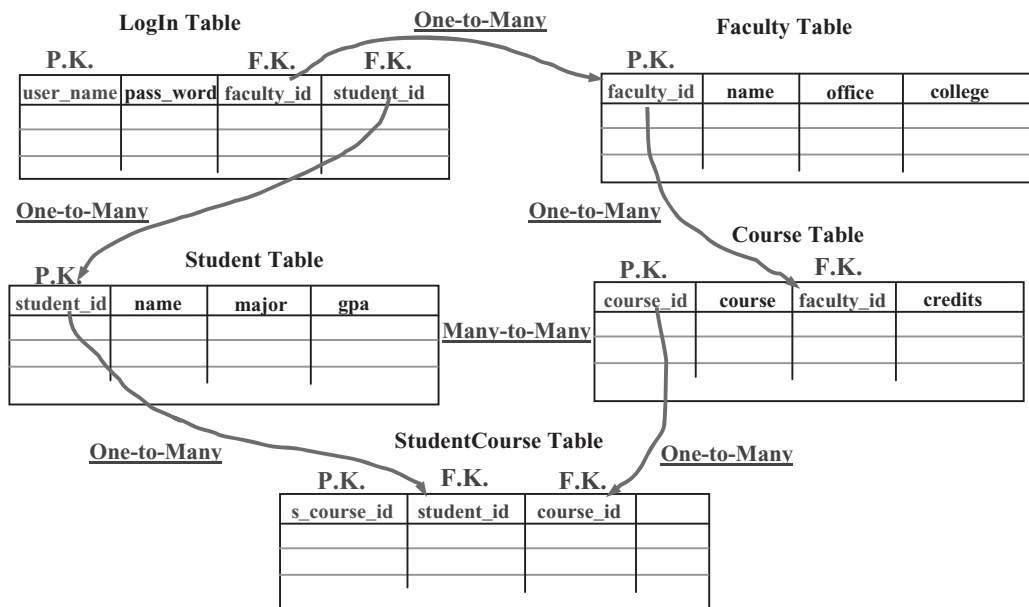


Figure 8.47 Relationships between five tables.

For example, if one wants to delete a faculty record from the Faculty table, one must perform the following deleting operations:

- The `faculty_id` is a primary key in the Faculty table, but it is a foreign key in the LogIn and the Course table. Therefore, the Faculty table is a parent table and the LogIn and the Course are child tables. Before one can delete any record from the Faculty table, one must first delete records that have the `faculty_id` as the foreign key from the child tables. In other words, one must first delete those records that use the `faculty_id` as a foreign key from the LogIn and the Course tables.
- When deleting records that use the `faculty_id` as a foreign key from the Course table, the related `course_id` that is a primary key in the Course table will also be deleted. The Course table right now is a parent table since the `course_id` is a primary key for this table. But as we mentioned, to delete any record from a parent table, one must first delete the related records from the child tables. Now the StudentCourse table is a child table for the Course table; therefore, the records that use the `course_id` as a foreign key in the StudentCourse table should be deleted first.
- After those related records in the child tables are deleted, finally the faculty member can be deleted from the parent table, Faculty table.

8.5.4.2 Data Deleting Order Sequence

Summarily, to delete a record from the Faculty table, one needs to perform the following deleting operations in the **order sequence** shown below:

1. Delete all records that use the `course_id` as the foreign key from the StudentCourse table.
2. Delete all records that use the `faculty_id` as the foreign key from the LogIn table.
3. Delete all records that use the `faculty_id` as the foreign key from the Course table.
4. Delete the desired faculty member from the Faculty table.

You can see how complicated the operations are to delete one record from the relational database from this example.

8.5.4.3 Use Cascade Deleting Option to Simplify Data Deleting

To simplify the data deleting operations, we can use the cascade deleting option provided by the SQL Server 2005 Database Management Studio.

Recall that when we created and built the relationship between our five tables, the following five **relationships** were built between tables:

1. A relationship between the LogIn and the Faculty tables was set up using the **faculty_id** as a foreign key `FK_LogIn_Faculty` in the LogIn table.
2. A relationship between the LogIn and the Student tables was set up using the **student_id** as a foreign key `FK_LogIn_Student` in the LogIn table.
3. A relationship between the Course and the Faculty tables was set up using the **faculty_id** as a foreign key `FK_Course_Faculty` in the Course table.
4. A relationship between the StudentCourse and the Course table was set up using the **course_id** as a foreign key `FK_StudentCourse_Course` in the StudentCourse table.

5. A relationship between the StudentCourse and the Student table was set up using the **student_id** as a foreign key FK_StudentCourse_Student in the StudentCourse table.

Refer to the data deleting sequence listed in the last section to delete a record from the Faculty table. One needs to perform four deleting operations in that sequence. Compared with those four deleting operations, the first one is the most difficult and the reason for that is: To perform the first data deleting operation, one must first find all course_ids that use the faculty_id as the foreign key from the Course table, and then based on those course_ids, one needs to delete all records that use those course_ids as the foreign keys from the StudentCourse table. For deleting operations in sequences 3 and 4, they are very easy, and each deleting operation only needs one deleting query. The conclusion for this discussion is: How do we find an easy way to complete the deleting operation in sequence 1?

A good solution to this question is to use the **Cascade** option, as we did in Chapter 2, to perform data deleting and updating in a cascaded mode. This Cascade option allows the SQL Server 2005 database engine to perform that deleting operation in sequence 1 as long as a **Cascade** option is selected for relationships 4 and 5 listed above.

Now let's use a real example to illustrate how to use this **Cascade** option to simplify the data deleting operations, especially for the first data deleting in that sequence. Open the SQL Server Management Studio Express by going to Start\All Programs\Microsoft SQL Server 2005\SQL Server Management Studio Express. On the opened Studio Express window, click on the Database and expand our sample database, C:\database\SQLServer\CSE_DEPT.mdf, to display all five tables. Since we are only interested in relationships 4 and 5, expand the dbo.StudentCourse table and expand the Keys folder to display all the keys we set up in Section 2.10.4. Double-click on the key FK_StudentCourse_Course to open it, which is shown in Figure 8.48.

On the opened dialog box, keep our desired foreign key FK_StudentCourse_Course selected from the left pane, and then click on the small plus icon before the item INSERT And UPDATE Specification, and you can find that a **Cascade** mode has been set for both Delete Rule and Update Rule items, which is shown in Figure 8.48.

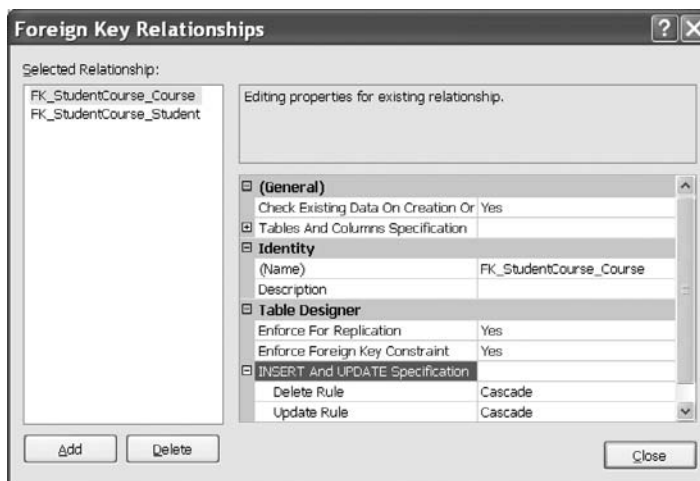


Figure 8.48 Foreign Key Relationship dialog box.

After this **Cascade** option is set up, each time you want to delete all records that use the `course_id` or the `student_id` as the foreign keys in the `StudentCourse` table, the SQL Server engine will perform those data deleting operations automatically for you in that cascaded sequence. Therefore, you can see how easy it is to perform data deleting in sequence 1.

Now let's create our codes for the Delete button's Click method to perform this data deleting operation.

8.5.4.4 Develop Codes to Perform Data Deleting

On the opened Visual Studio.NET, go to **File|Open Web Site** menu item to open our Web application project `SQLWebUpdateDelete`. Then open the Delete button's Click method from the Faculty Web form window by double-clicking on the Delete button. Enter the codes shown in Figure 8.49 into this method.

Let's take a closer look at this piece of codes to see how it works.

- A. The data deleting query string is declared first with the `faculty_name` as the query criterion.
- B. The data object and local variable used in this method are declared here. The integer variable `intDelete` is used to hold the returned value from calling the `ExecuteNonQuery()` method of the `Command` class later.
- C. The `Command` object is initialized by using the `Connection` object, `Command Type`, `Command Text`, and `Parameters` properties.
- D. After the `Command` object is initialized, the `ExecuteNonQuery()` method of the `Command` class is called to perform the data deleting action. This method will return a data value, and it is assigned to the local variable `intDelete`.
- E. A cleaning operation is performed to release all objects used in this method.

```

Faculty cmdDelete_Click()
protected void cmdDelete_Click(object sender, EventArgs e)
{
A   string cmdString = "DELETE FROM Faculty WHERE (faculty_name LIKE @FacultyName)";
B   SqlCommand sqlCommand = new SqlCommand();
   int intDelete = 0;
C   sqlCommand.Connection = (SqlConnection)Application["sqlConnection"];
   sqlCommand.CommandType = CommandType.Text;
   sqlCommand.CommandText = cmdString;
   sqlCommand.Parameters.Add("@FacultyName", SqlDbType.Char).Value = ComboName.Text;
D   intDelete = sqlCommand.ExecuteNonQuery();
E   sqlCommand.Dispose();
F   if (intDelete == 0)
   Response.Write("<script>alert('The data deleting is failed')</script>");
G   for (intDelete = 0; intDelete < 7; intDelete++) // clean up the Faculty textbox array
   {
   FacultyTextBox[intDelete] = new TextBox();
   FacultyTextBox[intDelete].Text = string.Empty;
   }
}

```

Figure 8.49 Coding for the Delete button's Click method.

Table 8.7 Data to Be Added into the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu

Table 8.8 Data to Be Added into the LogIn Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	NULL

Table 8.9 Data to Be Added into the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880

Table 8.10 Data to Be Added into the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1005	J77896	CSC-234A	3	CS/IS
1009	A78835	CSE-434	3	CE
1014	A78835	CSE-438	3	CE
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE

- F. The returned value from calling the `ExecuteNonQuery()` method is exactly equal to the number of rows that have been successfully deleted from our sample database. If this value is zero, which means that no row has been deleted or affected from our database and this data deleting has failed, a warning message is displayed. Otherwise if a nonzero value is returned, at least one row in our database has been deleted, and this data deleting is successful.
- G. A cleaning operation is performed to clean up the contents of all textboxes that stored the deleted faculty information.

At this point, we finished all coding operations to delete data in the SQL Server database via Web pages. Before we can run the project to test this deleting function, make sure that the starting page is the LogIn page. After the project runs, enter the suitable username and password to complete the LogIn process. Then open the Faculty page by clicking on the Faculty Information item from the Selection page, keep the default faculty Ying Bai selected from the combobox control, and then click on the Select button to retrieve and display this faculty's information.

Click on the Delete button to delete this faculty record from our database. To confirm this data deleting, keep the deleted faculty member Ying Bai selected in the combobox control `ComboName`, and click on the Select button to try to retrieve this deleted faculty information. A warning message: "No matched faculty found!" is displayed to indicate that this faculty member has been deleted from our sample database.

Another way to confirm this data deleting is to open our sample database and find that all records related to that deleted faculty, as shown in Tables 8.7 to 8.10, have been deleted from our database. Yes, our data deleting action is successful.

Before we can close the SQL Server Management Studio, it is highly recommended to recover all records that have been deleted from our sample database. To do this recovering operation, you need to take the following actions in the following order:

1. Recover the Faculty table by adding the deleted faculty record into the Faculty table, which is shown in Table 8.7.
2. Recover the LogIn table by adding the deleted login record into the LogIn table, as shown in Table 8.8.
3. Recover the Course table by adding the deleted courses taught by the deleted faculty member into the Course table, which is shown in Table 8.9.
4. Recover the StudentCourse table by adding the deleted courses taken by the associated students into the StudentCourse table, as shown in Table 8.10.

Some readers may have noticed the following interesting point: Although we have developed the codes to clean up all seven textboxes' contents after this deletion action, however, it looks like those pieces of code do not work and the deleted faculty information stored in those seven textboxes are still in there. What is the reason for that? Does our code have something wrong? Try to think about this and find the solution yourself. Yes, the answer is simple. This is a significant difference that exists between the Windows-based and Web-based applications since our project SQLWebUpdateDelete will run at the server side, and each time the server sends back a refreshed Faculty page as an action is performed from the client side. After a deletion action is performed, the server still sends back a refreshed page that contains the original faculty information in seven textboxes to the client.

A complete Web application project SQLWebUpdateDelete can be found at the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

8.6 DEVELOP ASP.NET WEB APPLICATIONS WITH LINQ TO SQL QUERY

In this section, we provide a fundamental end-to-end LINQ to SQL scenario for adding, modifying, and deleting data against our sample database via a Web page. As you know, LINQ to SQL queries can perform not only the data selections, but also the data insertion, updating, and deletion. The standard LINQ to SQL queries include:

- Select
- Insert
- Update
- Delete

To perform any of these operations or queries, we need to use entity classes and DataContext, which we discussed in Section 4.6.1, in Chapter 4 to do LINQ to SQL actions in our sample database. We already created a Console project QueryLINQSQL in that section to illustrate how to use LINQ to SQL to perform data queries, such as data selection, insertion, updating, and deleting, in our sample database CSE_DEPT.mdf. However, in this section, we want to create a Web-based project SQLWebLINQ by

adding a graphic user interface to perform the data selection, and data insertion, and data updating and deleting actions in our sample database `CSE_DEPT.mdf` using the LINQ to SQL query via Web pages. Now let's perform the following steps to create our new project `SQLWebLINQ`:

1. Create a new Visual C# Web-based project and name it `SQLWebLINQ`.
2. Create a new Web form page with (1) five button controls: Select, Insert, Update, Delete, and Exit; (2) eight TextBox controls, (3) one DropDownList control, and (4) one Image Box control.
3. Add a `System.Data.Linq` reference to this new project by right-clicking on our new project from the Solution Explorer window, selecting the `Add Reference` item and scroll down the list, and selecting the item `System.Data.Linq` from the list and clicking on the OK button.
4. Add the following directives at the top of the Faculty page file:
 - a. Using `System.Data.Linq`;
 - b. Using `System.Data.Linq.Mapping`;
5. Follow the steps listed in Section 4.6.1 in Chapter 4 to create entity classes using the Object Relational Designer. The database used in this project is `CSE_DEPT.mdf`, and it is located at the folder `C:\database\SQLServer`. Open the Server Explorer window and add this database by right-clicking on the Data Connections item and select `Add Connection` if it has not been added into our project.
6. We need to create five entity classes, and each of them is associated with a data table in our sample database. Drag each table from the Server Explorer window and place it on the Object Relational Designer canvas. The mapping file's name is `CSE_DEPT.dbml`. Make sure that you enter this name into the Name box in the Object Relational Designer.

Now let's begin the coding process for this project. Since we need to use the Select button's Click method to validate our data insertion and data updating and deleting actions, we need to divide our coding process into the following four parts:

1. Create a new object of the `DataContext` class and do some initialization coding.
2. Develop the codes for the Select button's Click method to retrieve the selected faculty information using the LINQ to SQL query.
3. Develop the codes for the Insert button's Click method to insert new faculty member using the LINQ to SQL query.
4. Develop the codes for the Update button's Click method to update the selected faculty member using the LINQ to SQL query.
5. Develop the codes for the Delete button's Click method to delete the selected faculty member using the LINQ to SQL query.

Before we can start the coding process, first let's create a new Web form page as our graphic user interface to perform those data actions.

8.6.1 Create New Web Form Page

In the newly created Web site project `SQLWebLINQ`, click on the View Designer button to open the default Web page, `Default.aspx`, and add the components listed in Table 8.11 into this Web page. Your finished Web page should match the one shown in Figure 8.50.

Table 8.11 Controls on the Faculty Web Page

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	CSE DEPT Faculty Page	0		Bold/Large
Image	PhotoBox		22		
Label	Label2	Faculty Photo	1		Bold/ Smaller
TextBox	txtPhoto		2		
Label	Label3	Faculty Name	3		Bold/Smaller
DropDownList	ComboName		4		
Label	Label3	Faculty ID	5		Bold/ Smaller
TextBox	txtID		6		
Label	Label4	Name	7		Bold/ Smaller
TextBox	txtName		8		
Label	Label5	Title	9		Bold/ Smaller
TextBox	txtTitle		10		
Label	Label6	Office	11		Bold/ Smaller
TextBox	txtOffice		12		
Label	Label7	Phone	13		Bold/ Smaller
TextBox	txtPhone		14		
Label	Label8	College	15		Bold/ Smaller
TextBox	txtCollege		16		
Label	Label9	Email	17		Bold/ Smaller
TextBox	txtEmail		18		
Button	cmdSelect	Select	19		Bold/ Small
Button	cmdInsert	Insert	20		Bold/ Small
Button	cmdUpdate	Update	21		Bold/ Small
Button	cmdDelete	Delete	22		Bold/ Small
Button	cmdExit	Exit	23		Bold/ Small

A key point in developing this Web page is that you have to set most controls' Position property to Absolute using the Format!Position menu item after you finish dragging each control from the Tool box window and placing it on the Web form page. These controls include all TextBoxes, DropDownList, Image control, and button controls. You also need to set the line-height size in the Style!Block property to 5 px for this Web page to align each label to make them vertically equal.

Another point is that you should not use the Copy!Paste menu items to create those buttons. Instead, you need to create those buttons one by one by dragging each of them from the ToolBox window and placing them on this page. Otherwise, the relationship between each button and its event method may be missed.

Finally, you need to set the Font size of the Text property of all label controls to smaller using the Format!Font menu item. Now let's start our coding process.

8.6.2 Create New Object of DataContext Class

We need to create this new object of the DataContext class since we need to use this object to connect to our sample database to perform data queries. We have connected

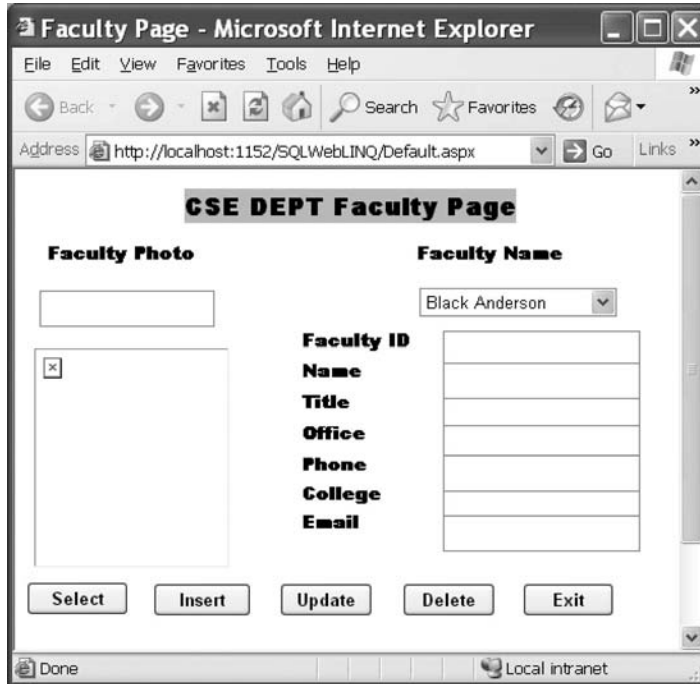


Figure 8.50 Default.aspx Web page.

this DataContext class to our sample database CSE_DEPT.mdf in step 5 in Section 8.6, and the connection string has been added into our web.config file when this connection is done. Therefore we do not need to indicate the special connection string for this object.

Some initialization coding includes retrieving all updated faculty members from the Faculty table in our sample database using the LINQ to SQL query and display them in the ComboName combobox control.

Open the code window and the Page_Load() method of the Faculty Web page, and enter the codes shown in Figure 8.51 into this method.

Let's take a close look at this piece of code to see how it works.

- A. A new field-level object of the DataContext class, cse_dept, is created first since we need to use this object to connect our sample database to this Web project to perform the data actions later.
- B. A user-defined UpdateFaculty() method is executed to retrieve all updated faculty members from our sample database and display them in the ComboName combobox control to allow users to select a desired faculty later. To avoid multiple displaying of retrieved faculty members, an if selection structure is adopted to make sure that we only display those updated faculty members in the combobox control ComboName at the first time as this Web page is loaded, and will not display them each time as the server sends back a refreshed Faculty page to the client when an action is performed in the client.
- C. Before we can update the combobox control ComboName by adding the updated faculty members into this control, a cleaning job is performed to avoid the multiple adding and displaying of those faculty members.

```

public partial class _Default : System.Web.UI.Page
{
    CSE_DEPTDataContext cse_dept = new CSE_DEPTDataContext();
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            UpdateFaculty();
            ComboName.SelectedIndex = 0;
        }
    }
    void UpdateFaculty()
    {
        ComboName.Items.Clear();
        var faculty = (from fi in cse_dept.Faculties
                      let fields = "faculty_name"
                      select fi);
        foreach (var f in faculty)
        {
            ComboName.Items.Add(f.faculty_name);
        }
    }
}

```

Figure 8.51 Initialization codes for the Faculty Web page.

- D.** The LINQ query is created and initialized with three clauses, **from**, **let**, and **select**. The range variable `fi` is selected from the Faculty entity in our sample database. All current faculty members (`faculty_name`) will be read back using the `let` clause and assigned to the query variable `faculty`.
- E.** The LINQ query is executed to pick up all queried faculty members and add them into the `ComboName` combobox control in our Faculty Form.

The coding for the Exit button's Click method is easy, just enter the following code line into this method: `Response.Write("<script>>window.close()</script>");`. The function of this line is to close this Web project if this Exit button is clicked.

8.6.3 Coding for Data Selection Query

Double-click on the Select button to open its Click method and enter the codes shown in Figure 8.52 into this method. The function of this piece of code is to retrieve all current faculty members from the Faculty table in our sample database and display them in the `ComboName` combobox control in the Faculty Form window as this Select button is clicked on by the user.

Let's take a close look at this piece of code to see how it works.

- A.** The user-defined `ShowFaculty()` method is executed to identify and display a matched faculty image for the selected faculty member. You can copy this piece of code from the Faculty Form class in the previous projects we developed in this chapter.
- B.** The LINQ query is created and initialized with three clauses: **from**, **where**, and **select**. The range variable `fi` is selected from the Faculty entity in our sample database based on a matched faculty members (`faculty_name`).

```

protected void cmdSelect_Click(object sender, EventArgs e)
{
    A   string strName = ShowFaculty(ComboName.Text);
    B   var faculty = (from fi in cse_dept.Faculties
                      where fi.faculty_name == ComboName.Text
                      select fi);
    C   foreach (var f in faculty)
        {
            txtID.Text = f.faculty_id;
            txtName.Text = f.faculty_name;
            txtTitle.Text = f.title;
            txtOffice.Text = f.office;
            txtPhone.Text = f.phone;
            txtCollege.Text = f.college;
            txtEmail.Text = f.email;
        }
}

```

Figure 8.52 Codes for the Select button Click method.

- C. The LINQ query is executed to pick up all columns for the selected faculty member and display them on the associated textbox in our Faculty Form.

It is recommended that you copy the body of the user-defined ShowFaculty() method from any Faculty Form page in the previous project SQLWebUpdateDelete we developed in this chapter, and paste it into this Faculty Form page. Now let's concentrate on the coding for our data insertion actions.

8.6.4 Coding for Data Insertion Query

Double-click on the Insert button from our Faculty Form page to open its Click method, and enter the codes shown in Figure 8.53 into this method.

Let's take a close look at this piece of code to see how it works.

- A. A new instance of the Faculty entity class is created since we need to add a new record into the Faculty table in our sample database.
- B. Seven pieces of new faculty information stored in seven textbox controls are assigned to the associated columns in the Faculty instance that can be mapped to the Faculty table in our sample database.
- C. A system method InsertOnSubmit() is executed to send our new created Faculty instance to our Faculty table via the DataContext class.
- D. Another system method SubmitChanges() is executed to perform this data insertion.
- E. After a new record has been inserted into our database, we need to update our combobox control ComboName to reflect that insertion. First, we need to clean up all original contents from this control to avoid multiple updating.
- F. The user-defined UpdateFaculty() method is called to complete this updating.
- G. In case the user wants to insert a new faculty image with that data insertion, the Text property of the textbox control txtPhoto, which stored a valid faculty image file, is assigned to the Application state function that works as a global variable. This global variable will


```

protected void cmdInsert_Click(object sender, EventArgs e)
{
A   Faculty newFaculty = new Faculty();
B   newFaculty.faculty_id = txtID.Text;
    newFaculty.faculty_name = txtName.Text;
    newFaculty.title = txtTitle.Text;
    newFaculty.office = txtOffice.Text;
    newFaculty.phone = txtPhone.Text;
    newFaculty.college = txtCollege.Text;
    newFaculty.email = txtEmail.Text;

    // Add the faculty members to the Faculty table.
D   cse_dept.Faculties.InsertOnSubmit(newFaculty);
    cse_dept.SubmitChanges();
E   ComboName.Items.Clear();
    UpdateFaculty();
G   Application["FacultyImage"] = txtPhoto.Text;
}

```

Figure 8.53 Codes for the Insert button Click method.

```

protected void cmdUpdate_Click(object sender, EventArgs e)
{
A   Faculty fi = cse_dept.Faculties.Where(f => f.faculty_name == ComboName.Text).First();
    // updating the existing faculty information
B   fi.faculty_name = txtName.Text;
    fi.title = txtTitle.Text;
    fi.office = txtOffice.Text;
    fi.phone = txtPhone.Text;
    fi.college = txtCollege.Text;
    fi.email = txtEmail.Text;
C   cse_dept.SubmitChanges();
D   ComboName.Items.Clear();
E   UpdateFaculty();
}

```

Figure 8.54 Codes for the Update button Click method.

be used later when we perform the confirmation of the data insertion in the Select button's Click method.

Now let's concentrate on the coding for our data updating and deleting actions.

8.6.5 Coding for Data Updating and Deleting Queries

Double-click on the Update button from our Faculty Form page window to open its Click method, and enter the codes shown in Figure 8.54 into this method.

Let's take a close look at this piece of code to see how it works.

- A. A selection query is executed using the Standard Query Operator method with the `faculty_name` as the query criterion. The `First()` method is used to return only the first

```

_Default | cmdDelete_Click()
protected void cmdDelete_Click(object sender, EventArgs e)
{
A   var faculty = (from fi in cse_dept.Faculties
                    where fi.faculty_name == ComboName.Text
                    select fi).Single<Faculty>();
B   cse_dept.Faculties.DeleteOnSubmit(faculty);
C   cse_dept.SubmitChanges();
    // clean up all textboxes
D   txtID.Text = string.Empty;
    txtName.Text = string.Empty;
    txtOffice.Text = string.Empty;
    txtTitle.Text = string.Empty;
    txtPhone.Text = string.Empty;
    txtCollege.Text = string.Empty;
    txtEmail.Text = string.Empty;
E   ComboName.Items.Clear();
F   UpdateFaculty();
}

```

Figure 8.55 Codes for the Delete button Click method.

matched record. It does not matter to our application since we have only one record that is associated with this specified `faculty_name`.

- B.** All six columns for the selected faculty member are updated by assigning the current value stored in the associated textbox to each column in the Faculty instance in our DataContext class object `cse_dept`.
- C.** This data updating can be really performed only after the system method `SubmitChanges()` is executed.
- D.** The combobox control `ComboName` is cleaned up to be ready to be updated.
- E.** The user-defined `UpdateFaculty()` method is executed to refresh the updated faculty members stored in that control.

Before we can run our Web project to test these data actions, let's complete the last coding for our data deleting action.

Double-click on the Delete button from our Faculty Form page window to open its Click method, and enter the codes shown in Figure 8.55 into this method.

Let's take a close look at this piece of code to see how it works.

- A.** A LINQ selection query is first executed to pick up the faculty member to be deleted. This query is initialized with three clauses: `from`, `where`, and `select`. The range variable `fi` is selected from the Faculty, which is exactly an instance of our entity class Faculty, and the `faculty_name` works as the query criterion for this query. All information related to the selected faculty members (`faculty_name`) will be retrieved and stored in the query variable `faculty`. The `Single()` means that only a single or the first record is queried.
- B.** The system method `DeleteOnSubmit()` is executed to issue a deleting action to the faculty instance, `Faculties` in our DataContext class object `cse_dept`.
- C.** Another system method `SubmitChanges()` is executed to exactly perform this deleting action against data tables in our sample database. Only after this method is executed is the selected faculty record deleted from our database.

- D. All textboxes that stored information related to the deleted faculty are cleaned up by assigning an empty string to each of them.
- E. The combobox control ComboName is cleaned up to be ready to be updated.
- F. The user-defined UpdateFaculty() method is executed to reflect deleting this faculty record for all faculty members stored in that control.

Now we can build and run our Web project to test the data actions against our sample database. One point we need to note before we can run the project is that we must make sure that all faculty image files should have been stored in the default folder in which our Web project SQLWebLINQ is located. In this application, it should be C:\Chapter 8\SQLWebLINQ.

Unlike other projects we developed in the previous chapters, in which a separate Insert Faculty Form must be used to perform the data insertion action, in this project, we can use the Faculty Form page to perform all data actions, including the data selection, data insertion, and data updating and deleting. As an example, let's run the project to test the data insertion action by inserting a new faculty member with the following information:

- P77777 Faculty ID textbox
- Peter Tom Faculty Name textbox
- Assistant Professor Title textbox
- MTC-200 Office textbox
- 750-378-2000 Phone textbox
- University of Miami College textbox
- ptom@college.edu Email textbox

Directly enter these new data into each associated textbox after the project runs, and you can select any faculty member from the combobox control ComboName to perform this insertion action. Click on the Insert button when you finish this data entering to all textboxes to perform this data insertion.

To confirm this data action, first select another faculty member from the combobox control ComboName and click on the Select button to retrieve and display that faculty's information. Then select the new inserted faculty Peter Tom, who should already be in the combobox control ComboName, and click on the Select button to try to retrieve that new inserted faculty's information and display it in this form. Your confirmation page should match the one shown in Figure 8.56.

A default faculty image is displayed for this data insertion since we did not include any faculty image file for this insertion. You can test to insert a new faculty with a selected faculty image by entering the name of that faculty image file into the Faculty Photo textbox control txtPhoto located at the upper-left corner of this page if you like.

Note that you had better recover any deleted faculty record if a data deleting action is tested for this project since we want to keep our database neat and complete. Refer to Tables 8.7 to 8.10 in section 8.5.4.4 to recover the deleted records to our sample database.

A complete Web page application project SQLWebLINQ can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1). Next let's take care of the Web applications with the Oracle database.



Figure 8.56 Testing status of the data insertion action.

8.7 DEVELOP ASP.NET WEB APPLICATION TO SELECT DATA FROM ORACLE DATABASES

Because of the coding similarity between the SQL Server and Oracle databases, we will emphasize the main differences between the codes in SQL Server and Oracle data actions. Also in order to save time and space, we will modify the existing Web application project SQLWebSelect we developed in the last section to make it our new project OracleWebSelect in this section.

The main coding differences between these two database operations are:

1. Connection string and Connection object in the LogIn page
2. LogIn query string in the LogIn page
3. Query string in the Faculty page
4. Query strings in the Course page, which include the query string in the Select button's Click method and the query string in the SelectedIndexChanged event method of the CourseList box control
5. Namespace and data objects used in the Selection page
6. Prefix for each data object and class used for the Oracle database operations
7. Data type of the passed arguments of methods for Oracle database operations

Now let's begin to modify the project SQLWebSelect based on the seven differences listed above to make it our new project OracleWebSelect. Open Windows Explorer and

create a new folder such as Chapter 8 if you have not created it. Copy the project SQLWebSelect from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1) and paste it in the folder C:\Chapter 8. Rename the project to OracleWebSelect.

Open the Visual Studio.NET, go to the File|Open Web Site menu item, and browse to our folder Chapter 8. Select our new project OracleWebSelect and then click on the Open button to open it.

Let's first modify the codes of the connection string in the LogIn page.

8.7.1 Modify Connection String and Connection Object on LogIn Page

Open the code page of the LogIn Web form by clicking on it from the Solution Explorer window, and then clicking on the View Code button. The first thing we need to do is to add the Oracle data client reference to our project. To do that, right-click on our project icon on the Solution Explorer window and select Add Reference item from the pop-up menu to open the Add Reference dialog box. Browse down on the list until you find the item System.Data.OracleClient. Click on this item to select it, and click on the OK button to add this reference to our project. Now we need to add one more Oracle Data Provider-related namespace to the top of this page to set the namespace that contains the data components for the Oracle Data Provider:

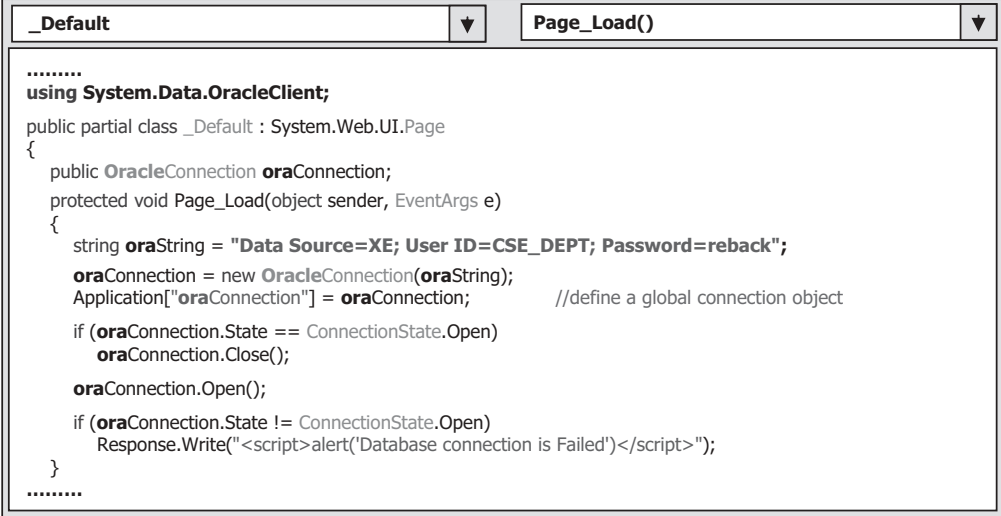
```
using System.Data.OracleClient;
```

Now open the Page_Load() method and perform the following modifications to the codes in this method:

- A. Add an Oracle Data Provider-related namespace to this page.
- B. Change the prefix for the global connection object from sqlConnection to oraConnection since we need to use the Oracle data components in this section.
- C. Change the connection string to contain the User ID and Password related to our sample Oracle database.
- D. Create a new instance of Oracle connection class with the Oracle connection string oraString as the argument. Also change the prefix for all Oracle data objects and classes from **sql** to **Oracle**, and from **sql** to **ora**, respectively.
- E. Change the prefix for the global connection object stored in the Application state function from **sql** to **ora**.
- F. Change the prefix for all following data components from **sql** to **ora**.

Your finished modifications to the Page_Load() method and the connection string should match the one shown in Figure 8.57. All modified parts have been highlighted in bold.

The next modification is to change the prefix of each Connection object in the Cancel button's Click method from sqlConnection to oraConnection. Your finished modification to this method should match the one shown in Figure 8.58. The modified parts have been highlighted in bold.



```

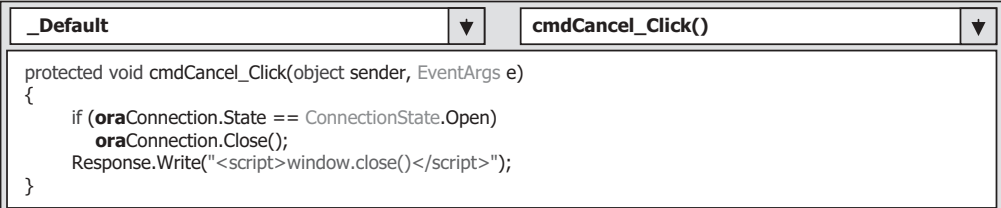
_Default Page_Load()
.....
A  using System.Data.OracleClient;
public partial class _Default : System.Web.UI.Page
{
B  public OracleConnection oraConnection;
protected void Page_Load(object sender, EventArgs e)
{
C  string oraString = "Data Source=XE; User ID=CSE_DEPT; Password=reback";
D  oraConnection = new OracleConnection(oraString);
E  Application["oraConnection"] = oraConnection; //define a global connection object
F  if (oraConnection.State == ConnectionState.Open)
    oraConnection.Close();

    oraConnection.Open();

    if (oraConnection.State != ConnectionState.Open)
        Response.Write("<script>alert('Database connection is Failed')</script>");
}
.....

```

Figure 8.57 Modified connection object and connection string.



```

_Default cmdCancel_Click()
protected void cmdCancel_Click(object sender, EventArgs e)
{
A  if (oraConnection.State == ConnectionState.Open)
B  oraConnection.Close();
    Response.Write("<script>window.close()</script>");
}

```

Figure 8.58 Modified connection object in Cancel button method.

8.7.2 Modify Query String in LogIn Page

Now open the LogIn button's Click method and perform the modifications shown in Figure 8.59 to the codes in this method. All modified parts have been highlighted in bold.

Let's take a look at this piece of code to see these modifications.

- A. Change the query string from the SQL Server database to the Oracle database. The Oracle database assignment operator `=:` is used to replace the SQL Server database assignment operator `LIKE @`.
- B. Change the prefix for all data components and classes from `sql` to `ora` and from `Sql` to `Oracle`, respectively.
- C. Change the nominal names for the dynamic parameters from `@name` to `name` and from `@word` to `word`, respectively. Also change the data type of these two dynamic parameters from `SqlDbType` to `OracleType`.
- D. Change the prefix for all data components and classes from `sql` to `ora` and from `Sql` to `Oracle`, respectively.
- E. Change the prefix for all data components from `sql` to `ora`.

```

_Default | cmdLogIn_Click()
protected void cmdLogIn_Click(object sender, EventArgs e)
{
A   string cmdString = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn ";
   cmdString += "WHERE (user_name=:name) AND (pass_word=:word)";
B   OracleCommand oraCommand = new OracleCommand();
   OracleDataReader oraReader;

   oraCommand.Connection = oraConnection;
   oraCommand.CommandType = CommandType.Text;
   oraCommand.CommandText = cmdString;
C   oraCommand.Parameters.Add("name", OracleType.Char).Value = txtUserName.Text;
   oraCommand.Parameters.Add("word", OracleType.Char, 8).Value = txtPassWord.Text;
D   oraReader = oraCommand.ExecuteReader();
   if (oraReader.HasRows == true)
   {
       Response.Redirect("Selection.aspx");
   }
   else
       Response.Write("<script>alert('No matched username/password found!')</script>");
E   oraCommand.Dispose();
   oraReader.Close();
}

```

Figure 8.59 Modifications to codes in LogIn button method.

Go to the FileSave All menu item to save these modifications. Now let's continue our modifications to the next page, the Faculty page.

8.7.3 Modify Query String in Faculty Page

The modifications to this page include the following contents:

1. Adding an Oracle Data Provider–related namespace to the top of this page
2. Modifications to the global connection object stored in the Application state function in the Page_Load() method
3. Modifications to the codes in the Select button's Click method
4. Modifications to the data type of the passed argument in the user-defined method FillFacultyReader()

Let's first add an Oracle Data Provider–related namespace to the namespace area located at the top of this page:

```
using System.Data.OracleClient;
```

Open the Page_Load() method and change the connection object stored in the Application state from sqlConnection to oraConnection. Your finished modifications to this method should match the one shown in Figure 8.60. The modified parts have been highlighted in bold.

Now open the Select button's Click method and perform the following modifications:

```

Faculty
Page_Load()
.....
using System.Data.OracleClient;
public partial class Faculty : System.Web.UI.Page
{
    private TextBox[] FacultyTextBox = new TextBox[7];
    protected void Page_Load(object sender, EventArgs e)
    {
        if (((OracleConnection)Application["oraConnection"]).State != ConnectionState.Open)
            ((OracleConnection)Application["oraConnection"]).Open();

        if (!IsPostBack)
        {
            ComboName.Items.Add("Ying Bai");
            ComboName.Items.Add("Satish Bhalla");
            ComboName.Items.Add("Black Anderson");
            ComboName.Items.Add("Steve Johnson");
            ComboName.Items.Add("Jenney King");
            ComboName.Items.Add("Alice Brown");
            ComboName.Items.Add("Debby Angles");
            ComboName.Items.Add("Jeff Henry");
        }
    }
}

```

Figure 8.60 Modified Page_Load method.

- A.** Change the query string by replacing the SQL Server database assignment operator LIKE @ with the Oracle database operator =: in the WHERE clause.
- B.** Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**, respectively.
- C.** Modify the global Connection object stored in the Application state from the **sqlConnection** to the **oraConnection**.
- D.** Modify the nominal name of the dynamic parameter @name by removing the @ symbol before the parameter name.
- E.** Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**.

Your finished modifications to this method should match the one shown in Figure 8.61. All modified parts have been highlighted in bold.

The modification to the data type of the passed argument in the user-defined method FillFacultyReader() is simple, and just change the data type of that passed argument from the SqlDataReader to the OracleDataReader.

8.7.4 Modify Query Strings in Course Page

The modifications to this page include the following contents:

1. Adding an Oracle Data Provider–related namespace to the top of this page
2. Modifications to the global connection object stored in the Application state in the Page_Load() method
3. Modifications to the codes in the Select button's Click method

Faculty	cmdSelect_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p>	<pre> protected void cmdSelect_Click(object sender, EventArgs e) { string cmdString = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty "; cmdString += "WHERE faculty_name =: name"; OracleCommand oraCommand = new OracleCommand(); OracleDataReader oraDataReader; oraCommand.Connection = (OracleConnection)Application["oraConnection"]; oraCommand.CommandType = CommandType.Text; oraCommand.CommandText = cmdString; oraCommand.Parameters.Add("name", OracleType.Char).Value = ComboName.Text; string strName = ShowFaculty(ComboName.Text); oraDataReader = oraCommand.ExecuteReader(); if (oraDataReader.HasRows == true) FillFacultyReader(oraDataReader); else Response.Write("<script>alert('No matched faculty found!')</script>"); oraDataReader.Close(); oraCommand.Dispose(); } </pre>

Figure 8.61 Modifications to the Select button's Click method.

4. Modifications to the codes in the SelectedIndexChanged event method of the listbox control CourseList
5. Modifications to the data type of the passed argument in the user-defined methods FillCourseReader() and FillCourseReaderTextBox()

Let's first add an Oracle Data Provider–related namespace to the namespace area that is located at the top of this page:

```
using System.Data.OracleClient;
```

Open the Page_Load() method and change the Connection object stored in the Application state function from sqlConnection to oraConnection. Your finished modifications to this method should match the one shown in Figure 8.62. The modified parts have been highlighted in bold.

Now open the Select button's Click method and perform the following modifications:

- A. The query string applied for the joined table must be modified since the syntax of the query string used for the SQL Server database is ANSI 92 standard, and this standard is up to date. However, this standard cannot be recognized by the Oracle database since the Oracle database still uses an old standard called ANSI 89 standard. In order to match the requirement of the Oracle database, the query string must be modified. Refer to Section 5.19.2.5 in Chapter 5 to get more detailed information about these two standards.
- B. Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**, respectively.
- C. Modify the global connection object stored in the Application state function from the **sqlConnection** to the **oraConnection**.
- D. Change the prefix for all data objects and classes from **sql** to **ora** and from **Sql** to **Oracle**.

Course	Page_Load()
<pre> using System.Data.OracleClient; public partial class Course : System.Web.UI.Page { private TextBox[] CourseTextBox = new TextBox[6]; protected void Page_Load(object sender, EventArgs e) { if (((OracleConnection)Application["oraConnection"]).State != ConnectionState.Open) ((OracleConnection)Application["oraConnection"]).Open(); if (!IsPostBack) //these items can only be added into the combo box in one time { ComboName.Items.Add("Ying Bai"); ComboName.Items.Add("Satish Bhalla"); ComboName.Items.Add("Black Anderson"); ComboName.Items.Add("Steve Johnson"); ComboName.Items.Add("Jenney King"); ComboName.Items.Add("Alice Brown"); ComboName.Items.Add("Debby Angles"); ComboName.Items.Add("Jeff Henry"); } } } </pre>	

Figure 8.62 Modified Page_Load method in the Course page.

Course	cmdSelect_Click()
<pre> protected void cmdSelect_Click(object sender, EventArgs e) { A string strCourse = "SELECT Course.course_id, Course.course FROM Course, Faculty "; B strCourse += "WHERE (Course.faculty_id=Faculty.faculty_id) AND (Faculty.faculty_name=:name)"; B OracleCommand oraCommand = new OracleCommand(); OracleDataReader oraDataReader; C oraCommand.Connection = (OracleConnection)Application["oraConnection"]; D oraCommand.CommandType = CommandType.Text; oraCommand.CommandText = strCourse; E oraCommand.Parameters.Add("name", OracleType.Char).Value = ComboName.Text; oraDataReader = oraCommand.ExecuteReader(); if (oraDataReader.HasRows == true) FillCourseReader(oraDataReader); else Response.Write("<script>alert('No matched course found!')</script>"); oraDataReader.Close(); oraCommand.Dispose(); } </pre>	

Figure 8.63 Modified Select button's Click method.

- E. Modify the nominal name of the dynamic parameter @name by removing the @ symbol before the parameter name. Also change the data type for this dynamic parameter from SqlDbType to OracleType.

Your modified Select button's Click method should match one that is shown in Figure 8.63. All modified parts have been highlighted in bold.

Next open the SelectedIndexChanged event method of the listbox control CourseList, and perform the following modifications:

Course	CourseList_SelectedIndexChanged()
<pre> protected void CourseList_SelectedIndexChanged(object sender, EventArgs e) { A string cmdString = "SELECT course, credit, classroom, schedule, enrollment, course_id FROM Course "; B cmdString += "WHERE course_id =: courseid"; B OracleCommand oraCommand = new OracleCommand(); B OracleDataReader oraDataReader; C oraCommand.Connection = (OracleConnection)Application["oraConnection"]; C oraCommand.CommandType = CommandType.Text; C oraCommand.CommandText = cmdString; D oraCommand.Parameters.Add("courseid", OracleType.Char).Value = CourseList.SelectedItem.ToString(); E oraDataReader = oraCommand.ExecuteReader(); if (oraDataReader.HasRows == true) FillCourseReaderTextBox(oraDataReader); else Response.Write("<script>alert('No matched course information found!')</script>"); oraDataReader.Close(); oraCommand.Dispose(); } </pre>	

Figure 8.64 Modified SelectedIndexChanged event method.

- A.** Modify the assignment operator for the dynamic parameter `courseid` in the query string by replacing the LIKE `@` with the Oracle assignment operator `=:`.
- B.** Change the prefix for all data objects and classes from `sql` to `ora` and from `Sql` to `Oracle`, respectively.
- C.** Modify the global connection object stored in the Application state function from the `sqlConnection` to the `oraConnection`.
- D.** Modify the nominal name of the dynamic parameter `@courseid` by removing the `@` symbol before the parameter `courseid`. Also change the data type for this dynamic parameter from `SqlDbType` to `OracleType`.
- E.** Change the prefix for all data objects and classes from `sql` to `ora` and from `Sql` to `Oracle`.

Your modified `SelectedIndexChanged` method should match one that is shown in Figure 8.64. All modified parts have been highlighted in bold.

Modifications to the data type of the passed argument in the user-defined methods `FillCourseReader()` and `FillCourseReaderTextBox()` are simple, and just change the data type of that passed argument from the `SqlDataReader` to the `OracleDataReader`.

The last modification is to add an Oracle Data Provider–related namespace into the Selection page and data objects used in the Selection page. Add the following namespace to the top of this page:

```
using System.Data.OracleClient;
```

Modify the global connection object stored in the Application state function from the `sqlConnection` to the `oraConnection` in the Exit button's Click method.

At this point, we have finished all modifications to this new project. Before we can run the project to test the function of our coding, the following two points must be noted:

1. Make sure that all faculty photo files have been stored in our default folder, in which our project file is located.
2. Make sure that the Start page in our Web application is LogIn page.

To confirm the second point, right-click on our project icon from the Solution Explorer window and select the **Start Options** item from the pop-up menu to open the Property Page. On the opened page window, select the **Specific page** radio button and click on the ellipsis button that is next to the Specific page box to open the **Select Page to Start** dialog box. In the opened dialog, click on the `LogIn.aspx` from the list and click on **OK** to select it as our start page. Finally click on the **OK** button to the Property Page to finish this setup.

Now you can click on the **Start Debugging** button to run the project to confirm the functionalities of our coding.

A complete Web application project `OracleWebSelect` can be found from the folder `DBProjects\Chapter 8` located at the accompanying ftp site (see Chapter 1).

8.8 DEVELOP ASP.NET WEB APPLICATION TO INSERT DATA INTO ORACLE DATABASES

Because of the coding similarity between the SQL Server and the Oracle databases, we only emphasize the important differences in the coding for these two databases. To save time and space, we need to modify an existing project `OracleWebSelect` by adding some new components and codes to this project.

Unlike other projects we developed in the previous sections, in which a new **Insert Faculty Form** page needs to be created to allow us to insert a new faculty member into our sample database, in this project, we will use the **Faculty Form** page as our graphical user interface to perform this data insertion. To do that, we need to perform the following tasks to finish developing this Web application.

First, we need to add the following two controls to the **Faculty Form** page to complete our graphic user interface design:

1. A **Faculty Photo** textbox control that allows users to enter a new faculty image file.
2. A label control to indicate the function of the **Faculty Photo** textbox control.

Next we need to perform some code modifications to the following methods:

1. The `Page_Load()` method in the **Faculty** page
2. The `ShowFaculty()` method in the **Faculty** page

Finally we need to develop new codes for the following methods:

1. The **Insert** button's `Click` method in the **Faculty** page
2. The user-defined `InsertParameters()` method
3. The `FacultyID` `TextChanged()` method

Now let's begin to perform these tasks to build our new project. First, let's start to modify an existing project `OracleWebSelect` to make it as our new project `OracleWebInsert`. Open Windows Explorer and create a new folder such as `Chapter 8` if you have not created it. Copy the project `OracleWebSelect` from the folder `DBProjects\Chapter 8` located at the accompanying ftp site (see Chapter 1) and paste it to our folder `C:\Chapter 8`. Rename the project to `OracleWebInsert`.

Open the Visual Studio.NET, go to the File|Open Web Site menu item, and browse to our folder Chapter 8. Then select our new project OracleWebInsert and click on the Open button to open it. First, let's complete our graphic user interface design by adding two controls to the Faculty page.

8.8.1 Add Two Controls to Faculty Page

Open the Faculty Form page window and add the two controls shown in Table 8.12 into this Faculty page. Your modified Faculty page should match the one shown in Figure 8.65.

Table 8.12 Added Controls to Faculty Web Page

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	Faculty Photo	0		Bold/Smaller
TextBox	txtPhoto		1		

Next let's perform some code modifications to some methods in the Faculty page.

8.8.2 Modify Codes to Some Methods on Faculty Page

1. First, we need to remove some codes from the Page_Load() method and add some new codes to retrieve the updated faculty members from our sample database to confirm the data insertion action.

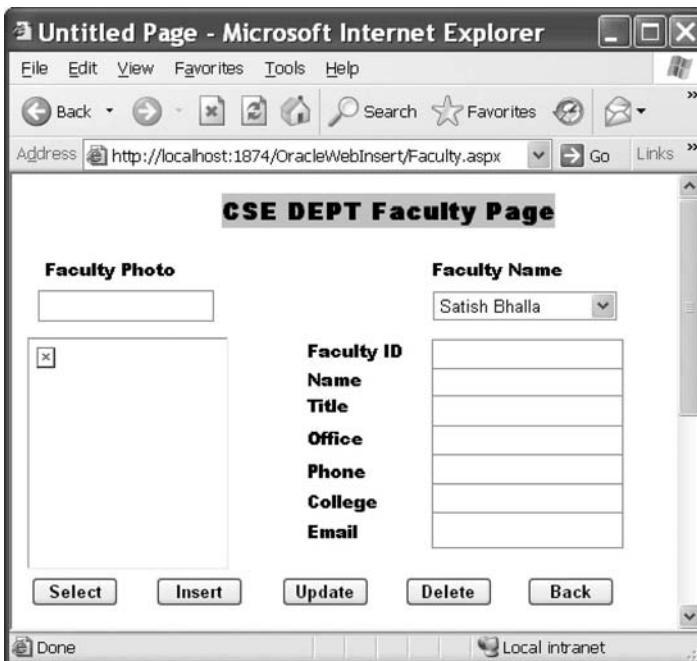


Figure 8.65 Modified Faculty page.

2. Second, we need to modify and add another piece of coding in the ShowFaculty() method to allow the new inserted faculty photo to be displayed as the new inserted data is validated.

Open the Page_Load() method in the Faculty page. Perform the modifications shown in Figure 8.66 to this method. The codes we developed in the previous section have been highlighted with shading.

Let's take a closer look at this piece of modified code to see how it works.

- A. Each time the Faculty page starts, we need to update the combobox control ComboName to reflect any new inserted or updated faculty members against our sample database. To do that, a new user-defined UpdateFaculty() method is added into this Page_Load() method to fulfill this task.
- B. To perform updating the faculty records in our sample database inside the UpdateFaculty() method, first an instance of the Oracle Command class is created.
- C. Then the system ExecuteReader() method is called to read back all updated faculty records from our sample database and assign them to the Oracle Data Reader object.
- D. A while loop is utilized to retrieve each record and add each into the combobox control ComboName.
- E. A cleaning job is performed to release all data objects used in this method.

Now let's take care of the code modifications to the user-defined method ShowFaculty(). The purpose of this modification is to enable users to insert a new faculty image with a new faculty record insertion, and this new inserted faculty image can be displayed as the data insertion validation process to be performed later.

	Faculty	Page_Load()
A		<pre> protected void Page_Load(object sender, EventArgs e) { if (((OracleConnection)Application["oraConnection"]).State != ConnectionState.Open) ((OracleConnection)Application["oraConnection"]).Open(); if (!IsPostBack) { UpdateFaculty(); ComboName.SelectedIndex = 0; } } </pre>
B		<pre> public void UpdateFaculty() { OracleCommand oraCommand = new OracleCommand("SELECT faculty_name FROM Faculty", (OracleConnection)Application["oraConnection"]); </pre>
C		<pre> OracleDataReader oraReader = oraCommand.ExecuteReader(); </pre>
D		<pre> while (oraReader.Read()) { ComboName.Items.Add(oraReader[0].ToString()); } </pre>
E		<pre> oraReader.Close(); oraCommand.Dispose(); } </pre>

Figure 8.66 Modified Page_Load method.

Open the user-defined ShowFaculty() method and make the modifications shown in Figure 8.67 to this method.

Let's take a closer look at these modified codes to see how they work.

- A. An if structure is used to check whether a valid faculty image has been detected. If it has, the detected faculty image file is assigned to the ImageUrl property of the PhotoBox control to display it.
- B. Otherwise, if the global variable FacultyImage stored in the Application state function is an empty string, the user did not want to insert a new faculty image with that data insertion

Faculty	ShowFaculty()
<pre>private string ShowFaculty(string fName) { string FacultyImage; switch (fName) { case "Black Anderson": FacultyImage = "Anderson.jpg"; break; case "Ying Bai": FacultyImage = "Bai.jpg"; break; case "Satish Bhalla": FacultyImage = "Satish.jpg"; break; case "Steve Johnson": FacultyImage = "Johnson.jpg"; break; case "Jenney King": FacultyImage = "King.jpg"; break; case "Alice Brown": FacultyImage = "Brown.jpg"; break; case "Debby Angles": FacultyImage = "Angles.jpg"; break; case "Jeff Henry": FacultyImage = "Henry.jpg"; break; default: FacultyImage = "No Match"; break; } } A if (FacultyImage != "No Match") PhotoBox.ImageUrl = FacultyImage; else B if (((string)Application["FacultyImage"] == string.Empty) (string)Application["FacultyImage"] == null) FacultyImage = "Default.jpg"; C else FacultyImage = (string)Application["FacultyImage"]; D PhotoBox.ImageUrl = FacultyImage; E return FacultyImage; } </pre>	

Figure 8.67 Modified user-defined ShowFaculty method.

action. If that global variable is a null object, no data insertion action has been performed. In either case, a default faculty image is displayed.

- C. If both above situations are not true, which means that the user did insert a new faculty image with that new data insertion action and the new faculty image file has been stored in the global variable FacultyImage, that global variable is assigned to the local variable FacultyImage, which will be displayed later.
- D. The FacultyImage is assigned to the ImageUrl property of the PhotoBox control to display this image.
- E. The FacultyImage string is returned to the calling method.

The functionality of these new added codes is that a default faculty photo file Default.jpg will be assigned to the FacultyImage variable if the global variable FacultyImage is empty. Otherwise the faculty photo file stored in the Application state will be assigned to the FacultyImage variable, which will be displayed later in the PhotoBox image control in the Faculty page.

8.8.3 Create Codes to Insert New Faculty on Faculty Page

Now let's handle developing new codes to the data insertion action. This procedure includes adding new codes to the following three methods:

1. The Insert button's Click method
2. The user-defined InsertParameters() method
3. The FacultyID TextChanged() method

Open the Insert button's Click method by double-clicking on the Insert button from the Faculty Form page window and enter the codes shown in Figure 8.68 into this method. Let's take a closer look at this piece of code to see how it works.

- A. An insert query string is declared here with the Oracle database query syntax. One of the most important differences between the SQL Server and Oracle database query syntax is the assignment operator for the dynamic parameter. Instead of using a LIKE @ symbol before the dynamic parameter, a =: operator is used for the Oracle database in the VALUES clause.
- B. The Command object and some local variables are created since we need to use them to perform this data insertion action.
- C. We need to reserve the faculty image file, that is, the location of this image file, and save it to a local variable since we need it in the data validation process later.
- D. If the content of this faculty image file is empty, which means that the user did not want to insert a new faculty image with this data insertion, a default faculty image file is used since we do not want to keep our Faculty Photo box empty during the validation process.
- E. This faculty image file is stored into an Application state function as a global variable, and it will be used later in the validation process.
- F. The Command object is initialized and built with Connection, CommandType, and CommandText properties.
- G. A user-defined InsertParameters() method is executed to fill out all inserted new parameters for the insert query string declared at the beginning of this method.

Faculty	cmdInsert_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p> <p>F</p> <p>G</p> <p>H</p> <p>I</p> <p>J</p> <p>K</p>	<pre> protected void cmdInsert_Click(object sender, EventArgs e) { string cmdString = "INSERT INTO Faculty (faculty_id, faculty_name, office, phone, college, title, email) " + "VALUES (:faculty_id,:faculty_name,:office,:phone,:college,:title,:email)"; OracleCommand oraCommand = new OracleCommand(); string FacultyImage; int intInsert = 0; FacultyImage = txtPhoto.Text; //reserve the new inserted faculty photo location if (FacultyImage == string.Empty) FacultyImage = "Default.jpg"; Application["FacultyImage"] = FacultyImage; //reserve faculty image for validation PhotoBox.ImageUrl = FacultyImage; oraCommand.Connection = (OracleConnection)Application["oraConnection"]; oraCommand.CommandType = CommandType.Text; oraCommand.CommandText = cmdString; InsertParameters(oraCommand); intInsert = oraCommand.ExecuteNonQuery(); oraCommand.Dispose(); if (intInsert == 0) Response.Write("<script>alert('The data insertion is failed')</script>"); else { cmdInsert.Enabled = false; //disable the Insert button ComboName.Items.Clear(); UpdateFaculty(); } } </pre>

Figure 8.68 Codes for the Insert button's Click method.

- H.** The system method `ExecuteNonQuery()` is called to perform this data insertion action against our sample database.
- I.** A cleaning job is performed to release the object we used in this method.
- J.** The system `ExecuteNonQuery()` method will return a data value to indicate whether this data insertion is successful or not. If a zero is returned, which means that no row or zero row has been inserted into our database and this data insertion has failed, in that case, a warning message is displayed.
- K.** Otherwise, a nonzero value is returned and this means that the data insertion is successful. In order to void multiple duplicated insertions, the Insert button is disabled after this insertion. Also the combobox control `ComboName` is cleaned up and the user-defined `UpdateFaculty()` method is executed to refill that control to update the faculty members stored in that combobox.

The detailed codes of the user-defined `InsertParameters()` method are shown in Figure 8.69.

This piece of code is straightforward and easy to understand. Input parameters or seven pieces of new faculty information are assigned to the associated dynamic parameters in the insert query string with the system `Add()` method.

Finally let's handle the coding for the `FacultyID TextChanged()` method. As we know, in order to void multiple duplicated insertions for the same data, the Insert button should be disabled after a new faculty record has been inserted into the database. The

Faculty	▼	InsertParameters()	▼
<pre>private void InsertParameters(OracleCommand cmd) { cmd.Parameters.Add("faculty_id", OracleType.Char).Value = txtID.Text; cmd.Parameters.Add("faculty_name", OracleType.Char).Value = txtName.Text; cmd.Parameters.Add("office", OracleType.Char).Value = txtOffice.Text; cmd.Parameters.Add("phone", OracleType.Char).Value = txtPhone.Text; cmd.Parameters.Add("college", OracleType.Char).Value = txtCollege.Text; cmd.Parameters.Add("title", OracleType.Char).Value = txtTitle.Text; cmd.Parameters.Add("email", OracleType.Char).Value = txtEmail.Text; } }</pre>			

Figure 8.69 Codes for the user-defined InsertParameters method.

Faculty	▼	txtID_TextChanged()	▼
<pre>protected void txtID_TextChanged(object sender, EventArgs e) { cmdInsert.Enabled = true; //enable the Insert button when the faculty_id is changed } }</pre>			

Figure 8.70 Codes for the txtID_TextChanged method.

question is: When should this Insert button be enabled again? The answer is: When another new faculty record is ready to be inserted into our database. How do we know another new faculty record is ready to be inserted? The answer is: When the `faculty_id` stored in the Faculty ID textbox is changed or different with the previous one. A TextChanged event will be created as soon as the content of the Faculty ID textbox is changed, which means that a new faculty record is ready to be inserted to our database, and the associated TextChanged method will be triggered. Therefore, we need to enable this Insert button as long as this situation happened.

Double-click on the Faculty ID textbox to open this method and enter the code shown in Figure 8.70 into this method.

At this point we have finished all modifications to our new project. Before we can run the project to test the data insertion function, make sure that the following three jobs have been done:

1. Make sure that all faculty image files and a default faculty image file **Default.jpg** has been saved to our default folder in which our Web application project is located. In our application, it is `C:\Chapter 8\OracleWebInsert`.
2. Make sure that the startup page is `LogIn`. To confirm this, right-click on our project icon from the Solution Explorer window, select the **Start Options** item from the pop-up menu. On the opened dialog box, be sure that the **Specific page** radio button is selected and the page `LogIn.aspx` is in that box. Click on the OK button to close this dialog box.
3. Make sure that a faculty named **Ali Mhamed** is not located at the Faculty table in our sample database because we will use this faculty as an example to insert it into our sample database. To confirm that, open the Faculty table from our sample database and delete this record if it is in there. The reason for us to do this is because the database does not allow us to insert the same record more than once, so we must first delete that record before we can insert the same data into the database. If you want to insert other faculty data other than **Ali Mhamed**, you don't need to take this step.

Now click on the Start Debugging button to run the project. Enter the suitable username and password to the LogIn page, and select the Faculty Information from the Selection page to open the Faculty page. First, select any faculty member from the combobox control ComboName and click on the Select button to retrieve back and display all information related to that selected faculty. Then enter the following data as the information for a new faculty member:

- Mhamed.jpg Faculty Photo textbox
- M56789 Faculty ID textbox
- Ali Mhamed Faculty Name textbox
- Professor Title textbox
- MTC-353 Office textbox
- 750-378-3355 Phone textbox
- University of Main College textbox
- amhamed@college.edu Email textbox

Click on the Insert button to insert this new record into the database.

To validate this data insertion, go to the ComboName combobox control, and you can find that the new inserted faculty member **Ali Mhamed** is already there. Click on it to select this faculty and then click on the Select button to retrieve this new inserted record from the database and display it on this page. The inserted record is displayed on this page, which is shown in Figure 8.71.

Our data insertion to the Oracle database is successful. Click on the Back button and then on the Exit button to close our project. A complete Web application project named



Figure 8.71 Data validation process.

OracleWebInsert can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

In the next section, we will discuss how to perform data updating and deleting in the Oracle database via the website.

8.9 DEVELOP ASP.NET WEB APPLICATION TO UPDATE AND DELETE DATA IN ORACLE DATABASES

Because of the coding similarity between the SQL Server and the Oracle databases, we only emphasize the important differences on the coding for these two databases. To save time and space, we want to modify an existing Web application project OracleWebInsert we developed in the previous section to make it as our new project OracleWebUpdateDelete. To do that, open Windows Explorer and create a new folder such as Chapter 8 if you have not created it. Copy the project OracleWebInsert from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1) and paste it in our folder C:\Chapter 8. Rename the project to OracleWebUpdateDelete.

We divide this section into two parts in terms of the coding function:

1. The first part is to modify the new project to perform data updating actions in the Oracle database.
2. The second part is to develop stored procedures to perform data deleting actions in the Oracle database.

Now let's start from the first part—modify the new project so it performs data updating in the Oracle database.

8.9.1 Modify Project to Perform Data Updating

Open the Visual Studio.NET and go to the File|Open Web Site menu item and browse to the folder C:\Chapter 8. Select our new project OracleWebUpdateDelete and then click on the Open button to open it.

The coding process to this page can be divided into the following two operations:

1. Create codes to the Update button's Click method.
2. Create a user-defined UpdateParameters() method.

Let's begin with the first modification.

8.9.1.1 Create Codes to Update Button Click Method

Open the Update button's Click method by double-clicking on the Update button from the Faculty Web form and enter the codes shown in Figure 8.72 to this method.

Let's take a closer look at this piece of code to see how it works.

- A. An updating query string is declared first with the oldName as the name of the dynamic parameter. This is because when you want to update the faculty name, the original name stored in the combobox control ComboName becomes the old name, and we need to

Faculty	cmdUpdate_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p> <p>F</p> <p>G</p> <p>H</p> <p>I</p> <p>J</p> <p>K</p> <p>L</p> <p>M</p>	<pre> protected void cmdUpdate_Click(object sender, EventArgs e) { string cmdString = "UPDATE Faculty SET faculty_name=:name, office=:office, phone=:phone, " + "college=:college, title=:title, email=:email WHERE (faculty_name =: oldName)"; OracleCommand oraCommand = new OracleCommand(); string FacultyImage; int intUpdate = 0; txtID.Text = string.Empty; FacultyImage = txtPhoto.Text; //reserve the new inserted faculty photo location if (FacultyImage == string.Empty) FacultyImage = "Default.jpg"; Application["FacultyImage"] = FacultyImage; //reserve faculty image for validation oraCommand.Connection = (OracleConnection)Application["oraConnection"]; oraCommand.CommandType = CommandType.Text; oraCommand.CommandText = cmdString; UpdateParameters(ref oraCommand); intUpdate = oraCommand.ExecuteNonQuery(); oraCommand.Dispose(); ComboName.Items.Clear(); UpdateFaculty(); txtPhoto.Text = string.Empty; //clean up the faculty photo box if (intUpdate == 0) Response.Write("<script>alert('The data updating is failed')</script>"); } </pre>

Figure 8.72 Coding for the Update button's Click method.

distinguish this old name with the updated faculty name stored in the textbox control txtName.

- B.** All data objects used in this method are created here, and a local integer variable `intUpdate` is also created, which is used as a holder to keep the returned data value from the execution of the `ExecuteNonQuery()` method later.
- C.** Before we can perform data updating, we need to clean up the Faculty ID textbox since we don't want to update this piece of information.
- D.** We need to reserve the faculty image file, that is, the location of this image file, and save it to a local variable since we need it in the data validation process later.
- E.** If the content of this faculty image file is empty, which means that the user did not want to insert a new faculty image with this data insertion, a default faculty image file is used since we do not want to keep our Faculty Photo box empty during the validation process.
- F.** This faculty image file is stored into an `Application` state function as a global variable, and it will be used later in the validation process.
- G.** The `Command` object is initialized with the `Connection` object, `Command` type, and `Command` text.
- H.** The user-defined `UpdateParameters()` method, whose detailed coding is shown in Figure 8.73, is called to assign all input parameters to the `Command` object.
- I.** The `ExecuteNonQuery()` method of the `command` class is called to execute the data updating operation. This method returns a feedback data to indicate whether this data updating is successful or not, and this returned datum is stored to the local integer variable `intUpdate`.

Faculty	▼	UpdateParameters()	▼
<pre>private void UpdateParameters(ref OracleCommand cmd) { cmd.Parameters.Add("name", OracleType.Char).Value = txtName.Text; cmd.Parameters.Add("office", OracleType.Char).Value = txtOffice.Text; cmd.Parameters.Add("phone", OracleType.Char).Value = txtPhone.Text; cmd.Parameters.Add("college", OracleType.Char).Value = txtCollege.Text; cmd.Parameters.Add("title", OracleType.Char).Value = txtTitle.Text; cmd.Parameters.Add("email", OracleType.Char).Value = txtEmail.Text; cmd.Parameters.Add("oldName", OracleType.Char).Value = ComboName.Text; } }</pre>			

Figure 8.73 Coding for the user-defined UpdateParameters method.

- J.** A cleaning job is performed to release all data objects used in this method.
- K.** The combobox control ComboName is cleaned up and the user-defined UpdateFaculty() method is executed to refill that control to update the faculty members stored in that combobox.
- L.** The content of the Faculty Photo textbox is cleaned up to make it ready for that next data insertion action.
- M.** The data value returned from calling the ExecuteNonQuery() is exactly equal to the number of rows that have been successfully updated in the database. If this value is zero, which means that no row has been updated and this data updating has failed, a warning message is displayed for that situation. Otherwise if this value is nonzero, this data updating is successful.

The codes for the user-defined UpdateParameters() method are shown in Figure 8.73. The function of this piece of code is: Seven input parameters are assigned to the Parameters collection property of the Command object using the Add() method.

At this point we have finished all coding development for the data updating action in the Oracle database in the Faculty page. Before we can run the project to test this data updating function, make sure that the starting page is the LogIn page and a default faculty image file Default.jpg has been stored in our default folder. To check the starting page, right click on our project icon from the Solution Explorer window, select the Start Options item from the popup menu, and then check the Specific page radio button and select the LogIn.aspx as the starting page.

Now we can start to run the project to test the data updating function in the Faculty page in our sample Oracle database.

8.9.2 Develop Stored Procedures to Perform Data Deleting

As we discussed at the beginning of this section, to delete a record from a relational database, one must follow the correct sequence. In other words, one must first delete the records that are related to the record to be deleted in the parent table from the child tables. For example, in our application, to delete a record from the Faculty table, one must first delete the related records from the LogIn and the Course tables, and then one can delete the desired record from the Faculty table. The reason for that is because the faculty_id is a primary key in the Faculty table, but it is a foreign key in other tables.

Based on the analysis above, it can be seen that to delete one record from a parent table such as the Faculty table in our sample database, many deleting queries will be executed to first delete related records from the child tables such the LogIn and the Course, and then delete the target record from the parent table. Fortunately, we have set the Cascade mode for our data updating and deleting actions when we built our sample Oracle database CSE_DEPT in Chapter 2. In that case, we did not need to handle those multiple updating and deleting jobs ourselves. Instead the Oracle database engine can do that cascaded updating and deleting for us automatically. However, an easy and flexible way to perform these multiple deleting queries is to use the stored procedure to perform this data deleting.

8.9.2.1 Delete Existing Record from Faculty Table

Recall that in Section 7.8.3.2 in Chapter 7, we discussed how to develop a stored procedure in the Oracle database and use that stored procedure to perform the data deleting operation in the Oracle database. In this section we still want to use our Faculty table as an example to show how to delete an existing record from related tables.

In our sample database, there are two child tables related to our Faculty table, the LogIn table and the Course table. Two child tables are connected with the Faculty table by using the `faculty_id`, which is a primary key in the Faculty table and a foreign key in two child tables. To delete a faculty member from the parent table, or the Faculty table, one must first delete those records that are related to that faculty member in the parent table from the child table, such as from the LogIn and the Course tables, and then one can delete that faculty member from the Faculty table. Basically, this deleting can be divided into the following three steps or three queries:

1. Delete all records related to the faculty member to be deleted in the Faculty table from the LogIn table. In our sample database, only one row is related to each faculty member in the LogIn table.
2. Delete all records that are related to the faculty member to be deleted in the Faculty table from the Course table. In our sample database, there are about four to six records related to each faculty member in the Course table since each faculty can teach four to six courses.
3. Delete the faculty member from the parent or the Faculty table.

These three steps are exactly equivalent to three deleting queries, and we can combine these three queries into a single stored procedure. However, since we have set the Cascade mode for our data updating and deleting actions when we built our Oracle sample database in Chapter 2, we can use only one deleting query to perform this data deleting action. By calling and executing this stored procedure we can easily complete this data deleting operation. The complete data deleting operation can be divided into the following three steps:

1. Develop the stored procedure in the Oracle database to perform the data deleting function.
2. Call the stored procedure from the ASP.NET Web application to perform the data deleting in the Oracle database.
3. Validate the data deleting action after the data deleting operation.

To save time and space, we will not discuss how to create a stored procedure in the Oracle database environment to perform this data deleting operation in this section since we discussed this topic in detail in Section 7.8.3.2 in Chapter 7. Refer to that section to get more detailed materials about this issue. We will use the stored procedure `DeleteFaculty_SP`, which was developed in Section 7.8.3.2 in Chapter 7, to perform the data deleting action in this section.

Therefore, in the following part, we assume that we have finished developing the stored procedure `DeleteFaculty_SP` and we only take care of the coding for steps 2 and 3.

8.9.2.2 Develop Codes for Delete Button Click Method

Open the Delete button's Click method by double-clicking on the Delete button from the Faculty Web form window, and enter the codes shown in Figure 8.74 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. The content of the query string is now equal to the name of the stored procedure `DeleteFaculty_SP` that we created in the Oracle database in Section 7.8.3.2. Refer to that section to get the detailed codes for this stored procedure. When calling a stored procedure, the content of the query string must be equal to the name of the stored procedure.
- B. The data object and the local variable used in this method are declared here. The integer variable `intDelete` is used to hold the returned value of executing the data deleting method `ExecuteNonQuery()` of the Command class later.
- C. The Command object is initialized with the associated objects. The first object is the Connection object `oraConnection` that is stored in the Application state function. The second object is the Command type, and the `StoredProcedure` property must be assigned to this Command type property to make sure that the application will call a stored procedure as a query during the project runs.
- D. The dynamic parameter is initialized with the real parameter, faculty name, which is stored in the combobox control `ComboName`. Note that you must use the faculty name stored in this combobox control, not the faculty name stored in the faculty name textbox control for this dynamic parameter since the latter is an updated faculty name, not an original faculty name.

Faculty	cmdDelete_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p> <p>F</p> <p>G</p>	<pre>protected void cmdDelete_Click(object sender, EventArgs e) { string cmdString = "DeleteFaculty_SP"; OracleCommand oraCommand = new OracleCommand(); int intDelete = 0; oraCommand.Connection = (OracleConnection)Application["oraConnection"]; oraCommand.CommandType = CommandType.StoredProcedure; oraCommand.CommandText = cmdString; oraCommand.Parameters.Add("FacultyName", OracleType.Char).Value = ComboName.Text; intDelete = oraCommand.ExecuteNonQuery(); oraCommand.Dispose(); if (intDelete == 0) Response.Write("<script>alert('The data deleting is failed')</script>"); }</pre>

Figure 8.74 Coding for the Delete button's Click method.

- E.** The `ExecuteNonQuery()` method of the `Command` class is called to run the stored procedure to perform the data deleting operation. This method will return an integer to indicate whether this calling is successful or not.
- F.** A cleaning job is performed to release all objects used in this method.
- G.** The integer value returned from the calling of the `ExecuteNonQuery()` method is equal to the number of rows that have been successfully deleted from the database. If this value is zero, which means that no row has been deleted from the database and this data deleting has failed, a warning message is displayed to indicate this situation. Otherwise if this value is nonzero, at least one row has been deleted from the database and this data deleting is successful.

Now we have finished all coding developments for the data deleting action in the Oracle database for the Faculty page. Before we can run the project to test the data deleting function, make sure that:

- The starting page is the LogIn page.
- A default faculty photo file `Default.jpg` has been stored in our default folder in which our Web application project is located.

Now we can run the project to test the data deleting function. Click on the Start Debugging button to run the project. Enter the suitable username and password to the LogIn page, and select the Faculty Information from the Selection page to open the Faculty page. Then select the faculty member Ying Bai from the combobox control `ComboName` and click on the Select button to retrieve and display this faculty information in the Faculty page. Now click on the Delete button from this page to try to delete this record from the Faculty table in our sample database. Immediately you can find that all seven textboxes that contain the selected faculty information are cleaned up. Does that mean our data deleting successful? To answer this question, let's perform the following steps to confirm this data deleting action.

8.9.2.3 Validate Data Deleting Action

There are two ways to confirm data deleting. The first way is to try to retrieve this deleted faculty record from the database. Our data deleting would be successful if no such faculty information can be found and retrieved from the database. The second way is to open our sample database to check the associated tables to confirm this data deleting.

Let's do this confirmation using the first way. Still in the Faculty page, keep the faculty name Ying Bai selected in the combobox control `ComboName` and click on the Select button to retrieve this faculty information back from the database and display it in the Faculty page. A warning message "No matched faculty found!" is displayed, which means that this faculty record has been successfully deleted from the database.

Next let's open the Oracle database to check the associated tables to confirm this data deleting. Open the Oracle Database 10g XE home page by going to `Start|All Programs|Oracle Database 10g Express Edition|Go To Database Home Page` items. On the opened Login page, enter the username and password such as `CSE_DEPT` and `reback`, and then click on the Login button to open the Home page. Click on the drop-down arrow on the Object Browser icon and select the item `Browse|Tables` to open the Table page.

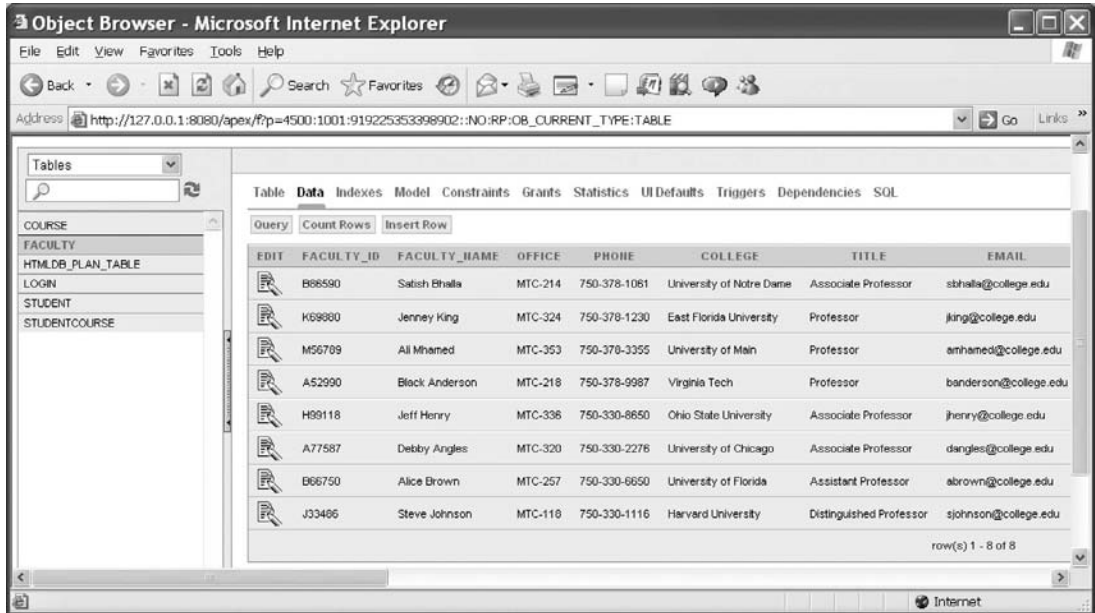


Figure 8.75 Faculty table after the data deleting.

On the opened Table page, click on the FACULTY table from the left pane, and then click on the Data tab to open this table, which is shown in Figure 8.75. You can find that the faculty member Ying Bai with the `faculty_id = B78880` has been deleted from this Faculty table.

As we mentioned before, our sample database is a relational database, and the Faculty table has some relationships with other tables such as LogIn and the Course; that is, the Faculty table has some relationships with all four tables in our sample database, which include the Student and the StudentCourse tables. But at this moment, we only take care of the LogIn and the Course tables, and we will discuss the other two tables in the next section.

Open the LogIn and the Course tables by clicking on each of them one by one from the left pane, and you can find that those records related to the faculty member Ying Bai with a `faculty_id = B78880` have been deleted from the LogIn and the Course tables. The relationship between the Faculty and the LogIn as well as between the Faculty and the Course is set up by the `faculty_id`, which is a primary key in the Faculty table and a foreign key in both LogIn and Course tables. This confirms that our data deleting is successful.

But the story is not finished. As you know, the Faculty table has some relationships with the other four tables in our sample database, which include the Student and the StudentCourse tables. To check this relationship, open the StudentCourse table. You can find that all courses related to (taught by) the faculty member Ying Bai have been deleted from this table, too! These courses include CSC-132B, CSC-234A, CSE-434, and CSE-438. That is not enough. Take a closer look at records in this table. You can find that the students who are identified by the `student_id` and took any of the four courses taught by the deleted faculty member Ying Bai have also been deleted from the

StudentCourse table! Why are those records deleted and who did it? To answer this question, we need to review the building process of our sample Oracle database. Recall that when we built our sample database in Chapter 2, we set up the relationships between four tables by using the foreign and the primary keys. Now let's take a closer look at those elements to try and answer this question in the next section.

8.9.2.4 Constraint Property—On Delete Cascade in Data Table

Recall that in Section 2.11.3 in Chapter 2, we used the constraint property to set up foreign keys and create relationships between tables. When we add a foreign key to a table, we need to indicate the **Constraint Name** and the **Constraint Type**. For example, to create a foreign key for the StudentCourse table and set up a relationship between the Course and the StudentCourse table, we selected the `course_id` as the primary key for the Course table and used it as a foreign key for the StudentCourse table. To create this foreign key to the StudentCourse table, the **Constraint Name** and the **Constraint Type** are:

- STUDENTCOURSE_COURSE_FK
- Foreign Key

The important point is that there is a checkbox named **On Delete Cascade**, which is located at the right of the **Constraint Type** textbox. We checked this checkbox when we created this foreign key for the StudentCourse table. To make this issue clear, we redisplay Figure 2.65 in Chapter 2, as Figure 8.76 in this section.

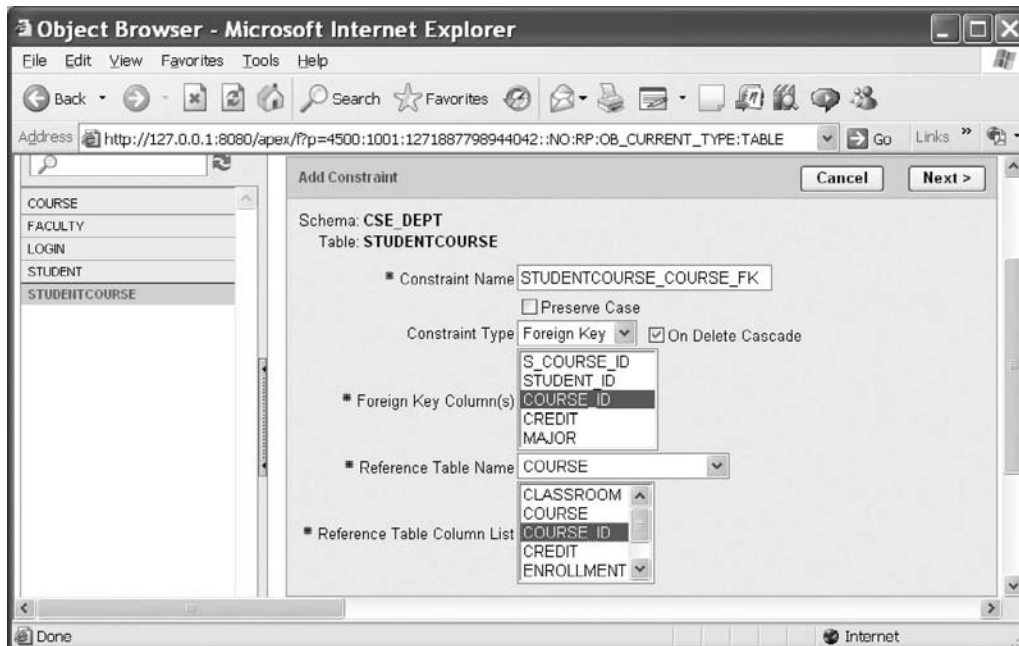


Figure 8.76 Create the foreign key between the StudentCourse and the Course table.

It can be found from this figure that the **On Delete Cascade** checkbox is checked. This means that all records related to this foreign key `course_id` in this `StudentCourse` table will be deleted if the primary key, which is the `course_id` in the `Course` table, is deleted. This is the meaning of the so-called cascaded deleting or **On Delete Cascade**. The word *cascade* means series, and *cascaded deleting* means if records that contain a primary key in a table (parent table) are deleted, all related records that have the same foreign key in all other tables would also be serially deleted.

Now we can answer the question we asked in the last section. All students who are identified by the associated `student_id` and took any of the four courses taught by the deleted faculty member Ying Bai have also been deleted from the `StudentCourse` table. The reason for that is because of the `course_id`, which is a primary key in the `Course` table but a foreign key in the `StudentCourse` table. Since the checkbox **On Delete Cascade** was checked when we set up the relationship between these two tables, all records related to this foreign key `course_id` in the `StudentCourse` table will be serially deleted by the database engine if the records that contain the primary key `course_id` in the `Course` table are deleted. The `faculty_id` in the `Course` table is a foreign key, and when any of the four courses that are identified by their `course_id` and taught by the faculty member Ying Bai are to be deleted from the `Course` table, all records related to that `course_id`, which is a foreign key in the `StudentCourse` table, will also be deleted since the `course_id` is a primary key in the `Course` table. It is the Oracle database engine that performed this cascaded or series data deleting if this checkbox **On Delete Cascade** was checked when the relationship was set up among the tables. Similar things happened to the `student_id`, which is also a foreign key in the `StudentCourse` table.

Before we can close the Oracle database 10g XE, it is highly recommended to recover all deleted records to the associated tables. Refer to Tables 8.13 to 8.16 to add those deleted records back to the associated tables. Now you can close the Oracle Database 10g XE.

Table 8.13 Data to Be Added in the Faculty Table

faculty_id	faculty_name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Associate Professor	ybai@college.edu

Table 8.14 Data to Be Added in the LogIn Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	

Table 8.15 Data to Be Added in the Course Table

course_id	course	credit	classroom	schedule	enrollment	faculty_id
CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
CSE-434	Advanced Electronics Systems	3	TC-213	M-W-F: 1:00-1:55 PM	26	B78880
CSE-438	Advd Logic & Microprocessor	3	TC-213	M-W-F: 11:00-11:55 AM	35	B78880

Table 8.16 Data to Be Added in the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1005	J77896	CSC-234A	3	CS/IS
1009	A78835	CSE-434	3	CE
1014	A78835	CSE-438	3	CE
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE

A complete Web application project OracleWebUpdateDelete can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1).

8.10 CHAPTER SUMMARY

A detailed and completed introduction to the ASP.NET and the .NET Framework, including the ASP.NET 3.5, is provided at the beginning of this chapter. This part is especially useful and important for readers who do not have any knowledge or background in the Web application developments and implementations.

Following the introduction section, a detailed discussion on how to install and configure the environment to develop the ASP.NET Web applications is provided. Some essential tools such as the Web server, IIS, and FrontPage Server Extension 2000, as well as the installation process of these tools, are introduced and discussed in detail.

Starting in Section 8.3, detailed developments and building processes of ASP.NET Web applications to access databases are discussed with seven real Web application projects. Two popular databases, SQL Server and Oracle, are utilized as the target databases for those projects. The seven real ASP.NET Web application projects include:

1. ASP.NET Web application to select and display data from the Microsoft SQL Server database
2. ASP.NET Web application to insert data into the Microsoft SQL Server database
3. ASP.NET Web application to update and delete data in the Microsoft SQL Server database
4. ASP.NET Web application to perform data actions with LINQ to SQL query
5. ASP.NET Web application to select and display data from the Oracle database
6. ASP.NET Web application to insert data into the Oracle database
7. ASP.NET Web application to update and delete data in the Oracle database

The stored procedures are utilized in the last project to help readers to perform the data deleting action against our sample databases more efficiently and conveniently. The detailed discussion on the data deleting order is provided to help readers to understand the integrity constraint built in the relational database. It is a tough topic to update or delete data from related tables in a relational database, and a clear and deep discussion on this topic will significantly benefit readers and improve their knowledge and hands-on experience on these issues.

HOMework

I. True/False Selections

- ___ 1. The actual language used in the communication between the client and the server is HTML.
- ___ 2. ASP.NET and .NET Framework are two different models that provide the development environments to the Web programming.
- ___ 3. The .NET Framework is composed of the Common Language Runtime (called runtime) and a collection of class libraries.
- ___ 4. You access the .NET Framework by using the class libraries provided by the .NET Framework, and you implement the .NET Framework by using the tools such as Visual Studio.NET provided by the .NET Framework, too.
- ___ 5. ASP.NET 3.5 is a programming framework built on the .NET Framework 3.5 and it is used to build Web applications.
- ___ 6. The fundamental component of ASP.NET is the Web Form. A Web Form is the Web page that users view in a browser, and an ASP.NET Web application can contain one or more Web Forms.
- ___ 7. A Web Form is a dynamic page that runs on the server side, and it can access server resources when it is viewed by users via the client browser.
- ___ 8. Similar to traditional Web pages, an ASP.NET 3.5 Web page can only run scripts on the client side.
- ___ 9. The controls you added to the Web form will run on the Web server when this Web page is requested by the user through a client browser.
- ___ 10. To allow a listbox control to response to a user click as the Web page runs, the AutoPostBack property of that listbox must be set to False.

II. Multiple Choices

1. When the user sends a request from the user's client browser to request a Web page, the server needs to build that form and sends it back to the user's browser in the _____ language format.
 - a. ASP.NET
 - b. .NET Framework
 - c. XML
 - d. HTML
2. Once a requested Web page is received by the client's browser, the connection between the client and the server is _____.
 - a. Still active
 - b. Terminated
 - c. Not active
 - d. Either active or inactive
3. As a Web application runs, the programs developed in any .NET-based language are converted into the _____ codes that can be recognized by the CLR, and the CLR can compile and execute the MSIL codes by using the Just-In-Time compiler.
 - a. Visual Studio.NET
 - b. Visual Basic.NET

- c. Microsoft Intermediate Language (MSIL)
 - d. C#
4. The terminal file of an ASP.NET Web application is a _____ file.
- a. Dynamic Linked Library (dll)
 - b. MSIL
 - c. XML
 - d. HTML
5. Because Web pages are frequently refreshed by the server, one must use the _____ to store the global variable.
- a. Global.asax file
 - b. Defaulty.aspx file
 - c. Config file
 - d. Application state function
6. One needs to use the _____ method to display a message box in Web applications.
- a. MessageBox.Show()
 - b. MessageBox.Display
 - c. Java script alert()
 - d. Response.Write()
7. Unlike the Windows-based applications that use the Form_Load as the first event method, a Web-based application uses the _____ as the first event method.
- a. Start_Page
 - b. Page_Load
 - c. First_Page
 - d. Web_Start
8. To delete data from a relational database, one must first delete the data from the _____ tables, and then one can delete the target data from the _____ table.
- a. Major, minor
 - b. Parent, child
 - c. Parent, parent
 - d. Child, parent
9. To allow the SQL Server database engine to delete all related records from the child tables, the Delete Rule item in the INSERT And UPDATE Specifications box of the Foreign Key Relationship dialog box must be set to _____.
- a. No action
 - b. Cascade
 - c. Default
 - d. Null
10. To display any message on a running Web page, one must use the _____ method.
- a. MessageBox.Show()
 - b. Response()
 - c. Response.Redirect()
 - d. Response.Write()

```

Course Page_Load()
protected void Page_Load(object sender, EventArgs e)
{
    if (((OracleConnection)Application["oraConnection"]).State != ConnectionState.Open)
        ((OracleConnection)Application["oraConnection"]).Open();

    if (!IsPostBack)
    {
        ComboName.Items.Add("Ying Bai");
        ComboName.Items.Add("Satish Bhalla");
        ComboName.Items.Add("Black Anderson");
        ComboName.Items.Add("Steve Johnson");
        ComboName.Items.Add("Jenney King");
        ComboName.Items.Add("Alice Brown");
        ComboName.Items.Add("Debby Angles");
        ComboName.Items.Add("Jeff Henry");
    }
}

```

Figure 8.77 Codes for the Page_Load method.

III. Exercises?

1. Write a paragraph to answer and explain the following questions:
 - a. What is ASP.NET?
 - b. What is the main component of the ASP.NET Web application?
 - c. How is an ASP.NET Web application executed?
2. Suppose we want to delete one record from the Student table in our sample database CSE_DEPT based on one student_id = "H10210". List all deleting steps and deleting queries including data deleting from the child and the parent tables.
3. Figure 8.77 shows a piece of code developed in the Page_Load() method. Explain the functionality of the statement **if (!IsPostBack)** block.
4. Add a Web page and develop codes to perform data deleting for the Student form page in the SQLWebUpdateDelete project (the project file can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1)).
5. Add a Web page UpdateCourse and develop the codes to perform the course updating actions for the Course Form page in the project OracleWebUpdateDelete (the project can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1)).
6. Develop a Web page called Student.aspx and create a stored procedure to delete one record from the Student table by using the project OracleWebUpdateDelete (the project can be found from the folder DBProjects\Chapter 8 located at the accompanying ftp site (see Chapter 1)). It is highly recommended to recover those deleted records after they are deleted.

Chapter 9

ASP.NET Web Services

We provided a very detailed discussion of the ASP.NET Web applications in the last chapter. In this chapter, we will concentrate on another ASP.NET related topic—ASP.NET Web Services.

Unlike the ASP.NET Web applications in which the user needs to access the Web server through the client browser by sending requests to the server to obtain the desired information, the ASP.NET Web Services provide an automatic way to search, identify, and return the desired information required by the user through a set of methods installed in the Web server, and these methods can be accessed by a computer program, not by the user, via the Internet. Another important difference between the ASP.NET Web applications and ASP.NET Web Services is that the latter do not provide any graphic user interfaces (GUIs), and users need to create these GUIs themselves to access the Web Services via the Internet.

When finished with this chapter, you will:

- Understand the structure and components of ASP.NET Web Services, such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI).
- Create correct SOAP namespaces for the Web Services to make used names and identifiers unique in the user's document.
- Create suitable security components to protect the Web methods.
- Build professional ASP.NET Web Service projects to access our sample database to obtain required information.
- Build client applications to provide GUIs to consume a Web Service.
- Build professional ASP.NET Web Service projects to access our sample database to insert new information into that database.
- Build professional ASP.NET Web Service projects to access our sample database to update and delete information in that database.

In order to help readers successfully complete this chapter, first we need to provide a detailed discussion of the ASP.NET Web Services and their components.

9.1 WEB SERVICES AND THEIR COMPONENTS

Essentially, Web Services can be considered a set of methods installed in a Web server that can be called by computer programs installed on the client's computer through the Internet. These methods can be used to locate and return the target information required by the computer programs. Web Services do not require the use of browsers or HTML, and therefore Web Services are sometimes called *application services*.

To effectively find, identify, and return the target information required by computer programs, a Web Service needs the following components:

1. XML (Extensible Markup Language)
2. SOAP (Simple Object Access Protocol)
3. UDDI (Universal Description, Discovery, and Integration)
4. WSDL (Web Services Description Language)

The function of each component is discussed below:

XML is a text-based data storage language, and it uses a series of tags to define and store data; that is, the so-called tags are used to “mark up” data to be exchanged between applications. The marked up data then can be recognized and used by different applications without any problem. As you know, the Web Services platform is XML + HTTP (Hypertext Transfer Protocol), and the HTTP protocol is the most popular Internet protocol. However, XML provides a kind of language that can be used between different platforms and programming languages to express complex messages and functions. In order to make the codes used in the Web Services to be recognized by applications developed in different platforms and programming languages, XML is used for the coding in the Web Services to make them up line by line.

SOAP is a communication protocol used for communications between applications. Essentially SOAP is a simple XML-based protocol to help applications developed in different platforms and languages to exchange information over HTTP. Therefore, SOAP is a platform-independent and language-independent protocol, which means that it can be run by any operating system with any programming language. That is, SOAP works as a carrier to transfer data or requests between applications. Whenever a request is made to the Web server to request a Web Service, that request is first wrapped into a SOAP message and sent over the Internet to the Web server. Thus, as the Web Service returns the target information to the client, the returned information is also wrapped into a SOAP message and sent over the Internet to the client browser.

WSDL is an XML-based language for describing Web Services and how to access them. In WSDL terminology, each Web Service is defined as an abstract endpoint or a port and each Web method is defined as an abstract operation. Each operation or method can contain some SOAP messages to be transferred between applications. Each message is constructed by using the SOAP protocol as a request is made from the client. WSDL defines two styles of how a Web Service method can be formatted in a SOAP message: Remote Procedure Call (RPC) and Document. Both RPC and Document style messages can be used to communicate with a Web Service using an RPC.

A single endpoint can contain a group of Web methods, and that group of methods can be defined as an abstract set of operations called a port type. Therefore, WSDL is an XML format for describing network services as a set of endpoints operating on SOAP

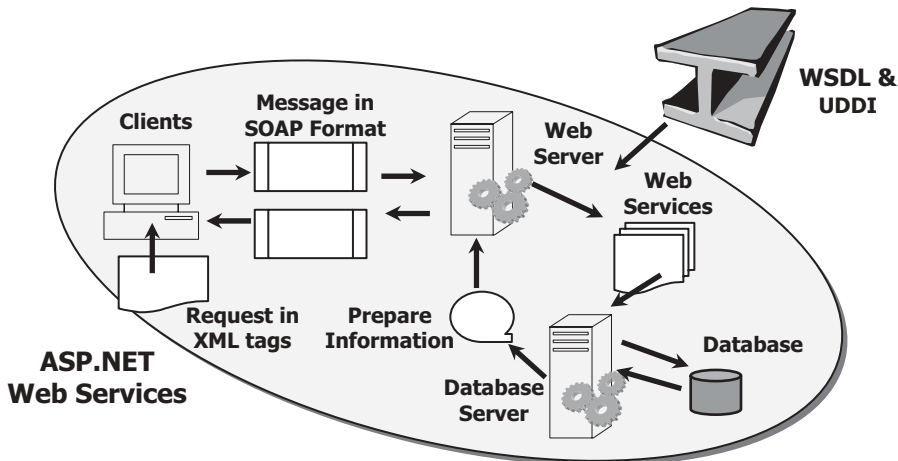


Figure 9.1 Typical process of a Web Service.

messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly and then bound to a concrete network protocol and message format to define an endpoint.

UDDI is an XML-based directory for businesses that list themselves on the Internet. The goal of this directory is to enable companies to find one another on the Web and make their systems interoperable for e-commerce. UDDI is often considered like a telephone book's yellow and white pages. By using those pages, it allows businesses to list themselves by name, products, locations, or the Web services they offer.

Thus, based on these components and their roles, we can conclude:

- XML is used to tag data to be transferred between applications.
- SOAP is used to wrap and pack the data tagged in the XML format into the messages represented in the SOAP protocol.
- WSDL is used to map a concrete network protocol and message format to an abstract endpoint, and describe the Web Services available in an WSDL document format.
- UDDI is used to list all Web Services that are available to users and businesses.

Figure 9.1 shows a diagram to illustrate these components and their roles in an ASP.NET Web Service process.

By now we have obtained the fundamental knowledge about the ASP.NET Web Services and their components. Next let's see how to build a Web Service project.

9.2 PROCEDURES TO BUILD A WEB SERVICE

Different methods and languages can be used to develop different Web Services such as the C# Web Services, Java Web Services, and Perl Web Services. In this section we only concentrate on developing the ASP.NET Web Services using the Visual C#.NET 2008. Before we can start to build a real Web Service project, let's first take a closer look at the structure of a Web Service project.

9.2.1 Structure of a Typical Web Service Project

A typical Web Service project contains the following components:

1. As a new Web Service project is created, two page files and two folders are created under this new project. The folder **App_Code** contains the code-behind page that has all real codes for a simple default Web Service and the Web Service to be created. The folder **App_Data** is used to store all project data.
2. The code-behind page **Service.cs** contains the real Visual C#.NET codes for a simple Web Service. Visual Web Developer includes three default namespace declarations to help users to develop Web Services on the top of this page, which are:

```
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
```

By default, a new code-behind file contains a class named **Service** that is defined with the **WebService** and **WebServiceBinding** attributes. The class defined a default Web method named **HelloWorld** that is a placeholder, and you can replace it with your own method or methods later when you develop your own Web Service project.

3. The main Web Service page file **Service.asmx** is used to display information about the Web Service's methods and provide access to the Web Service's WSDL information. The extension **.asmx** means that this is an Active Service Method file, and the letter **x** is just a rotation of the attached symbol **+** after the keyword **ASP** since the ASP.NET was called **ASP+** in the early days. If you open the **ASMX** file on a disk, you will see that it actually contains only one command line:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.
cs" Class="Service" %>
```

It indicates the programming language in which the Web Service's code-behind file is written, the code-behind file's location, and the class that defines the Web Service. When you request the Active Server Method File (**ASMX**) page through the Internet Information Services (**IIS**), **ASP.NET** uses this information to generate the content displayed in the Web browser.

4. The configuration file **web.config**, which is an XML-based file, is used to set up a configuration for the new created Web Service project, such as the namespaces for all kinds of Web components, Connection string, and default authentication mode. Each Web Service project has its own configuration file.

Of all the files and folders discussed above, the code-behind page is the most important file since all Visual C#.NET codes related to building a Web Service are located in this page, and our major coding development will be concentrated on this page, too.

9.2.2 Real Considerations When Building a Web Service Project

Based on the structure of a typical Web Service project, some issues related to building an actual Web Service project are emphasized here, and these issues are very important and should be followed carefully to successfully create a Web Service project in the Visual Studio.NET environment.

As a request is made and sent from a Windows or Web form client over the Internet to the server, the request is packed into a **SOAP** message and sent to the **IIS** on the client

computer. Then the IIS will pass the request to the ASP.NET to get it processed in terms of the extension `.asmx` of the main service page. ASP.NET checks the page to make sure that the code-behind page contains the necessary codes to power the Web Service, that is, to trigger the associated Web methods to search, find, and retrieve the information required by the client, pack it to the SOAP message, and return it to the client.

During this process, the following detailed procedures must be performed:

1. When ASP.NET checks the received request represented in a SOAP message, the ASP.NET will make sure that the names and identifiers used in the SOAP message are unique. In other words, those names and identifiers cannot be conflicted with any name and identifier used by any other message. To make names and identifiers unique, we need to use our specific namespace to place and hold our SOAP message.
2. Generally, a request contains a set of information, not a single piece of information. To request those pieces of information, we need to create a Web Service proxy class to consume Web Services. In other words, we do not want to develop separate Web methods to query each piece of information, which would make our project's size terribly large if we needed a lot of information. A good solution is to instantiate an object based on that class and integrate these pieces of information into that object. All information can be embedded into that object and can be returned if that object returns. Another choice is to design a Web method to make it return a DataSet, which is a convenient way to return all data.
3. As a professional application, we need to handle the exceptions to make our Web Service as perfect as possible. In that case, we need to create a base class to hold some error-checking codes to protect our real class, which will be instantiated to an object that contains all information we need. So this real class should be a child class inherited from the base class.
4. Since the Web Services do not provide any GUI, we need to develop some GUIs in either Windows-based or Web-based applications to interface to the Web Services to display returned information on GUIs.

Keep these real issues in mind and now let's begin to build a real Web Service project using an ASP.NET Web Service template.

9.2.3 Procedures to Build an ASP.NET Web Service

As we mentioned in the last section, a Web Service is basically composed of a set of Web methods that can be called by the computer programs by the client. To build these methods, generally one needs to perform the following steps:

1. Create a new ASP.NET Web Service project.
2. Create a base class to handle the error checking to protect our real class.
3. Create our real Web Service class to hold all Web methods and codes to response to requests.
4. Add all Web methods into our Web Service class.
5. Develop the detail codes for those Web methods to perform the Web Services.
6. Build a Windows-based and Web-based project to consume the Web Service to pick up and display the required information on the GUI.
7. Store our ASP.NET Web Service project files in a safe location.

In this chapter, we try to develop the following projects to illustrate the building and implementation process of a Web Services project:

- Build a professional ASP.NET Web Service project to access the SQL Server database to obtain required information.
- Build client applications to provide GUIs to consume a Web Service.
- Build a professional ASP.NET Web Service project to insert new information into the SQL Server database.
- Build a professional ASP.NET Web Service project to update and delete information against the SQL Server database.
- Build a professional ASP.NET Web Service project to access the Oracle database to obtain required information.
- Build a professional ASP.NET Web Service project to insert new information into the Oracle database.
- Build a professional ASP.NET Web Service project to update and delete information against the Oracle database.

Based on these procedures, we can start to build our first Web Service project.

9.3 BUILD ASP.NET WEB SERVICE PROJECTS TO ACCESS SQL SERVER DATABASE

To create a new ASP.NET Web Service project, open the Visual Studio.NET 2008, and then go to the File|New Web Site item. On the opened New Web Site dialog box, select the ASP.NET Web Service item from the Templates list and enter our Web Service project name `WebServiceSQLSelect` into the box that is next to the Location box, which is shown in Figure 9.2. Also select the Visual C# from the Language box since we need to develop a Visual C# Web Service project in this chapter.

Note that Visual Studio.NET 2008 introduced a Web project model that can use either IIS or the Local File System to develop Web applications. This model is good only when developing ASP.NET Web Services and Web pages that are running locally on a pseudo-Web server. This is our situation since we will run our Web Service in our local machine and use it as a development server, so the File System is used for our server location, which is shown in Figure 9.2.

Click on the OK button to create this new project in our default folder `C:\Chapter 9`.

9.3.1 Files and Items Created in the New Web Service Project

After this new Web Service project is created, four items are produced in our new project in the Solution Explorer window. As we discussed in the last section, the main service page file `Service.asmx`, which is used to display information about the Web Service's methods and provide access to the Web Service's WSDL information, and the configuration file `web.config`, which is used to set up a configuration for our new Web Service project, such as the namespaces for all kinds of Web components, connection strings for

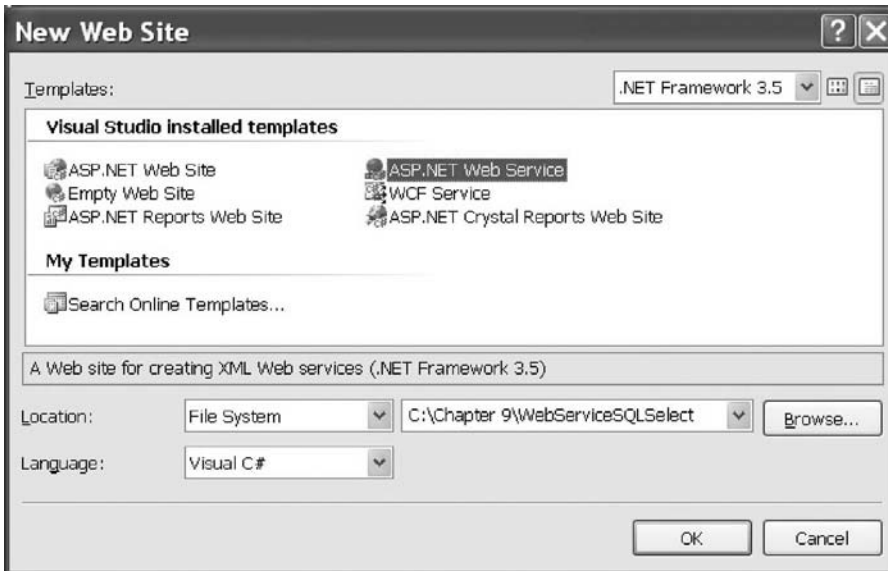


Figure 9.2 Create a new Web Service project.

data components, and Web Services and Windows authentication mode, are automatically created and added into our new project. More important, the page file **Service.aspx** is designed to automatically create extensible WSDL, dispatch Web methods, serialize and deserialize parameters, and provide hooks for message interception within our applications. However, our default file **Service.aspx** only contains a compile directive as this new Web Service project is created and opened from the File System.

Two folders, named **App_Code**, which is used to store our code-behind page **Service.cs**, and **App_Data**, which is used to save the project data, are also created. The code-behind page **Service.cs** is the place we need to create and develop the codes for our Web Services. This page contains a default class named **Service** that is defined with the **WebService** and **WebServiceBinding** attributes. The class defined as a default Web method named **HelloWorld** is a placeholder, and we can replace it with our own method or methods later based on the requirement of our Web Service project.

Now double-click on this code-behind page **Service.cs**, which is shown in Figure 9.3, and let's take a closer look at the code in this page.

- A. The Web Services–related namespaces that contain the Web Service components are added into this page to allow us to access and use those service-related components to build our Web Service project. A detailed description about those namespaces and their functions is shown in Table 9.1.
- B. Some **WebService** attributes are defined in this part. Generally, **WebService** attributes are used to identify additional descriptive information about deployed Web Services. The namespace attribute is one of examples. As we discussed in the last section, we need to use our own namespace to store and hold names and identifiers used in our SOAP messages to distinguish them from any other SOAP messages used by other Web Services. Here, in this new project, Microsoft provided a default namespace "http://tempuri.org/", which is a temporary system-defined namespace to identify all Web Services code generated


```

A using System;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;

B [WebService(Namespace = "http://tempuri.org/")]
C [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
D // [System.Web.Script.Services.ScriptService]
E public class Service : System.Web.Services.WebService
F {
    public Service () {

        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
G public string HelloWorld() {
        return "Hello World";
    }
}

```

Figure 9.3 Default coding for the code-behind page Service.cs.

Table 9.1 Web Service Namespaces

Namespace	Functionality
System.Web	Enable browser and server communication using the .Net Framework
System.Web.Services	Enable creations of XML Web services using ASP.NET
System.Web.Services.Protocol	Define the protocol used to transmit data across the wire during the communication between the Web Service clients and servers

by the .NET framework, to store this default Web method. We need to use our own namespace to store our Web methods later when we deploy our Web Services in a real application.

- C.** This Web Service Binding attribute indicates that the current Web Service complies with the Web Services Interoperability Organization (WS-I.org) Basic Provide 1.1. Here, basically, a binding is equivalent to an interface in which it defines a set of concrete operations.
- D.** This commented attribute indicates that if this Web Service is called from any script language, such as ASP.NET AJAX, the associated namespace ScriptService should be used, and this coding line should be uncommented.
- E.** Our Web Service class **Service** is a child class derived from the parent class **WebService** located in the namespace **System.Web.Services**.
- F.** The constructor of our **Service** class contains an **InitializeComponent()** method used to initialize all user-defined components used in this Web Service project. Generally, we do not need to create any specific components for most of our Web Service projects, therefore we keep the comment for this method.

- G.** The default Web method `HelloWorld` is defined as a global function and this function returns a string “Hello World” when it is returned to the client.

Next let’s double-click on the main service page file `Service.asmx`, which is the entry point of our project to open it. This one line code contains only a compile directive shown below since this project is created and opened using a File System:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.
cs" Class="Service" %>
```

As we mentioned in the last section, this code indicates the programming language in which the Web Service’s code-behind file is written, the code-behind file’s name and location, and the class that defines the Web Service. In a real application, both the code-behind file name and the class name should be renamed to match our file and class names used in our project, respectively. We will do those renames later in the following sections. However, first let’s run the default `HelloWorld` Web Service project to get a feeling about what it looks like and how it works.

9.3.2 Feeling of Hello World Web Service Project as It Runs

Click on the Start Debugging button to run the default `HelloWorld` project. After the project is running, a message box is displayed with a warning message displayed, which is shown in Figure 9.4.

Generally, a Web Service project should not be debugged when it is deployed, and this is defined in the `web.config` file with a control of disabling the debugging. However, the debugging can be enabled during the development process by modifying the `web.config` file. To do that, keep the default radio button selected in this warning message box and click on OK to continue to run our project. Our `Service.asmx` page should be the starting page and the following IE page is displayed as shown in Figure 9.5.

This page displays the Web Service class named `Service` and all Web methods or operations developed in this project. By default, only one method `HelloWorld` is created and used in this project.

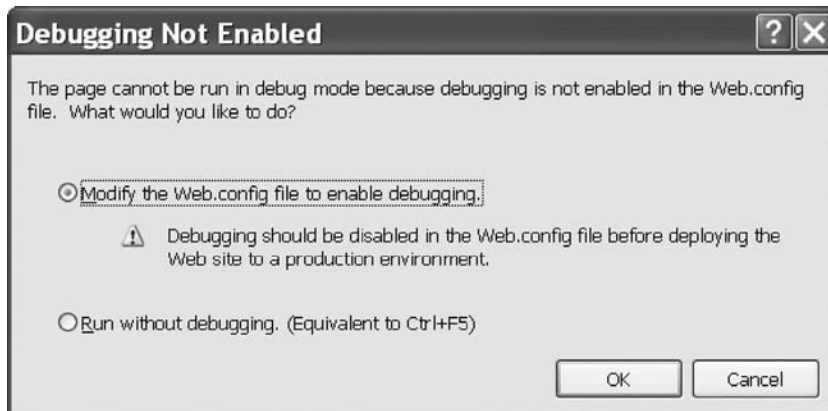


Figure 9.4 Debugging Not Enabled message box.

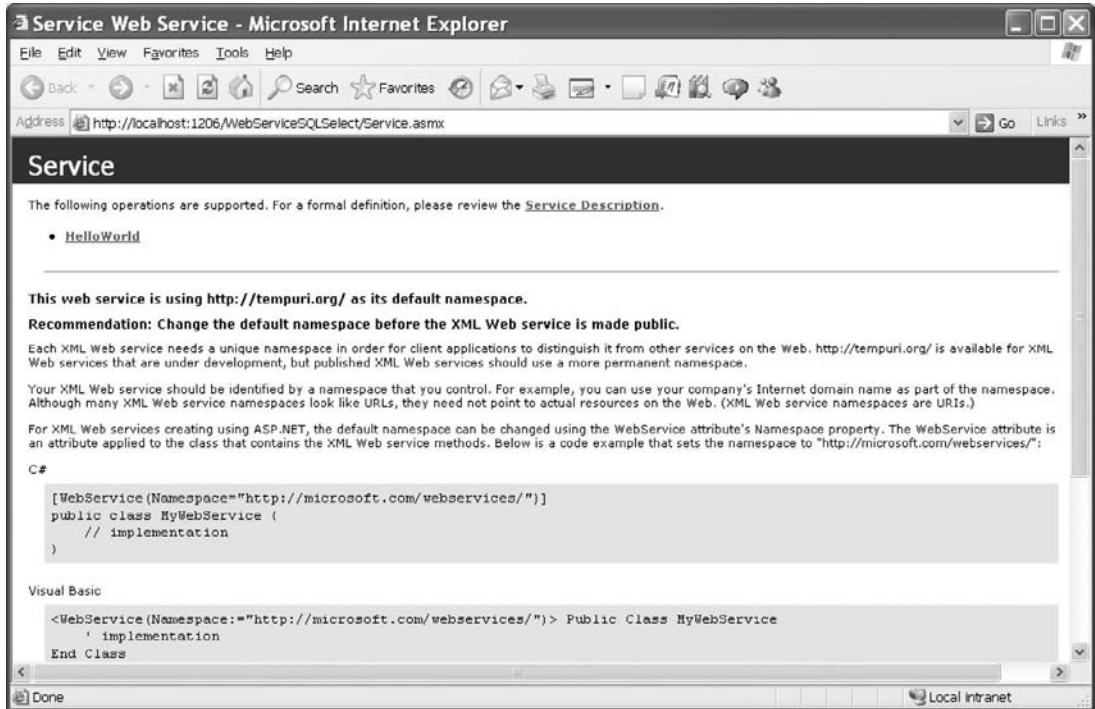


Figure 9.5 Running status of the default Web service project.

Below the method, the default namespace in which the current method or operation is located is shown, and a recommendation that suggests that we create our own namespace to store our Web Service project is displayed. Following this recommendation, some example namespaces used in C#, Visual Basic, and C++ are listed.

Now let's access our Web Service by clicking on the HelloWorld method. The test method page appears, which is shown in Figure 9.6.

The Invoke button is used to test our HelloWorld method using the HTTP Protocol. Below the Invoke button, some message examples created by using the different protocols are displayed. These include the requesting message and responding message created in SOAP 1.1, SOAP 1.2, and HTTP Post. The placeholder located at the default namespace "http://tempuri.org/" should be replaced by the actual namespace when this project is modified to a real application.

Now click on the Invoke button to run and test the default method HelloWorld. As the Invoke button is clicked on, a URL that contains the default namespace and the default HelloWorld method's name is activated, and a new browser window, shown in Figure 9.7, is displayed. When the default method HelloWorld is executed, the main service page Service.asmx sends a request to the IIS. Furthermore, the IIS sends it to the ASP.NET runtime to process this request based on that URL.

The ASP.NET runtime will execute the HelloWorld method and pack the return data as a SOAP message, and send it back to the client. The returned message contains only a string object, that is, a string of "Hello World" for this default method.

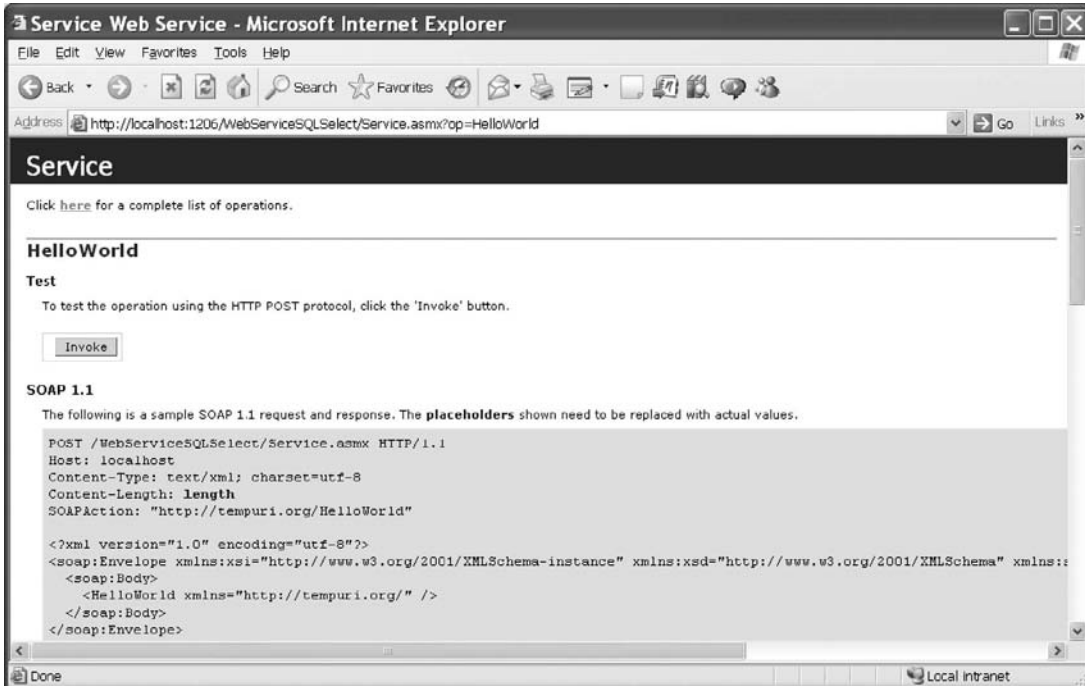


Figure 9.6 Test method page.



Figure 9.7 Running status of the default method.

In this returned result, the version and the encoding of the used XML code is indicated first. The `xmlns` attribute is used to indicate the namespace used by this string object that contains only a string of "Hello World".

As we discussed in the previous section, ASP.NET Web Service did not provide any GUI, so the running result of this default project is represented using the XML codes in some Web interfaces we have seen. This is because those Web interfaces are only provided and used for the testing purpose for the default Web Service. In a real application, no such Web interface will be provided and displayed.

Click on the **Close** button located on the upper-right corner of the browser for two browser pages to close them. At this point, we should have a basic understanding and feeling about a typical Web Service project and its structure as well as its operation process. Next we will create our own Web Service project by modifying this default project to perform the request to our sample database, that is, to the Faculty table, to get the desired faculty information.

We will develop our Web Service project in the following sequence:

1. Modify the default Web Service project to make it our new Web Service project.
2. Create a base class to handle error-checking codes to protect our real Web Service class.
3. Create our real Web Service class to hold all Web methods and codes to respond to requests to pick up desired faculty information.
4. Add Web methods into our Web Service class to access our sample database.
5. Develop the detail codes for those Web methods to perform the Web Services.
6. Build a Windows-based and a Web-based project to consume the Web Service to pick up and display the required information on the GUI.
7. Deploy our completed Web Service to Internet Information Service (IIS).

The modifications defined in step 1 include the rename of the main service page's name, the code-behind page's name, the class name, and the namespace defined in the code-behind page and the main service page.

Let's start with the step 1.

9.3.3 Modify Default Web Service Project

We will modify a default Web Service project to make it our new Web Service project and allow it to access our sample SQL Server database to pick up the desired faculty information.

The following modifications must be made to this default project:

- Rename the main service page from the default name `Service` to our new name `WebServiceSQLSelect`.
- Rename the code-behind page's name from `Service.cs` to our new name `WebServiceSQLSelect.cs`.
- Modify the code-behind page's name and class name in the main service page.
- Rename the namespace defined in the code-behind page.
- Add both reference and namespace `System.Windows.Forms` to this Web Service project since we need to test our project using the `MessageBox()` method, and this method is involved in that namespace.
- Add some other namespaces related to the System Data components and SQL Server Data Provider since we need to use them to perform data actions in our sample database.

Let's start these modifications from step 1 listed above.

Right-click on the main service page `Service.asmx` from the Solution Explorer window and select the **Rename** item from the pop-up menu, and rename this page to a new name `WebServiceSQLSelect.asmx`. Perform a similar operation to the code-behind page `Service.cs` and rename it to `WebServiceSQLSelect.cs`.

Now open our new main service page `WebServiceSQLSelect.asmx` by double-clicking on it, and change the compiler directive from

```
CodeBehind="~/App_Code/Service.cs"
```

to

```
CodeBehind="~/App_Code/WebServiceSQLSelect.cs"
```

Also change the class name from `Class="Service"` to `Class="WebServiceSQLSelect"`.

Go to the **File|Save All** menu item to save these modifications. Now open our new code-behind page `WebServiceSQLSelect.cs` by double-clicking on it from the Solution Explorer window, and perform the modifications shown in Figure 9.8 in the **bold** to this page.

Let's take a closer look at this piece of modified code to see how it works.

- A. We need to use our own namespace to replace the default namespace used by Microsoft to tell the ASP.NET runtime the location from which our Web Service can be found and loaded as it runs. This specific namespace is unique because it is the home page of IEEE Press appended with this book's ISBN number. In fact, you can use any unique location as your specific namespace to store your Web Service project if you like.
- B. The default class `Service` is replaced by a new class `WebServiceSQLSelect`, which is our desired Web Service class used in this new project.
- C. The name of the constructor of our new class is also modified.

```

using System;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;

A [WebService(Namespace = "http://www.ieee.org/9780521712354/")]
  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
  // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
  // [System.Web.Script.Services.ScriptService]
B public class WebServiceSQLSelect : System.Web.Services.WebService
C {
    public WebServiceSQLSelect()
    {
        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}

```

Figure 9.8 Modified code-behind page.

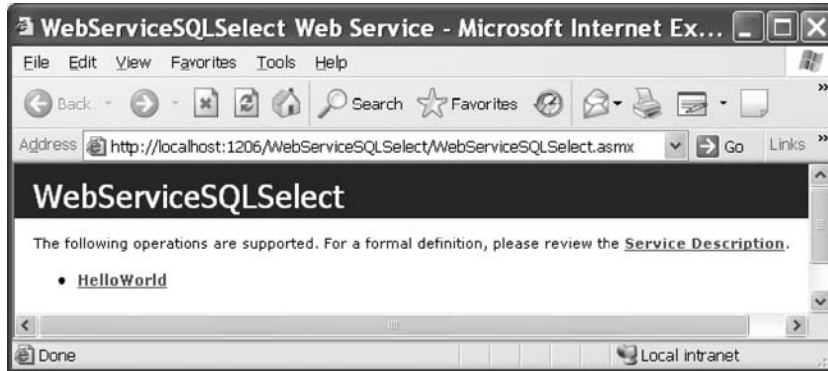


Figure 9.9 Running status of our new Web Service project.

Now, if we run our new project, a default Web interface is displayed with our new project name, `WebServiceSQLSelect`, as shown in Figure 9.9.

If you click on the default method `HelloWorld` and then on the `Invoke` button to test that method, you can find that the namespace has been updated to our new specific namespace. Now, let's add a system reference `System.Windows.Forms` to our project. Right-click on our project `WebServiceSQLSelect` located at the Solution Explorer window and click on the `Add Reference` item from the pop-up menu to open the `Add Reference` dialog box. Scroll down along this list until you find the item `System.Windows.Forms`, click on this item to select it, and then click on `OK` to add it into our project. Then add the following namespaces into the namespace area on this page:

```
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
```

The first namespace contains the definitions of all Windows-based applications' controls and methods including `MessageBox()`. The following three namespaces contain the prototypes of all Data objects, SQL Server Data Provider, and Data Configuration classes used for general data actions between the Web Services and databases.

9.3.4 Create a Base Class to Handle Error Checking for Our Web Service

In this section we want to create a parent class or base class and use it to handle some possible errors or exceptions as our project runs. It is possible that our requests cannot be processed and returned properly. One of the most possible reasons for that is the security issue. To report any errors or problems that occurred in the processing of requests, a parent or base class is a good candidate to perform those jobs. We name this base class as `SQLSelectBase`, and it has two member data:

- `SQLRequestOK` As Boolean: True if the request is fine, otherwise a False is set.
- `SQLRequestError` As String: A string used to report the errors or problems.

```

SQLSelectBase
SQLSelectBase()

public class SQLSelectBase
{
    public bool SQLRequestOK;
    public string SQLRequestError;
    public SQLSelectBase()
    {
        //
        // TODO: Add constructor logic here
        //
    }
}

```

Figure 9.10 Class member data.

To create a new base class in our new project, right-click on our new service project located at the top of the Solution Explorer window, and select the item **Add New Item** from the pop-up menu. On the opened **Add New Item** dialog box, select the **Class** item from the Template list, and enter **SQLSelectBase.cs** into the **Name** box as our new class name. Then click on the **Add** button to add this new class into our project. Click on **Yes** to the message box to place this new class into the **App_Code** folder in our new Web Service project.

Now double-click on this new added class and enter the following codes shown in Figure 9.10 in bold into this class as the class member data. Two public class member data, **SQLRequestOK** and **SQLRequestError**, are added into this new base class. These two data will work together to report possible errors or problems during the request processing.

9.3.5 Create Real Web Service Class

Now we need to create our real Web Service class, which will be instantiated and returned to us with our required information as the project runs. This class should be a child class of our base class **SQLSelectBase** we just created. We name this class as **SQLSelectResult**.

Right-click on our new Web service project from the Solution Explorer window, and select the item **Add New Item** from the pop-up menu. On the opened **Add New Item** dialog box, select the **Class** item from the Template list. Enter **SQLSelectResult.cs** into the **Name** box as the name for this new class, and then click on the **Add** button to add this new class into our project.

Click on **Yes** to the message box to place this new class into the **App_Code** folder in our new Web Service project. Double-click on this new added class and enter the codes shown in Figure 9.11 into this class as the member data to this class. The new added codes have been highlighted in bold.

Since this class will be instantiated to an object that will be returned to us with our desired faculty information as the Web method is called, all desired faculty information should be added into this class as the member data. When we make a request to this Web service project, and furthermore, to our sample database, the following desired faculty information should be included and returned:


```

SQLSelectResult
SQLSelectResult()

public class SQLSelectResult:SQLSelectBase
{
    public string FacultyID;
    public string FacultyOffice;
    public string FacultyPhone;
    public string FacultyCollege;
    public string FacultyTitle;
    public string FacultyEmail;

    public SQLSelectResult()
    {
        //
        // TODO: Add constructor logic here
        //
    }
}

```

Figure 9.11 Member data for the child class SQLSelectResult.

- Faculty id
- Faculty office
- Faculty phone
- Faculty college
- Faculty title
- Faculty email

All this information, which can be exactly mapped to all columns in the Faculty table in our sample database, is added into this class as the member data. Although this does not look like a professional schema, that is true. A better option is that we do not need to create any class that will be instantiated to an object to hold this information. Instead, we can use a DataSet to hold this information and allow the Web method to return that DataSet as a whole package for those pieces of faculty information. However, that better option is relatively complicated compared with our current class. Therefore at this moment we prefer to start our project with an easier method. Later on we can discuss how to use the DataSet to return our desired information.

As we mentioned before, this class is a child class of our base class SQLSelectBase. In other words, this class is inherited from that base class. All six pieces of faculty data is declared here as the member data for this class.

Next we need to take care of our Web method, which will response to our request and return our desired faculty information to us as this method is called.

9.3.6 Add Web Methods into Our Web Service Class

Before we can add a Web method to our project and perform the coding for it, we want to emphasize an important point that is easy to be overlooked by users, that is, the Web Service class and those user-defined classes we just created in the last sections.

The Web Service class WebServiceSQLSelect is a system class, and it is used to contain all codes we need to access the Web Service and Web methods to execute our requests. The base class SQLSelectBase and the child class SQLSelectResult were created

by us and they belong to the application classes. These application classes will be instantiated to the associated objects that can be used by the Web methods developed in the system class `WebServiceSQLSelect` to return the requested information as the project runs. Keep this difference in mind, and this will help us understand them better as we develop a new Web Service project.

We can modify the default method `HelloWorld` and make it as our new Web method in our system class `WebServiceSQLSelect`. This method will use an object instantiated from the application class `SQLSelectResult` we created in the previous section to contain and return the faculty data we required.

9.3.7 Develop the Codes for Web Methods to Perform the Web Services

The name of this Web method is `GetSQLSelect`, and it contains an input parameter `Faculty Name` with the following functions as this method is called:

1. Set up a valid connection to our sample database.
2. Create all required data objects and local variables to perform the necessary data operations later.
3. Instantiate a new object from the application class `SQLSelectResult` and use it as the returned object that contains all required faculty information.
4. Execute the associated data object's method to perform the data query to the `Faculty` table based on the input parameter—`Faculty Name`.
5. Assign each piece of acquired information obtained from the `Faculty` table to the associated class member data defined in the class `SQLSelectResult`.
6. Release and clean up all data objects used.
7. Return the object to the client.

9.3.7.1 Web Service Connection Strings

Among these functions, function 1—set up a valid connection—is the most challenging task. There are two ways to perform this database connection in a Web Service application. One way is to directly use the connection string and the `Connection` object in the Web Service class as we did in the previous projects. Another way is to define the connection string in the `web.config` file. The second way is a better way since the `web.config` file provides an attribute `<connectionStrings/>` for this purpose, and ASP.NET 3.5 recommends storing the data components' connection string in the `web.config` file.

In this project, we will use the second way to store our connection string. To do that, open the `web.config` file by double-clicking on it, and then enter the following codes into the attribute `<connectionStrings/>`:

```
<connectionStrings>
  <add name="sql_conn" connectionString="Server=localhost\
    SQLEXPRESS;
    Integrated Security=SSPI;Database=
    C:\database\SQLServer\CSE_DEPT.
    mdf;" />
</connectionStrings>
```

The following important points should be noted when creating this connection string:

1. This `connectionStrings` attribute must be written in a **single line** in the `web.config` file. Because of the limitation of the space in this page, we used two lines to represent this attribute. However, in your real coding, you must place this attribute in a single line in your `web.config` file; otherwise a grammar problem would be encountered.
2. Web Services that require a database connection in this project use SQL Server authentication with a login ID and password for a user account. However, because we used Windows Authentication Mode when we created our sample database in Chapter 2, we do not need any login ID and password for the database connection in our application. One important issue is the database we are using is not a real SQL Server 2005 database. Instead we are using SQL Server 2005 Express. Therefore we have to add the `InstanceName` of our database, which is `SQLEXPRESS`, into this connection string to tell the ASP.NET runtime to make the correct connection. Attach this instance name after the `localhost` in the `ServerName` item.

To test and confirm this `connectionString`, we can develop some codes and modify the coding of the default `HelloWorld Web` method in the code-behind page to do that. Close the `web.config` file and open the code-behind page `WebServiceSQLSelect.cs` by double-clicking on it from the Solution Explorer window, and then enter the codes into this page shown in Figure 9.12.

All modified codes have been highlighted in bold, and let's see how this piece of code works to test our connection string defined in the `web.config` file.

- A. Four namespaces that contain the prototypes and definitions of Windows-based controls, SQL Server Data Provider, and configuration classes are added into this page since we need to use them to perform the testing of our connection and data actions against our sample database via this Web Service project.
- B. The `ConnectionStrings` property of the `ConfigurationManager` class is used to pick up the connection string we defined in the `web.config` file, which can be considered as a default connection configuration. The connection name `sql_conn` works as an argument for this property and must be identical with the name we defined for the connection name in the `web.config` file. When this property is used, it returns a `ConnectionStringSettingsCollection` object containing the contents of the `ConnectionStringsSection` object for the current application's default configuration, and a `ConnectionStringSection` object contains the contents of the configuration file's `connectionStrings` section.
- C. A new SQL Connection object is created and initialized with the connection string we obtained above.
- D. The `Open()` method of the SQL Connection object is executed to try to open our sample database and set up a valid connection.
- E. By checking the `State` property of the Connection object, we can determine whether this connection is successful or not. If the `State` property is not equal to the `ConnectionState.Open`, which means that a valid database connection has not been installed, a warning message is displayed.
- F. Otherwise the connection is successful, a successful message is displayed, and the connection is closed.

```

A using System;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;
A using System.Data;
A using System.Data.SqlClient;
A using System.Configuration;
A using System.Windows.Forms;

[WebService(Namespace = "http://www.ieee.org/9780521712354/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
// [System.Web.Script.Services.ScriptService]
public class WebServiceSQLSelect : System.Web.Services.WebService
{
    public WebServiceSQLSelect()
    {
        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld()
    {
B         string cmdString = ConfigurationManager.ConnectionStrings["sql_conn"].ConnectionString;
C         SqlConnection sqlConnection = new SqlConnection();
C         sqlConnection.ConnectionString = cmdString;
D         sqlConnection.Open();
E         if (sqlConnection.State != System.Data.ConnectionState.Open)
F             MessageBox.Show("Database Open is failed");
        else
        {
            MessageBox.Show("Database Open is successful");
            sqlConnection.Close();
        }
        return "Hello World";
    }
}

```

Figure 9.12 Modified coding to test the connection string.

Now you can run the project by clicking on the Start Debugging button. Click the HelloWorld method from the built-in Web interface, and then click on the Invoke button to execute that method to test our database connection.

A successful message should be displayed if this connection is fine. Click on the OK button on the message box, and you can get the returned result from the execution of the method HelloWorld.

An issue is that when you run this project, it may take a little while to complete this database connection. The reason for that is because the MessageBox() is used, and it is displayed behind the current Web page when it is activated. So you need to move the current page by dragging it down and then you can find that MessageBox. Click on OK to that MessageBox, and the project will be continued and the running result can be displayed.

Another issue is that this piece of code is only used for the testing purpose, and we will modify this piece of code and place it into a user-defined function called SQLConn() later when we develop our real project.

9.3.7.2 Modify Existing Web Method

Now let's start to take care of our Web methods. In this project, we want to modify the default method HelloWorld as our first Web method and develop codes for this method to complete those function (2 to 7) listed at the beginning of Section 9.3.7.

Open the Web Service code-behind page if it is not opened, and make the following modifications:

1. Change the Web method's name from HelloWorld to GetSQLSelect.
2. Change the data type of the returned object of this method from string to SQLSelectResult, which is our child application class we developed before.
3. Add a new input parameter FacultyName as an argument to this method using Passing-By-Value mode.
4. Create a new object based on our child application class SQLSelectResult and name this object as SQLResult.
5. Create the following data components used in this method:
 - a. SQL Command object sqlCommand
 - b. SQL Data Reader object sqlReader
6. Replace the default returned object in the method from the "Hello World" string to the new created object SQLResult.
7. Move the connection testing codes we developed in this section into a user-defined method SQLConn().

Your finished Web method GetSQLSelect() is shown in Figure 9.13.

```

WebServiceSQLSelect
GetSQLSelect()

[WebMethod]
A public SQLSelectResult GetSQLSelect(string FacultyName)
  {
    B   SqlConnection sqlConnection = new SqlConnection();
    C   SQLSelectResult SQLResult = new SQLSelectResult();
    D   SqlCommand sqlCommand = new SqlCommand();
    E   SqlDataReader sqlReader;

    F   return SQLResult;
  }

G protected SqlConnection SQLConn()
  {
    string cmdString = ConfigurationManager.ConnectionStrings["sql_conn"].ConnectionString;
    SqlConnection conn = new SqlConnection();
    conn.ConnectionString = cmdString;
    conn.Open();
    if (conn.State != System.Data.ConnectionState.Open)
    {
      MessageBox.Show("Database Open is failed");
      conn = null;
    }
    return conn;
  }

```

Figure 9.13 Modified Web method—GetSQLSelect.

Let's take a closer look at this piece of modified code to see how it works.

- A.** Modification steps 1, 2, and 3 listed above are performed at this line. The method's name and the returned data type are changed to `GetSQLSelect` and `SQLSelectResult`, respectively. Also an input parameter `FacultyName` is added into this method as an argument.
- B.** Modification step 4 is performed at this line, and an instance of the application class `SQLSelectResult` is created here.
- C.** Modification step 5 is performed at this line and two SQL data objects are created: `sqlCommand` and `sqlReader`.
- D.** Modification step 6 is performed at this line, and the original returned data is changed to the current object `SQLResult`.
- E.** Modification step 7 is performed here and a new user-defined method `SQLConn()` is created with the codes we developed to test the connection string in the previous section.
- F.** If this connection fails, a warning message is displayed and the returned `Connection` object is assigned with a `null` object. Otherwise a successful `Connection` object is assigned to the returned `Connection` object `conn`.
- G.** The `Connection` object is returned to the Web method.

Next we need to develop codes to execute the associated data object's method to perform the data query to the `Faculty` table and place these codes into the space between steps **C** and **D** in Figure 9.13.

9.3.7.3 Develop Codes to Perform Database Queries

To perform the database query via our Web Service project, we need to perform the following coding operations:

- Add the main codes to perform the data query into our Web method.
- Create a user-defined method, `FillFacultyReader()`, to handle the data assignments to our returned object.
- Create an error or exception-processing method, `ReportError()`, to report any errors encountered when the project runs.

Let's first concentrate on adding the codes into the space between steps **C** and **D** in Figure 9.13 to perform the data query to our sample database `CSE_DEPT`.

Open our Web method and add the codes shown in Figure 9.14 into this method. The codes we developed in the previous sections have been highlighted with shading.

Let's take a closer look at these new added codes to see how they work.

- A.** The query string is declared at the beginning of this method. One point you need to note is that a space must be attached at the end of the first part of this query string. In other words, after the "... FROM Faculty" this space works as a separator between the first and the second part of this query string. The query function could not be executed correctly without this space.
- B.** Initially we assume that our Web method works fine by setting the Boolean variable `SQLRequestOK`, which we defined in our base class `SQLSelectBase`, to `true`. This variable will keep this value until an error or exception is encountered.

```

WebServiceSQLSelect GetSQLSelect()
[WebMethod]
public SQLSelectResult GetSQLSelect(string FacultyName)
{
    SqlConnection sqlConnection = new SqlConnection();
    SQLSelectResult SQLResult = new SQLSelectResult();
    SqlCommand sqlCommand = new SqlCommand();
    SqlDataReader sqlReader;
A   string cmdString = "SELECT faculty_id, office, phone, college, title, email FROM Faculty " +
B   "WHERE faculty_name LIKE @facultyName";
C   SQLResult.SQLRequestOK = true;
D   sqlConnection = SQLConn();
   if (sqlConnection == null)
   {
       SQLResult.SQLRequestError = "Database connection is failed";
       ReportError(SQLResult);
       return null;
   }
E   sqlCommand.Connection = sqlConnection;
   sqlCommand.CommandType = CommandType.Text;
   sqlCommand.CommandText = cmdString;
F   sqlCommand.Parameters.Add("@facultyName", SqlDbType.Text).Value = FacultyName;
   sqlReader = sqlCommand.ExecuteReader();
G   if (sqlReader.HasRows == true)
       FillFacultyReader(ref SQLResult, sqlReader);
H   else
   {
       SQLResult.SQLRequestError = "No matched faculty found";
       ReportError(SQLResult);
   }
I   sqlReader.Close();
   sqlConnection.Close();
   sqlCommand.Dispose();
J   return SQLResult;
}

```

Figure 9.14 Modified codes for the Web method.

- C.** The user-defined method `SQLConn()`, whose detailed codes are shown in Figure 9.13, is called to perform the database connection. This method will return a `Connection` object if the connection is successful. Otherwise, the method will return a `null` object.
- D.** If a `null` is returned from calling the method `SQLConn()`, which means that the database connection has something wrong, a warning message is displayed and another user-defined method `ReportError()`, whose codes are shown in Figure 9.16, is executed to report the encountered error.
- E.** The `Command` object is initialized with the `Connection` object that is obtained from the method `SQLConn()`, `Command` type and `Command` text. Also the input parameter `@facultyName` is assigned with a real input parameter `FacultyName` that is an input parameter to the Web method. One issue is the data type for this parameter. For this application, it does not matter whether a `SqlDbType.Char` or `SqlDbType.Text` is used.
- F.** The `ExecuteReader()` method of the `Command` class is called to invoke the `DataReader` and to perform the data query to our `Faculty` table.
- G.** By checking the `HasRows` property of the `DataReader`, we can determine whether this query is successful or not. If this property is `true`, which means that at least one row has been returned and the query is successful, the user-defined method `FillFacultyReader()` is

```

WebServiceSQLSelect | FillFacultyReader()
protected void FillFacultyReader(ref SQLSelectResult sResult, SqlDataReader sReader)
{
A   if (sReader.Read() == true)
B   {
        sResult.FacultyID = Convert.ToString(sReader["faculty_id"]);
        sResult.FacultyOffice = Convert.ToString(sReader["office"]);
        sResult.FacultyPhone = Convert.ToString(sReader["phone"]);
        sResult.FacultyCollege = Convert.ToString(sReader["college"]);
        sResult.FacultyTitle = Convert.ToString(sReader["title"]);
        sResult.FacultyEmail = Convert.ToString(sReader["email"]);
    }
}

```

Figure 9.15 Codes for the user-defined FillFacultyReader method.

called to assign all queried data columns to the associated member data we created in our child class SQLSelectResult. Two arguments, SQLResult (our returning object) and SqlDataReader (our SqlDataReader object), are passed into that method. The difference between these two arguments is the passing mode; the returning object SQLResult is passed by using a passing-by-reference mode, which means that an address of that object is passed into the method, and all assigned data columns to that object can be brought back to the calling procedure. This is very similar to a returned object from calling a function. However, the SqlDataReader sqlReader is passed by using a passing-by-value mode, which means that only a copy of that object is passed into the method and any modification to that object is temporary.

- H.** If the HasRows property returns false, which means that the data query is failed? An error message is assigned to the member data SQLRequestError defined in our base class SQLSelectBase, and our ReportError() method is called to report this error.
- I.** A cleaning job is performed to release all data objects used in this method.
- J.** The object SQLResult is returned as the query result to our Web Service.

9.3.7.4 Develop Codes for User-Defined Methods

The codes for the user-defined FillFacultyReader() method are shown in Figure 9.15.

Let's take a look at the coding in this subroutine to see how it works.

- A.** The Read() method of the SqlDataReader is executed to read out the queried data rows. In our case, only one row that is matched to the input faculty name is read out and fed into the SqlDataReader object sReader.
- B.** Each data column in the Faculty table can be identified by using its name from the SqlDataReader object sReader and converted to a string using the Convert class method ToString(), and finally assigned to the associated member data in our returning object.

Optionally you can use the GetString() method to retrieve each data column from the SqlDataReader sReader if you like. An index that is matched to the position of each column in the query string cmdString must be used to locate each data item if this method is used.

WebServiceSQLSelect	ReportError()
<pre>protected void ReportError(SQLSelectResult ErrSource) { ErrSource.SQLRequestOK = false; MessageBox.Show(ErrSource.SQLRequestError); }</pre>	

Figure 9.16 Codes for the user-defined ReportError method.

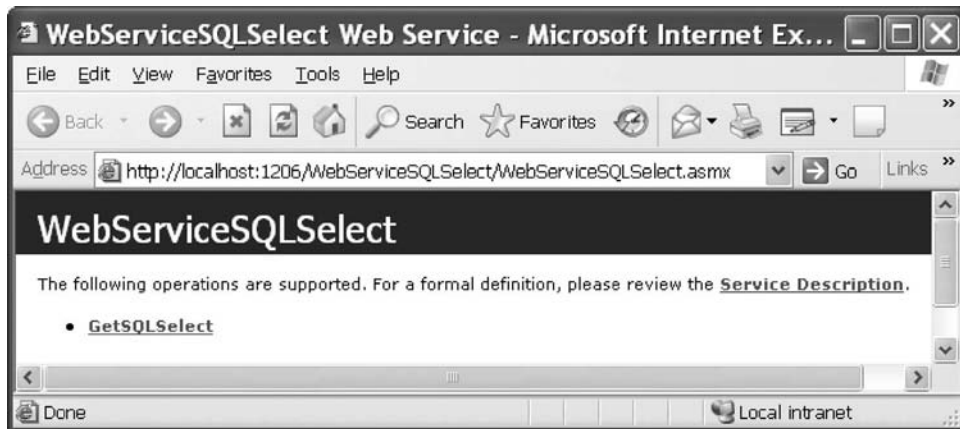


Figure 9.17 Running status of the Web service.

The key point for this method is the passing mode for the first argument. A passing-by-reference mode is used for our returning object, and this is equivalent to return an object from a function.

The codes for another user-defined method, ReportError(), are shown in Figure 9.16. The input parameter to this method is our returning object. A false is assigned to the SQLRequestOK member data, and the error message is assigned to the SQLRequestError string variable defined in our base class SQLSelectBase. Since our returning object is instantiated from our child class SQLSelectResult that is inherited from our base class, our returning object can access and use those member data defined in the base class.

At this point, we finished all coding jobs for our Web Service project. Now let's run our project to test the data query function. Click on the Start Debugging button to run the project and the built-in Web interface is displayed, which is shown in Figure 9.17.

Click on our Web method GetSQLSelect to open the built-in Web interface for our Web method, which is shown in Figure 9.18. Enter the faculty name Ying Bai into the Value box of the FacultyName as our input parameter, and then click on the Invoke button to execute the Web method to trigger the ASP.NET runtime to perform our data query.

The running result is returned and displayed in the XML format, which is shown in Figure 9.19.

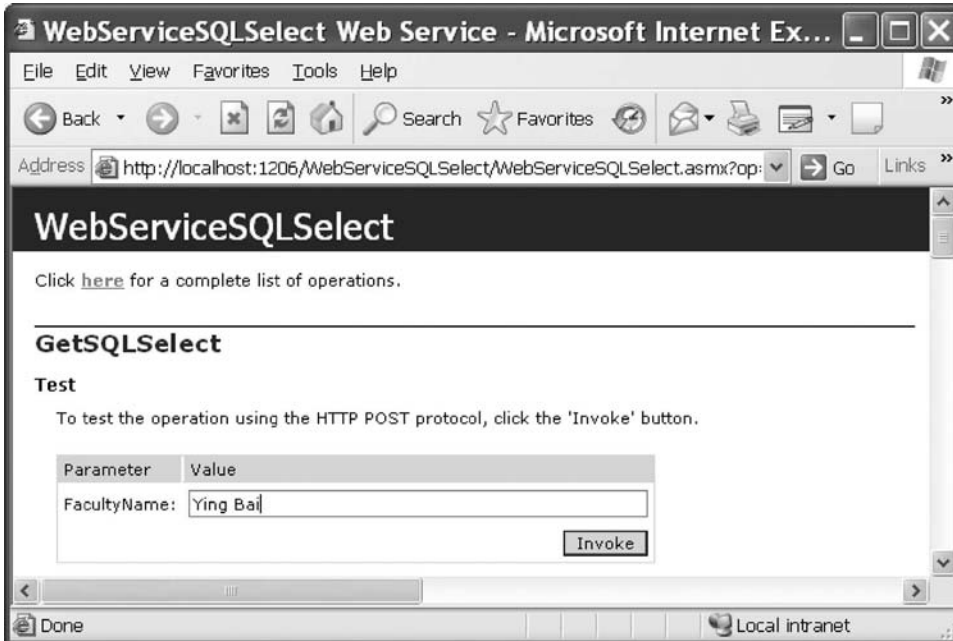


Figure 9.18 Running status of our Web method.

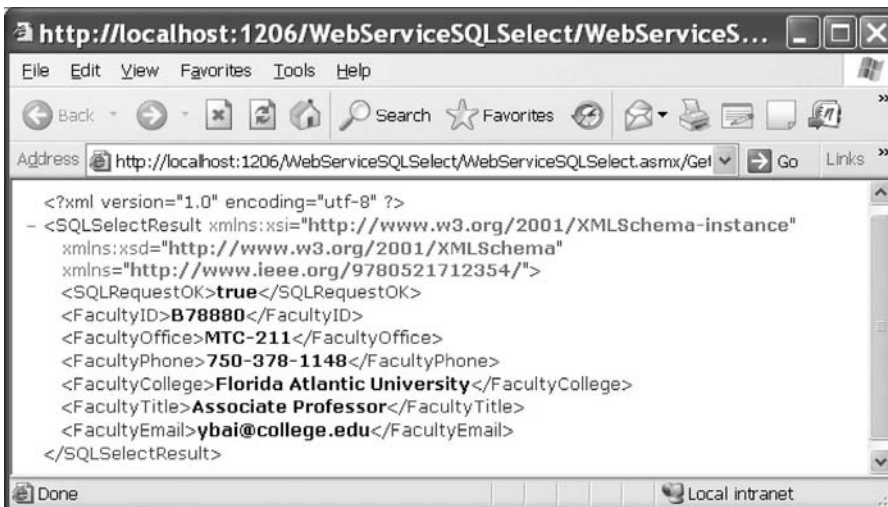


Figure 9.19 Running result of our Web Service project.

Each returned data is enclosed by a pair of XML tags to indicate or markup its facility. For example, the B78880, which is the queried `faculty_id`, is enclosed by the tag `<FacultyID>...</FacultyID>`, and the name of this tag is defined in our child class `SQLSelectResult`. Our first Web Service is very successful.

As we mentioned before, a Web Service did not provide any user interface, and one needs to develop some user interfaces oneself to consume a Web Service if one wants to

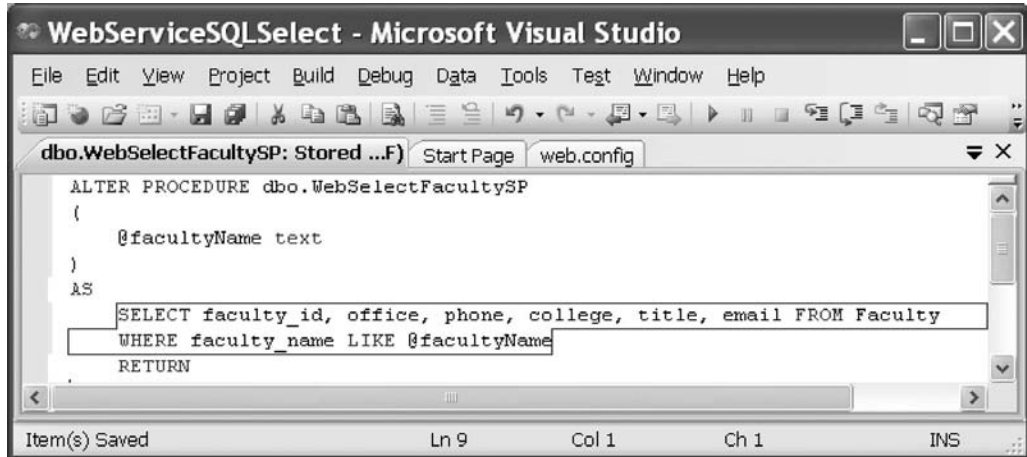


Figure 9.20 Stored procedure.

display those pieces of data obtained from the Web Services. Here, a built-in Web interface is provided by Microsoft to help users display queried information from the Web Services. In real applications, users need to develop user interfaces themselves to perform these data displaying or other graphic-control-related operations.

Click on the Close button located at the upper-right corner for both pages to close our Web Service project.

9.3.8 Develop Stored Procedures to Perform the Data Query

An optional and better way to perform the data query via Web Service is to use the stored procedures. The advantage of using this method is that the coding can be greatly simplified, and most query jobs can be performed in the database side. Therefore the query execution speed can be improved. The query efficiency can also be improved, and the query operations can be integrated into a single group or block of code to strengthen the integrity of the query.

9.3.8.1 Develop Stored Procedure *WebSelectFacultySP*

Now let's first develop our stored procedure in the Server Explorer window in the Visual Studio.NET environment. Open the Visual Studio.NET and open the Server Explorer window. Click on the small plus icon in front of our SQL Server data file CSE_DEPT.mdf to expand our sample database. Then right-click on the Stored Procedures folder and select the item Add New Stored Procedure from the pop-up menu to open a new stored procedure. Enter the codes into this new stored procedure, which are shown in Figure 9.20.

Go to the File|Save StoredProcedure1 menu item to save this new stored procedure with a name of `dbo.WebSelectFacultySP`. We can run this stored procedure in the Visual Studio.NET environment to confirm that it works fine. Right-click on this new created stored procedure from the Server Explorer window and select the Execute item

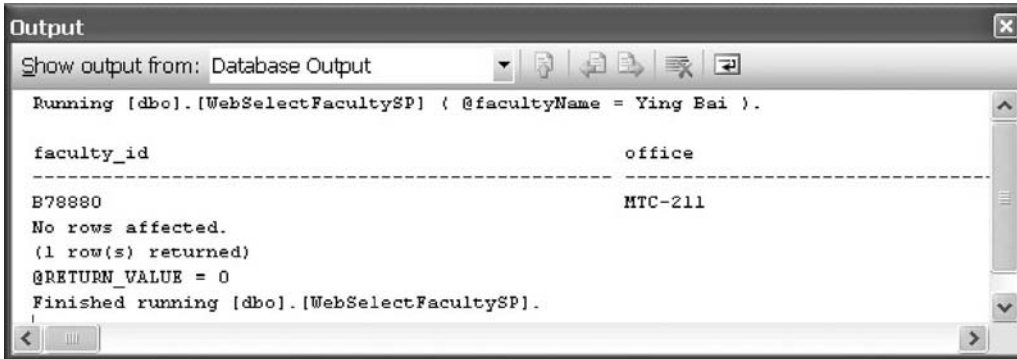


Figure 9.21 Running result of the stored procedure.

from the pop-up menu to open the Run Stored Procedure dialog box. Enter the faculty name Ying Bai to the Value box as the input parameter and click the OK button to run this stored procedure. The running result is displayed in the Output window located at the bottom of this running dialog box, which is shown in Figure 9.21.

All queried six columns, which include the `faculty_id`, `office`, `phone`, `college`, `title`, and `email`, in the Faculty table are displayed in this Output window. You need to move the horizontal bar at the bottom to see all of these six columns. Each column name and its data value are separated with a dash line.

Our stored procedure is successful. Now let's handle the coding in our Web Service project to call this stored procedure to perform this data query.

9.3.8.2 Add Another Web Method to Call the Stored Procedure

To distinguish from the first Web method we developed in the previous section, we had better add another Web method to perform this data query by calling the stored procedure. To do that, highlight and select the whole coding body of our first Web method `GetSQLSelect()`, including both the method header and the code body. Copy this whole body and paste it at the bottom of our code-behind page (must be inside our Web Service class). Perform the following modifications to this copied Web method to make it our second Web method `GetSQLSelectSP()`:

- A. Change the Web method's name by attaching the two letters `SP` to the end of the original Web method's name, and the new Web method's name becomes `GetSQLSelectSP`.
- B. Change the content of the query string `cmdString` to `"dbo.WebSelectFacultySP"`. To call a stored procedure from a Web Service project, the content of the query string must be exactly equal to the name of the stored procedure we developed in the last section. Otherwise a running error may be encountered when the project runs because the project cannot find the target stored procedure.
- C. Change the `CommandType` property of the Command object from the `CommandType.Text` to the `CommandType.StoredProcedure`. This is very important since we need to call a stored procedure to perform the data query. Therefore we must tell the ASP.NET runtime that a stored procedure should be called when the Command object is executed.

```

WebServiceSQLSelect
GetSQLSelectSP()

[WebMethod]
A public SQLSelectResult GetSQLSelectSP(string FacultyName)
{
    SqlConnection sqlConnection = new SqlConnection();
    SQLSelectResult SQLResult = new SQLSelectResult();
    SqlCommand sqlCommand = new SqlCommand();
    SqlDataReader sqlReader;
    B string cmdString = "dbo.WebSelectFacultySP";
    SQLResult.SQLRequestOK = true;
    sqlConnection = SQLConn();

    if (sqlConnection == null)
    {
        SQLResult.SQLRequestError = "Database connection is failed";
        ReportError(SQLResult);
        return null;
    }
    C sqlCommand.Connection = sqlConnection;
    sqlCommand.CommandType = CommandType.StoredProcedure;
    sqlCommand.CommandText = cmdString;
    sqlCommand.Parameters.Add("@facultyName", SqlDbType.Text).Value = FacultyName;
    sqlReader = sqlCommand.ExecuteReader();

    if (sqlReader.HasRows == true)
        FillFacultyReader(ref SQLResult, sqlReader);
    else
    {
        SQLResult.SQLRequestError = "No matched faculty found";
        ReportError(SQLResult);
    }

    sqlReader.Close();
    sqlConnection.Close();
    sqlCommand.Dispose();
    return SQLResult;
}

```

Figure 9.22 Modified Web method—GetSQLSelectSP.

Your finished modified Web method `GetSQLSelectSP()` should match the one shown in Figure 9.22. All modified parts have been highlighted in bold shown in steps **A**, **B**, and **C** in Figure 9.22.

Now we can run the project to test this new Web method. Click on the second Web method `GetSQLSelectSP` as the project runs, and then click on the `Invoke` button to run it. The same running result as we got from the last project can be obtained.

You can see how easy it is to develop codes to perform the data query by calling a stored procedure in the Web Service project. Next we want to discuss how to use a `DataSet` as a returning object to contain all pieces of information we need from executing a Web Service project.

9.3.9 Use DataSet as Returning Object for Web Method

The advantage of using a `DataSet` as the returning object for a Web method is that we do not need to create any application class to instantiate a returning object. Another advantage is that a `DataSet` can contain multiple records coming from the different tables,

```

[WebMethod]
public DataSet GetSQLSelectDataSet(string FacultyName)
{
    SqlConnection sqlConnection = new SqlConnection();
    SQLSelectResult SQLResult = new SQLSelectResult();
    SqlCommand sqlCommand = new SqlCommand();
    SqlDataAdapter FacultyAdapter = new SqlDataAdapter();
    DataSet dsFaculty = new DataSet();
    int intResult = 0;

    string cmdString = "SELECT faculty_id, office, phone, college, title, email FROM Faculty " +
        "WHERE faculty_name LIKE @facultyName";

    SQLResult.SQLRequestOK = true;
    sqlConnection = SQLConn();
    if (sqlConnection == null)
    {
        SQLResult.SQLRequestError = "Database connection is failed";
        ReportError(SQLResult);
        return null;
    }
    sqlCommand.Connection = sqlConnection;
    sqlCommand.CommandType = CommandType.Text;
    sqlCommand.CommandText = cmdString;
    sqlCommand.Parameters.Add("@facultyName", SqlDbType.Text).Value = FacultyName;
    FacultyAdapter.SelectCommand = sqlCommand;
    intResult = FacultyAdapter.Fill(dsFaculty, "Faculty");
    if (intResult == 0)
    {
        SQLResult.SQLRequestError = "No matched faculty found";
        ReportError(SQLResult);
    }
    sqlConnection.Close();
    sqlCommand.Dispose();
    FacultyAdapter.Dispose();
    return dsFaculty;
}

```

Figure 9.23 Modified Web method—GetSQLSelectDataSet.

and we do not need to create multiple member data in our application class to hold those data items. Finally the size of our coding body will be greatly reduced when a DataSet is used, especially for a large block of data that is queried via a Web Service project.

To distinguish the Web methods we developed before, we can add another new Web method named GetSQLSelectDataSet into our Web Service project. To do that, open our code-behind page if it is not opened, highlight and select the whole coding body of our first Web method GetSQLSelect, including the method header and coding body. Then copy and paste it at the bottom of our page (must be inside our Web service class). Perform the modifications shown in Figure 9.23 to this copied Web method to make it our third Web method. The modified parts have been highlighted in bold.

Let's take a closer look at this modified Web method to see how it works.

- A.** The Web method's name is modified by attaching "DataSet" to the end of the original method name. Also the nominal name of the returning object is changed to DataSet, which means that this Web method will return a DataSet.
- B.** Two new data objects, FacultyAdapter and dsFaculty, are created, and these two objects work as a DataAdapter and a DataSet, respectively. A local integer variable intResult is

also created, and it will be used to hold the returning value from calling the Fill() method of the DataAdapter to perform the data query later.

- C. The initialized Command object is assigned to the SelectCommand property of the DataAdapter class. This Command object will be executed when the Fill() method is called to perform the data query, exactly to fill the faculty table in the DataSet dsFaculty.
- D. The Fill() method of the DataAdapter class is executed to fill the Faculty table in our DataSet. This method will return an integer value to indicate whether this calling is successful or not, and this returned value is stored into the local integer variable intResult that will be checked later.
- E. If the returned value is zero, it means that no row has been retrieved from the Faculty table in our sample database and no row has been filled into our Faculty table in our DataSet dsFaculty. Therefore this data query has failed. An error message will be sent to our member data in our base class and that error will be reported by using the user-defined ReportError() method.
- F. Otherwise if the returned value is nonzero, which means that at least one row has been retrieved and filled into the Faculty table in our DataSet, a cleaning job is performed to release all objects used for this Web method. Typically this returned value is equal to the number of rows that have been successfully retrieved from the Faculty table in our database and filled into the Faculty table in our DataSet. In our application, this value should be equal to 1 since only one record is returned and filled.
- G. Finally the filled DataSet dsFaculty, that is, the filled Faculty table in this DataSet, is returned to the Web Service.

Now we can run the project to test this returned DataSet function. Click on the Start Debugging button to run the project. Now we have three Web methods available to this Web Service, which are shown in Figure 9.24.

Click on the second method GetSQLSelectDataSet from the built-in Web interface window and enter the faculty name Ying Bai into the Value box as our desired faculty,

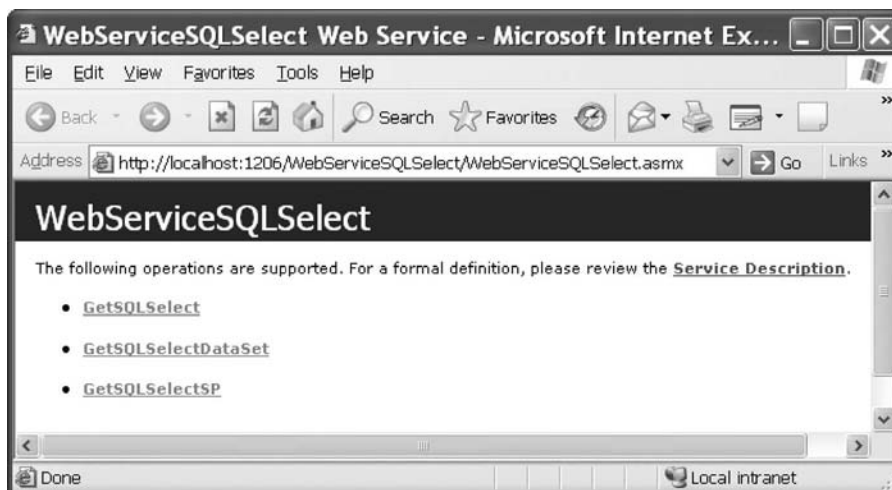


Figure 9.24 Three Web methods built-in Web interface window.

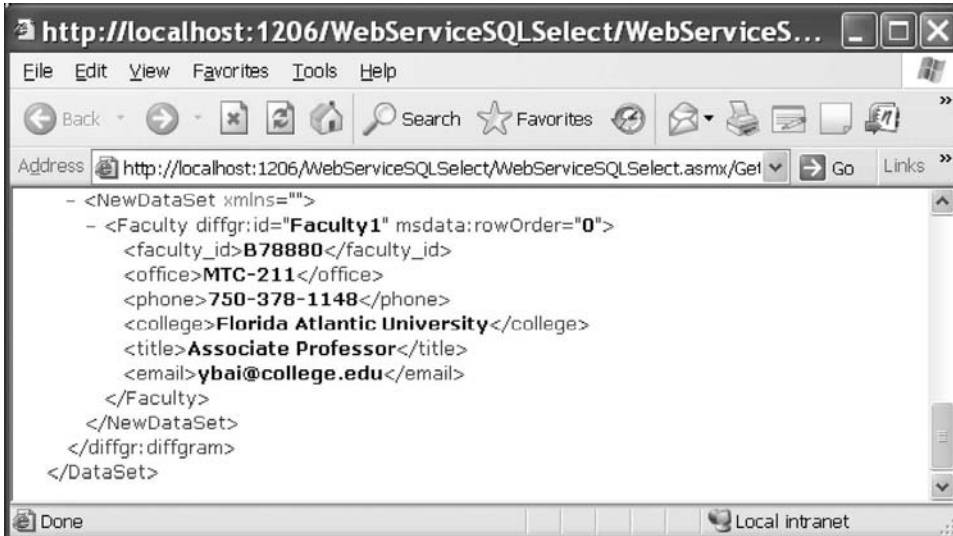


Figure 9.25 Running result of the Web method—GetSQLSelectDataSet.

and then click on the Invoke button to call this Web method to perform the data query. The following running result is returned, which is shown in Figure 9.25.

A new DataSet is created since we used a nontyped DataSet in this application, and all six pieces of faculty information related to the desired faculty member Ying Bai are retrieved and filled into the Faculty table in our DataSet. Also these pieces of data are returned to our Web Service project and displayed in the built-in Web interface window, as shown in Figure 9.25.

At this point, we have finished all developing jobs in our Web Service project in the server side. A complete Web Service project WebServiceSQLSelect that contains three Web methods can be found at the folder DBProjects\Chapter 9 located at the ftp://ftp.wiley.com/public/sci_tech_med/practical_database site.

Next we want to develop some professional Windows-based or Web-based applications with beautiful graphic user interfaces to use or to consume the Web Service application we developed in the previous sections. Those Windows-based or Web-based applications can be considered as Web service clients.

9.3.10 Build Windows-Based Web Service Clients to Use the Web Services

To use a Web Service, first we need to create a Web Service proxy class in our Windows-based or Web-based applications. Then we can create a new instance of the Web Service proxy class and execute the desired Web methods located in that Web Service class. The process of creating a Web Service proxy class is equivalent to adding a Web reference to our Windows-based or Web-based applications.

9.3.10.1 Create a Web Service Proxy Class

Basically, adding a Web reference to our Windows-based or Web-based applications is to execute a searching process. During this process, Visual Studio.NET 2008 will try to find all Web Services available to our applications. The following operations will be performed by Visual Studio.NET 2008 during this process:

1. When looking for Web Services from local computers, Visual Studio.NET 2008 will check all files that include a Web Service main page with a `.asmx` extension, a WSDL file with a `.wsdl` extension, or a Discovery file with a `.disco` extension.
2. When searching for Web Services from the Internet, Visual Studio.NET 2008 will try to find a UDDI file that contains all registered Web Services with their associated Discovery documents.
3. When all available Web Services are found, either from a local computer or from the Internet, you can select your desired Web Services from them by adding them into the Web client project as Web references. Also you can open each of them to take a look at the detail description for each Web Service and its Web methods. Once you selected the desired Web Services, you can modify the names of the selected Web Services as you want. The point is that even the name of the Web Service is changed in the Web client side. The ASP.NET runtime can remember and still use the original name of that Web Service as it is consumed.
4. As those Web Services have been referenced to the client project, a group of necessary files or documents are also created by Visual Studio.NET 2008. These files include:
 - a. A Discovery Map file that provides the necessary SOAP interfaces for communications between the client project and the Web Services.
 - b. A Discovery file that contains all available XML Web Services on a Web server, and these Web Services are obtained through a process called XML Web Services Discovery.
 - c. A WSDL file that provides a detailed description and definition about those Web Services in an abstract manner.

To add a Web reference to our client project, we need first to create a client project. Now let's create a Windows-based application to consume the Web Service we developed in the previous section. First let's add a Web reference to our new project.

Open Visual Studio.NET 2008 and create a new Windows-based Visual C# project, and name this project WinClientSQLSelect. Perform the following operations to open the Add Web Reference dialog box:

1. Right-click on our new project WinClientSQLSelect from the Solution Explorer window, and select the item Add Service Reference from the pop-up menu to open the Add Service Reference dialog box.
2. Click on the Advanced button to open the Service Reference Settings dialog box.
3. Click on the Add Web Reference button to open the Add Web Reference dialog box, which is shown in Figure 9.26.

There are two ways we can select the desired Web Service and add it as a reference to our client project: One way is to use the Browser provided by the Visual Studio .NET 2008 to find the desired Web Service. Another way is to copy and paste the desired Web Service's URL to the URL box located in this Add Web Reference dialog box.

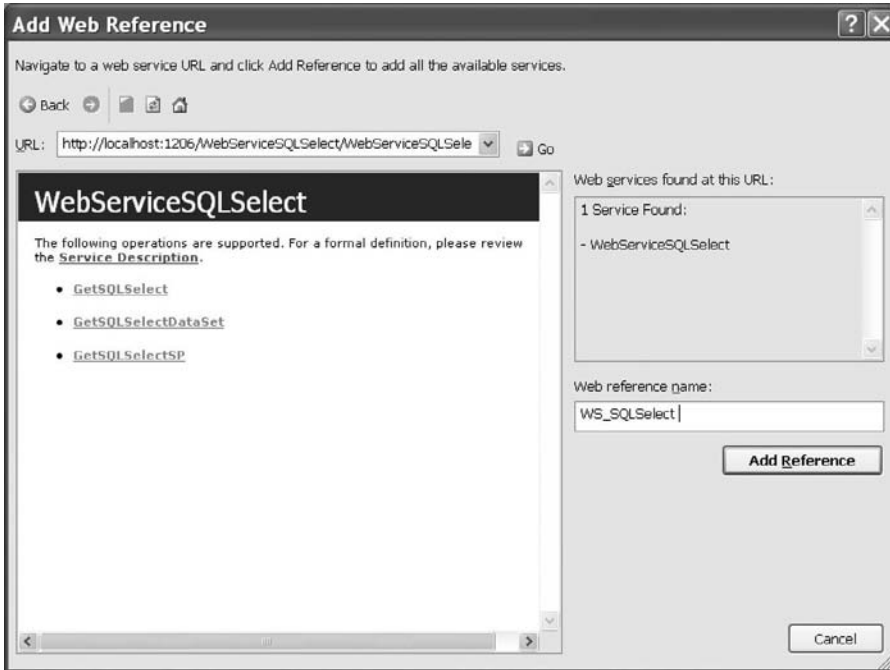


Figure 9.26 Add Web Reference dialog box.

The second way needs you first to run the Web Service, and then copy its URL and paste it to the URL box in this dialog if you did not deploy that Web Service to IIS. If you did deploy that Web Service, you can directly type that URL into the URL box in this dialog.

Because we developed our Web Service using the File System on our local computer, and we have not deployed our Web Service to IIS, we must use the second way to find our Web Service. Open our Web Service project and click on the Start Debugging button to run it. Copy the URL from the Address bar and then switch back to our client project WinClientSQLSelect, and paste that URL into the URL box in the Add Web Reference dialog. Click on the Go button to allow the Visual Studio.NET 2008 to begin to search it. When the Web Service is found, the name of our Web Service is displayed in the right pane, which is shown in Figure 9.26.

Alternately you can change the name for this Web reference from localhost to any meaningful name such as WS_SQLSelect in our case. Click on the Add Reference button to add this Web Service as a reference to our new client project. Click on the Close button from our Web Service built-in Web interface window to terminate our Web Service project.

Immediately you can find that the Web Service WS_SQLSelect, which is under the folder Web References, has been added into the Solution Explorer window in our project. This reference is the so-called Web Service proxy class.

Next let's develop our graphic user interface by adding useful controls to interface to our Web Service and to display the queried information.

9.3.10.2 Develop Graphic User Interface for Windows-Based Client Project

Perform the following modifications to our new project:

1. Rename the Form File object from the default name `Form1.cs` to our desired name `WinClientForm.cs`.
2. Rename the Window Form object from the default name `Form1` to our desired name `FacultyForm` by modifying the `Name` property of the form window.
3. Rename the form title from the default title `Form1` to `CSE DEPT Faculty Form` by modifying the `Text` property of the form.

To save time and space, we can use a graphic user interface located in the project `SQLUpdateDeleteRTOject` we developed in Chapter 7. Open that project from the folder `DBProjects\Chapter 7` located at the accompanying ftp site (see Chapter 1). Then open the `Faculty Form` window and select all controls from that form by going to the menu item `Edit>Select All`, and then go to the `Edit|Copy` menu item to copy all controls selected from this form window.

Return to our new Windows-based Web Service client project `WinClientSQLSelect`, open our `Faculty Form` window, and paste those controls we copied from the `Faculty Form` in the project `SQLUpdateDeleteRTOject` to this form. The only modification to these controls is to change the name of the `Faculty Name` textbox `txtName` to `txtID` and the associated label to `Faculty ID` inside the `Faculty Information` group box since we need this piece of data. Your finished graphics user interface is shown in Figure 9.27.

The purpose of the combobox control method is used to select three different Web methods developed in our Web Service project to get our desired information:

1. Method that uses an object to return our queried information
2. Method that uses a stored procedure to return our queried information
3. Method that uses a `DataSet` to return our queried information

The `Faculty Name` combobox control is used to select the desired faculty as the input parameter to the Web method to pick up our desired faculty information.

Figure 9.27 Graphics user interface.

In this application, only the Select and Back buttons are used. The function of this project is: When the project runs, as the desired method and the faculty name have been selected from the associated controls, the Select button will be clicked on by the user. Our client project will connect to our Web Service based on the Web reference we provided, and call the selected method based on the method chosen from the Method combobox control to perform the data query to retrieve the desired faculty information from our sample database, and display it in this graphic user interface.

Now let's take care of the coding for this project to connect to our Web Service using the Web reference we developed in the last section.

9.3.10.3 Develop Code to Use the Web Service

The coding job can be divided into four parts:

1. Coding for the constructor of the FacultyForm class to initialize the Method combobox control and the Faculty Name combobox control: The first initialization will set up three Web methods that can be selected by the user to perform the data query from the Web Service. The second one is to set up a default list of faculty members that can be selected by the user to perform the associated faculty information query.
2. Coding for the Back button's Click method to terminate the project.
3. Coding for the Select button's Click method.
4. Coding for other user-defined methods, such as ShowFaculty(), ProcessObject(), FillFacultyObject(), and FillFacultyDataSet().

The main coding job is performed inside the Select button's Click method. As we discussed, as this button is clicked on by the user, a connection to our Web Service needs to be established using the Web reference we set up in the previous section. Therefore we need first to create an object based on that Web reference or instantiate that Web Service to get an instance, then we can access the different Web methods to perform our data query via that instance. This process is called to instantiate the proxy class and invoke the Web methods. The protocol to instantiate a proxy class is:

```
WebReference.WebService newInstance = new WebReference.WebService();
```

After this new instance is created, then a connection between our client project and our Web service can be set up by using this instance. The pseudocodes for this procedure are listed below:

- A. A new Web Service instance wsSQLSelect is created using the protocol given above.
- B. A new object wsSQLResult is also created, and it can be used as a mapping of the real object SQLSelectResult developed in the Web Service. We can easily access the Web method to perform our data query and pick up the result from that returning object by assigning it to the mapping object.
- C. A new DataSet object is created, and it is used to call the Web method that returns a DataSet.
- D. Based on the method selected by the user from the Method combobox control, different Web methods can be called to perform the data query.

```

public FacultyForm()
{
    InitializeComponent();
    ComboName.Items.Add("Ying Bai");
    ComboName.Items.Add("Satish Bhalla");
    ComboName.Items.Add("Black Anderson");
    ComboName.Items.Add("Steve Johnson");
    ComboName.Items.Add("Jenney King");
    ComboName.Items.Add("Alice Brown");
    ComboName.Items.Add("Debby Angles");
    ComboName.Items.Add("Jeff Henry");
    ComboName.SelectedIndex = 0;
    ComboMethod.Items.Add("Object Method");
    ComboMethod.Items.Add("Stored Procedure Method");
    ComboMethod.Items.Add("DataSet Method");
    ComboMethod.SelectedIndex = 0;
}

```

Figure 9.28 Coding for the constructor of the FacultyForm class.

- E. The returned data that are stored in the real object are assigned to our mapping object, and each piece of data can be retrieved from this object and displayed in our graphic user interface.
- F. If a DataSet method is used, the returned DataSet object is assigned to our mapping DataSet, and the method FillFacultyDataSet() is called to fill the textboxes in the client form with the information picked up from the DataSet.

9.3.10.3.1 Develop Codes for the Constructor of FacultyForm Class Now let's begin to develop codes for the constructor of the FacultyForm class to complete the initialization jobs listed in step 1 above. Open the constructor of the FacultyForm class and enter the codes shown in Figure 9.28 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. Eight default faculty members are added into the combobox control ComboName using the Add() method, and this will allow users to select one desired faculty from this combobox control to perform the data query as the project runs. The default-selected faculty is the first one by setting the SelectedIndex property to zero.
- B. Three Web methods are also added into another combobox control ComboMethod to allow users to select one of them to perform the associated data query via our Web service. The default method is selected as the first one.

9.3.10.3.2 Develop Codes for Back Button Click Method This coding is very easy. Just open the Back button's Click method by double-clicking on it from the Faculty Form window and enter the following one-line code into this method:

```
Application.Exit();
```

The purpose of this coding line is to terminate our project as this button is clicked on by the user.

```

WinClientSQLSelect.FacultyForm  cmdSelect_Click()
private void cmdSelect_Click(object sender, EventArgs e)
{
A   WS_SQLSelect.WebServiceSQLSelect wsSQLSelect = new WS_SQLSelect.WebServiceSQLSelect();
   WS_SQLSelect.SQLSelectResult wsSQLResult = new WS_SQLSelect.SQLSelectResult();
   DataSet wsDataSet = new DataSet();

B   if (ComboMethod.Text == "Object Method")
   {
C     try{wsSQLResult = wsSQLSelect.GetSQLSelect(ComboName.Text);}
D     catch (Exception err) {MessageBox.Show("Web service is wrong: " + err.Message);}
   }
E   else if (ComboMethod.Text == "Stored Procedure Method")
   {
F     try { wsSQLResult = wsSQLSelect.GetSQLSelectSP(ComboName.Text); }
G     catch (Exception err) { MessageBox.Show("Web service is wrong: " + err.Message); }
   }
H   else
   {
I     try { wsDataSet = wsSQLSelect.GetSQLSelectDataSet(ComboName.Text); }
J     catch (Exception err) { MessageBox.Show("Web service is wrong: " + err.Message); }
   }
}

```

Figure 9.29 Coding for the Select button click method.

9.3.10.3.3 Develop Codes for Select Button Click Method Open the Select button's Click method and enter the codes shown in Figure 9.29 into this method.

Let's take a closer look at this piece of code to see how it works.

- A.** Some data objects are created at the beginning of this method, which include a new Web Service instance `wsSQLSelect` that is created using the protocol given above, a new object `wsSQLResult` that can be used as a mapping of the real object `SQLSelectResult` developed in our Web Service. Also we can easily access the Web method to perform our data query and pick up the result from that returning object by assigning it to this mapping object later, and a new `DataSet` object is used to call the Web method that returns a `DataSet`.
- B.** If the user selected the **Object Method** from the `ComboMethod` control, a `try ... catch` block is used to call the associated Web method `GetSQLSelect()`, which is developed in our Web Service, through the instantiated reference class to perform the data query. The selected faculty that is located in the `Text` property of the `ComboName` combo control is passed as a parameter for this calling.
- C.** The `catch` statement is used to collect any possible exceptions if any error occurred for this calling, and the error message is displayed using a message box.
- D.** If no exception occurred, the user-defined `ProcessObject()` method is executed to pick up all pieces of retrieved information from the returned object and display them in this form window.
- E.** If the user selected the **Stored Procedure Method**, the associated Web method `GetSQLSelectSP()`, which is developed in our Web Service, is called via the instance of the Web reference class to perform the data query.

- F. The `catch` statement is used to collect any possible exceptions if any error occurred for this calling, and the error message is displayed using a message box.
- G. Similarly the user-defined `ProcessObject()` method is executed to pick up all pieces of retrieved data from the returned object and display them in this form window.
- H. If users selected the `DataSet Method`, the Web method `GetSQLSelectDataSet()` is called through the instance of the Web reference class, and the method returns a `DataSet` that contains our desired faculty information.
- I. The `catch` statement is used to collect any possible exceptions if any error occurred for this calling, and the error message is displayed using a message box.
- J. Another user-defined method, `FillFacultyDataSet()`, is called to pick up all pieces of retrieved data from the returned `DataSet` and display them in this form.

9.3.10.3.4 Develop Codes for User-Defined Methods The codes for the user-defined methods `ProcessObject()` and `FillFacultyObject()` are shown in Figure 9.30.

Both methods use our child class `SQLSelectResult` as the data type of the passed argument since our returned object is an instance of this class. The function of this piece of code is:

- A. If the member data `SQLRequestOK` that is stored in the instance of our child class or returned object is set to `true`, which means that our Web method is executed successfully, the user-defined `FillFacultyObject()` method is called and executed with the returned object that contains our requested faculty information as a reference argument to pick up each piece of data from that returned object and display it in this form window.
- B. Otherwise some exceptions have occurred and a warning message is displayed with a message box.
- C. As the user-defined method `FillFacultyObject()` is called, all six pieces of faculty data stored in the returned object are picked up and assigned to the associated textboxes to be displayed in this form.

The screenshot shows a code editor window titled "WinClientSQLSelect.FacultyForm" with a dropdown menu showing "ProcessObject()". The code is as follows:

```

private void ProcessObject(ref WS_SQLSelect.SQLSelectResult wsResult)
{
    if (wsResult.SQLRequestOK == true)
        FillFacultyObject(ref wsResult);
    else
        MessageBox.Show("Faculty information cannot be retrieved: " + wsResult.SQLRequestError);
}

private void FillFacultyObject(ref WS_SQLSelect.SQLSelectResult sqlResult)
{
    txtID.Text = sqlResult.FacultyID;
    txtOffice.Text = sqlResult.FacultyOffice;
    txtPhone.Text = sqlResult.FacultyPhone;
    txtCollege.Text = sqlResult.FacultyCollege;
    txtTitle.Text = sqlResult.FacultyTitle;
    txtEmail.Text = sqlResult.FacultyEmail;
}

```

Labels A, B, and C are placed to the left of the code to indicate specific lines: A is next to the `if` statement, B is next to the `else` block, and C is next to the `FillFacultyObject` method signature.

Figure 9.30 Codes for two user-defined methods.

```

WinClientSQLSelect.FacultyForm  FillFacultyDataSet()
private void FillFacultyDataSet(ref DataSet ds)
{
A   DataTable FacultyTable = new DataTable();
   DataRow FacultyRow;
B   FacultyTable = ds.Tables["Faculty"];
C   FacultyRow = FacultyTable.Rows[0];           //only one row in the Faculty table
D   txtID.Text = FacultyRow["faculty_id"].ToString();
   txtOffice.Text = FacultyRow["office"].ToString();
   txtPhone.Text = FacultyRow["phone"].ToString();
   txtCollege.Text = FacultyRow["college"].ToString();
   txtTitle.Text = FacultyRow["title"].ToString();
   txtEmail.Text = FacultyRow["email"].ToString();
}

```

Figure 9.31 Codes for the FillFacultyDataSet method.

The detailed codes for the user-defined FillFacultyDataSet() method are shown in Figure 9.31. The argument passed into this method is an instance of DataSet we created in the Select button's Click method.

Let's take a look at this piece of code to see how it works.

- A.** Two data objects, FacultyTable, which is a new object of the DataTable class, and FacultyRow, which is a new instance of the DataRow class, are created first since we need to use these two objects to access the DataSet to pick up all requested faculty information later.
- B.** The returned Faculty table that is embedded in our returned DataSet is assigned to our new created object FacultyTable. Because the DataSet we created in the Select button Click method is an untyped DataSet, the table name must be clearly indicated with a string "Faculty". For typed DataSet, you can directly use the table name to access the desired table without needing any string.
- C.** Since we only request one record or a unique row from the Faculty table, the returned Faculty table contains only one row information, which is located at the top row with an index of zero. This record is assigned to our FacultyRow object we created above.
- D.** We can access each column from the returned data row using the column name represented by a string with a class method ToString(). As we mentioned, the DataSet we are using is an untyped DataSet. Therefore the column name must be indicated with a string and the value of that column must be converted to a string by using the ToString() method. If a typed DataSet is used, you can directly use the column name (no string to cover it) to access that column without needing to use the ToString() method. Each piece of information returned is assigned to the associated textbox, and it will be displayed there.

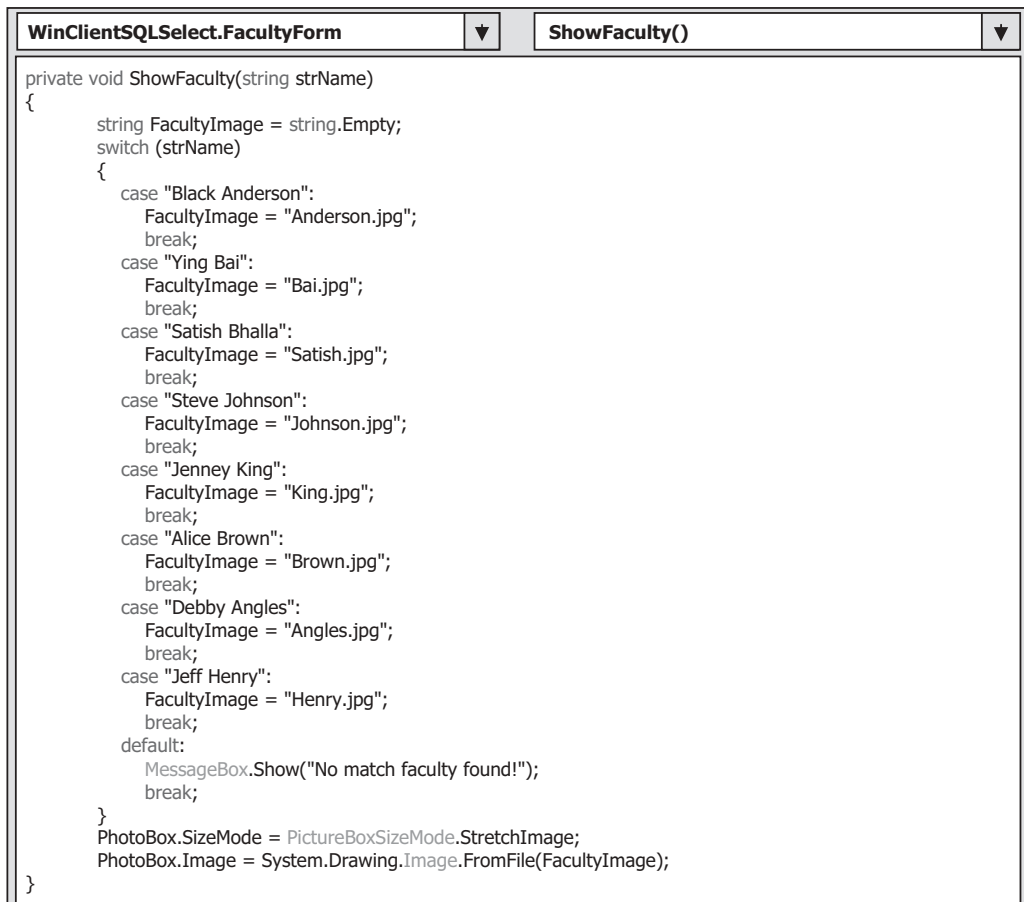
At this point we have almost finished coding for this Windows-based Web Service client project. We have one more step to go to complete this client project, which is to add a user-defined method ShowFaculty() to display the requested faculty photo in the image box in this form.

9.3.10.3.5 Develop a ShowFaculty Method to Display the Faculty Image All default faculty images can be found in the folder Image located at the accompanying ftp site (see Chapter 1). All faculties and students's photo files used for our sample database

in this book are stored in that folder. To display the requested faculty image in a form, the following operations need to be performed:

1. Copy all default faculty photos from the folder `Image` located at the accompanying ftp site (see Chapter 1), and paste them into our current client project folder, that is, into the `Debug` folder that is under our project's `bin` folder. For example, in this application, it is the folder `C:\Chapter 9\WinClientSQLSelect\bin\Debug`.
2. Develop the codes to perform this faculty image displaying.

Now let's develop the codes for this method. Open the code window from our client project, and type the codes shown in Figure 9.32 into this code window to create our method `ShowFaculty()`. The codes for this method are straightforward with no trick. A `switch-case` structure is used to pick up each associated or matched faculty image file based on the input faculty name. The selected faculty image file is passed as an argument to the system method `Image.FromFile()` and then is displayed in the `PhotoBox` control in this form window. A warning message will be displayed if no matched faculty image can be found.



```

WinClientSQLSelect.FacultyForm ShowFaculty()
private void ShowFaculty(string strName)
{
    string FacultyImage = string.Empty;
    switch (strName)
    {
        case "Black Anderson":
            FacultyImage = "Anderson.jpg";
            break;
        case "Ying Bai":
            FacultyImage = "Bai.jpg";
            break;
        case "Satish Bhalla":
            FacultyImage = "Satish.jpg";
            break;
        case "Steve Johnson":
            FacultyImage = "Johnson.jpg";
            break;
        case "Jenney King":
            FacultyImage = "King.jpg";
            break;
        case "Alice Brown":
            FacultyImage = "Brown.jpg";
            break;
        case "Debby Angles":
            FacultyImage = "Angles.jpg";
            break;
        case "Jeff Henry":
            FacultyImage = "Henry.jpg";
            break;
        default:
            MessageBox.Show("No match faculty found!");
            break;
    }
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage;
    PhotoBox.Image = System.Drawing.Image.FromFile(FacultyImage);
}

```

Figure 9.32 Codes for the `ShowFaculty` method.

To call this method, display the selected faculty image, add one more instruction, which is shown below to the Select button's Click method. Add this instruction as the first code line under the data objects declaration part:

```
ShowFaculty(ComboName.Text);
```

Now we can start to run this client project to interface with our Web Service and furthermore to access and use the Web methods to perform our data query.

Wait a moment! There is one important issue you need to note before you can run this project: You must run our Web Service project `WebServiceSQLSelect` **first** to make our Web Service available to all clients, and then you can run our client project to access and interface to our Web Service to perform the data query. Otherwise, you may encounter some running exceptions such as the Web Service or a remote computer cannot be found when your client project runs.

Once our Web Service project runs, you can stop it and then access it using our client project without any problem at all. But one point you need to remember is that our Web Service project must be kept in the open status (even if it is terminated) in order to allow our client project to access it. An exception will be encountered if you closed our Web Service when you run this client project to try to access it.

Make sure that our Web Service has been run at one time and it is in the open status, which can be identified by a small Web Service running a white icon in the task bar on the bottom of your screen. Start our client project by clicking on the Start Debugging button from the project `WinClientSQLSelect`. The running status is shown in Figure 9.33.

Keep the default Web method and the faculty name `Ying Bai` selected and click on the Select button to call the associated Web method to retrieve the desired faculty information. The returned faculty information is displayed in the associated textboxes with the faculty photo, which is shown in Figure 9.33.

You can try to select two other Web methods such as the Stored Procedure or the DataSet method and other faculty members to perform this data query. The running result confirms that both our Web Service and our Windows-based Web Service client projects are very successful. Click on the Back button to terminate our project.

The screenshot shows a Windows application window titled "CSE DEPT Faculty Form". It features a "Method" dropdown menu set to "DataSet Method" and a "Faculty Name" dropdown menu set to "Ying Bai". Below the dropdowns is a small photograph of a man. To the right of the photo is a "Faculty Information" section with several text boxes containing the following data: Faculty ID: 878880, Title: Associate Professor, Office: MTC-211, Phone: 750-378-1148, College: Florida Atlantic University, and Email: ybai@college.edu. At the bottom of the form are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 9.33 Running status of our client project.

A complete Windows-based Web Service client project `WinClientSQLSelect` can be found at the folder `DBProjects\Chapter 9` located at the accompanying ftp site (see Chapter 1). You need to load both this client project and our Web Service project from that folder and install them on your computer if you want to run and test this client project. Also you must run our Web Service project once to make sure that our Web Service is ready to be consumed by that client project.

Next we want to develop a Web-based project to use our Web Service by retrieving the desired faculty information.

9.3.11 Build Web-Based Web Service Clients to Use the Web Service

Developing a Web-based client application to use a Web Service is very similar to developing a Windows-based client project to reference and access a Web Service as we did in the last section. As long as a Web Service is referenced by the Web-based client project, one can access and call any Web method developed in that Web Service to perform the desired data queries via the Web-based client project without any problem. Visual Studio .NET will create the same document files such as the Discovery Map file, the WDSL file, and the DISCO file for the client project even if this Web Service is used by a Windows-based or a Web-based client application.

To save time and space, we can modify an existing ASP.NET Web application `SQLWebSelect` we developed in Chapter 8 to make it our new Web-based Web Service client project named `WebClientSQLSelect`; that is, we can copy and rename that entire project as our new Web-based client project. However, here we prefer to create a new ASP.NET website project and only copy and modify the Faculty page.

This section can be developed in the following sequences:

1. Create a new ASP.NET website project `WebClientSQLSelect` and add an existing Web page `Faculty.aspx` from the project `SQLWebSelect` into our new project.
2. Add a Web Service reference to our new project and modify the Web form window of the `Faculty.aspx` to meet our data query requirements.
3. Modify the codes in the related methods of the `Faculty.aspx.cs` file to call the associated Web method to perform our data query. The code modifications include the following sections:
 - a. Modify the codes in the `Page_Load()` method of the `Faculty.aspx.cs` file.
 - b. Modify the codes inside the `Select` button's `Click` method.
 - c. Add three user-defined methods: `ProcessObject()`, `FillFacultyObject()`, and `FillFacultyDataSet()`. These three methods are basically identical with those we developed in the last Windows-based Web service client project `WinClientSQLSelect`, and one can copy and paste them into our new project. The only modification is for the `ProcessObject()` method.
 - d. Modify the codes in the `Back` button's `Click` method.
 - e. Remove two unused user-defined methods, `FillFacultyReader()` and `MapFacultyTable()`, since we will not use them in this application.

Now let's start with the first step listed above.

9.3.11.1 Create a New Website Project and Add an Existing Web Page

Open Visual Studio.NET and go to the File|New Web Site menu item to create a new website project. Enter C:\Chapter 9\WebClientSQLSelect into the name box that is next to the Location box, and click on OK to create this new project.

On the opened new project window, right-click on our new project icon WebClientSQLSelect from the Solution Explorer window, and select the item Add Existing Item from the pop-up menu to open the Add Existing Item dialog box. Browse to our Web project SQLWebSelect, select it, and then click on the Add button to open all existing items for this website project. Select both items, Faculty.aspx and Faculty.aspx.cs, from the list and click on the Add button to add these two items into our new website project.

One issue we need to emphasize is that we had better add all faculty photo files into our new project before we can continue to the next step. In this way, the selected faculty photo can be displayed as that faculty's information is queried. This step is highly recommended since we need those faculty photo files later when we display each of them for each selected faculty as we perform the data query.

Two ways can be used to complete this photo file's adding function. First, you can copy and paste all faculties and students photo files from the folder Image located at the accompanying ftp site (see Chapter 1). The second way is that you can do this copy-paste of photo files from the project SQLWebSelect. To do this, right-click on our new project icon WebClientSQLSelect from the Solution Explorer window, and select the Add Existing Item from the pop-up menu. Browse to our project SQLWebSelect, select it, and then click on the Add button to open all existing items for this website project. Select all files that have an extension .jpg and click on the Add button to add them into our new project.

9.3.11.2 Add a Web Service Reference and Modify the Web Form Window

Just as we did in the last project, to add a Web reference of our Web Service to this new website project, right-click on our new project icon from the Solution Explorer window and select the item Add Web Reference from the pop-up menu to open the Add Web Reference dialog box. Now open our Web Service project WebServiceSQLSelect and click on the Start Debugging button to run it. As the project runs, copy the URL from the Address box and paste it into the URL box in our Add Web Reference dialog box. Then click on the green button Go to add this Web Service as a reference to our client project. You can modify this Web reference name to any name you want. In this application, we prefer to change it to WS_SQLSelect. Your finished Add Web Reference dialog box should match the one shown in Figure 9.34.

Click on the Add Reference button to finish this adding Web reference process. Immediately you can find that the following three files are created in the Solution Explorer window under the new added folder App_WebReferences\WS_SQLSelect:

- WebServiceSQLSelect.disco
- WebServiceSQLSelect.discomap
- WebServiceSQLSelect.wsdl

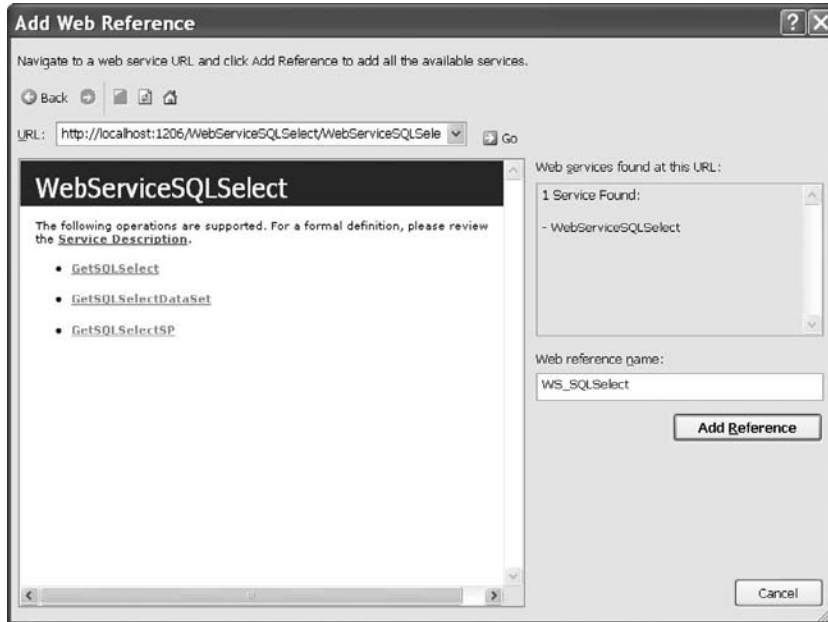


Figure 9.34 Adding a Web reference.

The only modification to the Web form of the `Faculty.aspx` is to add one more `DropDownList` control and the associated label at the top of the faculty image box control. Name this `DropDownList` as `ComboMethod` and the label as `Method`. This `DropDownList` control is used to store three Web methods developed in our Web Service. Allow users to select one of them to perform the associated data query as the project runs. Also reduce the size of the faculty image box to allow us to add this new `DropDownList` control. Your modified `Faculty.aspx` Web form window should match the one shown in Figure 9.35. Go to the `File|Save All` menu item to save these modifications.

9.3.11.3 Modify Codes for Related Methods

The first modification is to change the codes in the `Page_Load()` method and some global variables.

9.3.11.3.1 Modify Codes in Page_Load Method Perform the following changes to complete this modification:

1. Remove the namespace directory **using `System.Data.SqlClient`**; from the top of this page since we do not need it in this application.
2. Remove the field-level variable `FacultyTextBox` that is a textbox array.
3. Remove the `if` block and the associated global connection object that is stored in the Application state function `Application["sqlConnection"]`.
4. Add codes to display three Web methods in the combobox control `ComboMethod` by using the system method `Add()`.

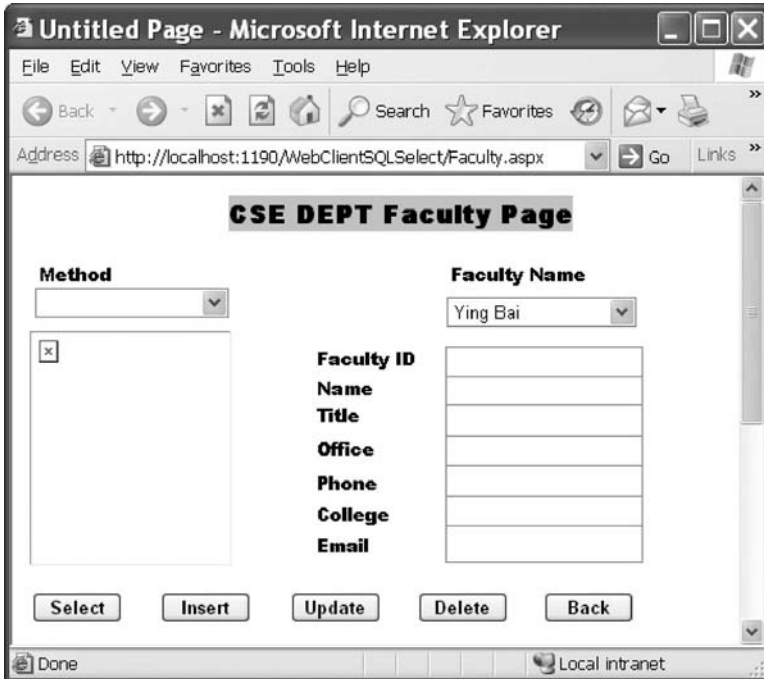


Figure 9.35 Modified Faculty Web page.

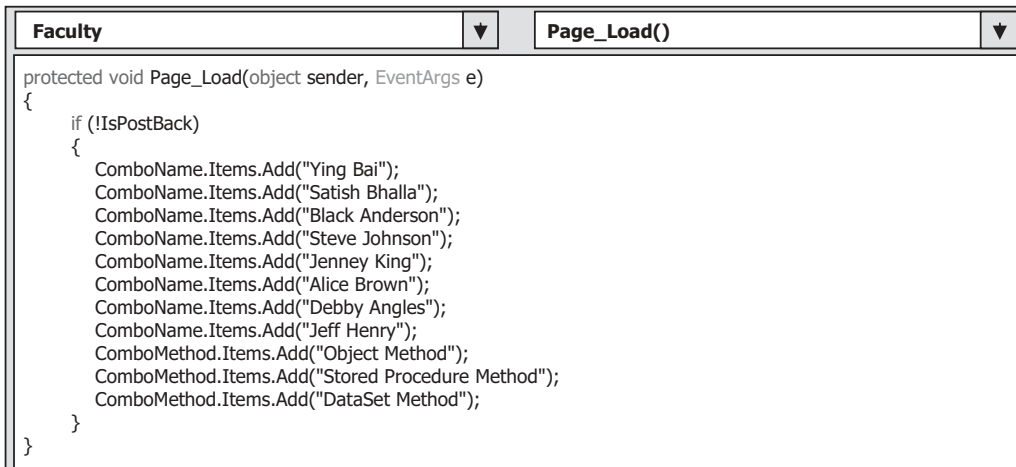


Figure 9.36 Modified Page_Load method.

Your finished coding modification to the Page_Load() method should match the one shown in Figure 9.36. The new adding codes have been highlighted in bold.

The next modification is to change the codes inside the Select button's Click method to access our Web Service to perform the data query actions against our sample database via the Web server.

9.3.11.3.2 Modify Codes in Select Button Click Method Replace all codes in this method with the following modified codes in the following sequence:

- A. Create three new instances:
 1. `wsSQLSelect` for the proxy class of our Web Service
 2. `wsSQLResult` for the child class of our Web Service
 3. `wsDataSet` for our `DataSet` class
- B. Create a local string variable `errMsg` and used it to store the possible error message.
- C. Call the user-defined method `ShowFaculty()` to display the selected faculty photo.
- D. If the user selected the **Web Object** method, a `try ... catch` block is used to call the first Web method `GetSQLSelect()` that we developed in our Web Service project with the selected faculty name as the input parameter. The returned object that contains our queried faculty information is assigned to our local mapping object `wsSQLResult` if this calling is successful. Otherwise an error message is displayed using the `Write()` method of the `Response` object of the server.
- E. The user-defined method `ProcessObject()` is executed to assign the retrieved faculty information to the associated textbox in our Web page to display them.
- F. If the user selected the **Stored Procedure Method**, the associated Web method `GetSQLSelectSP()`, which is developed in our Web Service, is called via the instance of the Web reference class to perform the data query. The `catch` statement is used to collect any possible exceptions if any error occurred for this calling, and the error message is displayed using the `Write()` method of the `Response` object of the server. Similarly the user-defined method `ProcessObject()` is executed to pick up all pieces of retrieved information from the returned object and displays them on this form page.
- G. If users selected the **DataSet Method**, the Web method `GetSQLSelectDataSet()` is called through the instance of the Web reference class, and the method returns a `DataSet` that contains our desired faculty information. The `catch` statement is used to collect any possible exceptions if any error occurred for this calling, and the error message is displayed using the `Write()` method of the `Response` object of the server.
- H. The method `FillFacultyDataSet()` is called to pick up all pieces of retrieved information from the returned `DataSet` and display them on this form page.

All of these modification steps are shown in Figure 9.37.

9.3.11.3.3 Add Three User-Defined Methods We need to add three user-defined methods—`ProcessObject()`, `FillFacultyObject()`, and `FillFacultyDataSet()`—into this project. The codes for these three methods are basically identical with those we developed in the last Windows-based Web Service client project `WinClientSQLSelect`, and one can copy and paste them into our new project with little modification.

Open our Windows-based Web Service client project `WinClientSQLSelect`, copy these three methods from that project, and paste them into the code page of our current Faculty page. The only modification we need to make is the `MessageBox()` method used in the `ProcessObject()` method. In the website project, we need to use the `Write()` method provided by the `Response` object of the server class to replace the Windows-based method `MessageBox()` to display an error message. Create a local string variable in this method to hold the possible error message. Your modified codes for this method should match one that is shown in Figure 9.38. The modified parts have been highlighted in bold.

Faculty	cmdSelect_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p> <p>F</p> <p>G</p> <p>H</p>	<pre> protected void cmdSelect_Click(object sender, EventArgs e) { WS_SQLSelect.WebServiceSQLSelect wsSQLSelect = new WS_SQLSelect.WebServiceSQLSelect(); WS_SQLSelect.SQLSelectResult wsSQLResult = new WS_SQLSelect.SQLSelectResult(); DataSet wsDataSet = new DataSet(); string errMsg; ShowFaculty(ComboName.Text); if (ComboMethod.Text == "Object Method") { try {wsSQLResult = wsSQLSelect.GetSQLSelect(ComboName.Text);} catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert('" + errMsg + "')</script>"); } ProcessObject(wsSQLResult); } else if (ComboMethod.Text == "Stored Procedure Method") { try { wsSQLResult = wsSQLSelect.GetSQLSelectSP(ComboName.Text); } catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert('" + errMsg + "')</script>"); } ProcessObject(wsSQLResult); } else { try { wsDataSet = wsSQLSelect.GetSQLSelectDataSet(ComboName.Text); } catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert('" + errMsg + "')</script>"); } } FillFacultyDataSet(wsDataSet); } </pre>

Figure 9.37 Modified codes for the Select button's Click method.

Faculty	ProcessObject()
	<pre> private void ProcessObject(ref WS_SQLSelect.SQLSelectResult wsResult) { string errMsg; errMsg = "Faculty information cannot be retrieved: " + wsResult.SQLRequestError; if (wsResult.SQLRequestOK == true) FillFacultyObject(ref wsResult); else Response.Write("<script>alert('" + errMsg + "')</script>"); } </pre>

Figure 9.38 Modified codes for the ProcessObject method.

Faculty	▼	cmdBack_Click()	▼
<pre>protected void cmdBack_Click(object sender, EventArgs e) { Response.Write("<script>window.close(</script>"); } </pre>			

Figure 9.39 Modified coding for the Back button's Click method.

There is no other modification needed for the other two user-defined methods `FillFacultyObject()` and `FillFacultyDataSet()`.

9.3.11.3.4 Modify Coding for Back Button Click Method The modification to the Back button's Click method is to use the Web-based `Close()` method to replace the `Response.Redirect()` method to terminate our Web client page project. Your modified Back button's Click method should match the one shown in Figure 9.39. The modified parts have been highlighted in bold.

Next let's perform the last modification to the coding part on this project.

9.3.11.3.5 Remove Unused User-Defined Methods Two user-defined methods—`FillFacultyReader()` and `MapFacultyTable()`—will not be used in this current application. Therefore we can remove the two methods from this project. Highlight the codes of these two methods and remove them by going to the menu item `Edit/Delete`.

Now it is time for us to run our Web-based Web Service client project to test the functionality of our data query and our Web Service. However, before we can run our project, we need to make sure that the following two things have been done:

1. Make sure that the starting page is our `Faculty.aspx` page as the project runs. To confirm that, right-click on our `Faculty.aspx` page from the Solution Explorer window and select the item `Set As Start Page` from the pop-up menu.
2. Make sure that our Web Service `WebServiceSQLSelect` has been run at least once and that Web Service status is opening, which can be identified by a small white icon located in the task bar at the bottom of the screen.

Now click on the Start Debugging button to run our project. The Faculty page is displayed and it is shown in Figure 9.40.

Keep the default Web method in the combobox control `ComboMethod` and select the faculty member `Steve Johnson` from the combobox control `ComboName`. Then click on the Select button to call the associated Web method developed in our Web Service to retrieve the selected faculty information from our sample database via the Web server. The query result is shown in Figure 9.40.

You can try to select different Web methods with different faculty members to test this project. Our Web-based Web Service client project is very successful.

A complete Web-based Web Service client project `WebClientSQLSelect` can be found at the folder `DBProjects\Chapter 9` at the accompanying ftp site (see Chapter 1).

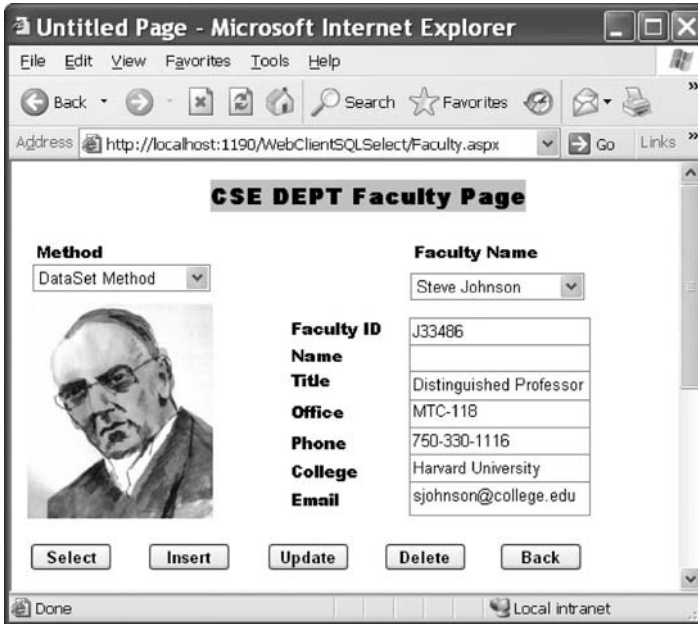


Figure 9.40 Running status of our Web-based client project.

9.3.12 Deploy the Completed Web Service to Production Servers

When we finished developing and testing our Web Service in our local machine, we need to deploy it to the .NET SDK or an Internet Information Services (IIS) 5 or higher virtual directory to allow users to access and use it via a production server. We could have discussed this topic in an earlier section when we finished developing our Web Service project. The reason we delayed this discussion until this section is that we wanted to show readers that we do not have to perform this Web Service deployment if we are running our Web Service and accessing it using a client project in our local computer (development server). However, you must deploy your Web Service to IIS if you want to run it in a formal Web server (production server).

Basically, there are two ways to do this deployment: First, you can copy our Web Service files to a server running the IIS 5 or higher, or to the folder that is or contains our virtual directory. Another way is to use the Builder provided by the Visual Studio .NET to precompile the Web pages and copy the compiled files to our virtual directory. The so-called virtual directory is a default directory that can be recognized and accessed by a Web server such as IIS to run our Web Services. In both ways, we need a virtual directory to store our Web Service files and allow Web server to pick up and run our Web Service from that virtual directory. Let's see how to create an IIS virtual directory.

The following steps describe how to create an IIS virtual directory using the IIS Manager:

1. First, create a new folder to save our virtual directory's files. Typically we need to create this folder under the default Web Service root folder `C:\inetpub\wwwroot`. In our case, create a new folder named `WSSQLSelect` and place it under the root folder `C:\inetpub\wwwroot`.
2. Open the IIS Manager by first clicking on the **Performance and Maintenance** icon and then clicking on the **Administrative Tools** icon from the Control Panel. On the opened dialog window, double-click on the icon **Computer Management**. Then, expand the item **Services and Applications** from the opened dialog and continue to expand the item **Internet Information Services**. Two items are listed under this icon: **Web Sites** and **Default SMTP Virtual Server**.
3. Expand the **Web Sites** folder and right-click on the expanded item **Default Web Site** and select the item **New|Virtual Directory** from the pop-up menu to open the **Creation Wizard**. Click on **Next** to go to the next step.
4. Enter `WSSQLSelect` into the **Alias** box as the name for this virtual directory, and then click on the **Next** to continue.
5. In the next window, click on the **Browse** button to find the folder we created at step 1, which is `WSSQLSelect`. Click on **OK** and then **Next** to go to the next step.
6. Keep all default setting in the opened window and click on **Next** to continue.
7. Click on the **Finish** button to complete this process.

Our virtual directory is created but the story is not finished. As you know, there is no `Default.asmx` page in our Web Service project. So in order to allow the Web server to find our starting page, we need to modify the default page for this virtual directory. Follow these steps to finish this modification:

1. Right-click on our new created virtual directory `WSSQLSelect` from the **Computer Management** window and select the **Properties** item to open the **Property** window.
2. Click on the **Documents** tab from the **Properties** window and remove all items from the listbox by selecting them and clicking on the **Remove** button.
3. Click on the **Add** button to open the **Add Default Document** dialog box. Enter our starting page `WebServiceSQLSelect.asmx` into the **Default Document Name** box as our default page; then click on **OK**.
4. Click on the **Apply** and then **OK** buttons to close this window.

After our virtual directory is set up, we can deploy our Web Service by either copying files to this virtual directory or performing a precompile process. First, let's do that by copying all files to the virtual directory since it is relatively simple.

9.3.12.1 Copy Web Service Files to Virtual Directory

Perform the following steps to complete this copying process:

1. Open Visual Studio.NET and our Web Service project `WebServiceSQLSelect`.
2. Go to the **Website|Copy Web Site** menu item to open the **Copy Web Site** window shown in Figure 9.41.
3. Click on the **Connect** button located to the right of the **Connections** text box.

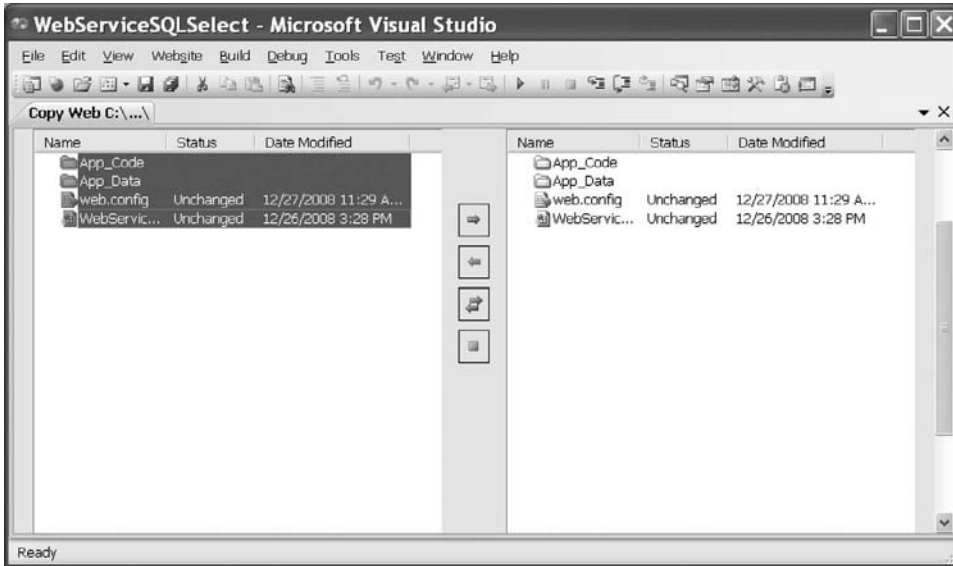


Figure 9.41 Copying Web Service files to the virtual directory.

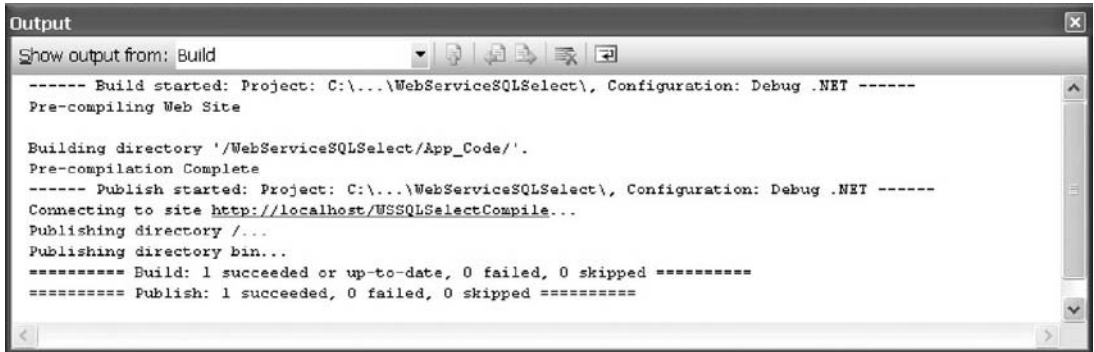
4. On the opened window, click on the Local IIS icon from the left pane and then expand the Default Web Site item to find our virtual directory WSSQLSelect. Click on this item to select it and then click on the Open button.
5. Select all files and folders from our Web Service project, and click on the right-arrow button to copy all files to our virtual directory, as shown in Figure 9.41.
6. Now go to the File|Open Web Site menu item to open the Open Web Site dialog window. Click on the Local IIS icon from the left pane and select our virtual directory WSSQLSelect, and then click on the Open button. Click on Yes to the message box to allow our site to be configured for use with ASP.NET 2.0 if this message box is displayed.
7. On the opened Web Service project, open the web.config file and replace the attribute `<compilation debug="true"/>` with `<compilation debug="false"/>`.
8. Rebuild our Web Service project and run it again. This will replace the built-in Web server. Check the Run without Debugging radio button if a warning message box is displayed.

Next we will discuss how to publish a Web Service to the production server using the precompiled Web Service method.

9.3.12.2 Publish Precompiled Web Service

Before publishing our Web Service to a production server, make sure that a virtual directory has been created. In our case, this virtual directory is a new folder named WSSQLSelectCompile, and it is located under the root folder C:\inetpub\wwwroot. Follow steps 1 to 7 listed in Section 9.3.12 to create this virtual directory if you have not done so yet.

Now open our Web Service project if it is not opened. Go to the Build|Publish Web Site menu item to open the Publish Web Site dialog box. Enter the virtual directory we



```

Output
Show output from: Build
----- Build started: Project: C:\...\WebServiceSQLSelect\, Configuration: Debug .NET -----
Pre-compiling Web Site

Building directory '/WebServiceSQLSelect/App_Code/'.
Pre-compilation Complete
----- Publish started: Project: C:\...\WebServiceSQLSelect\, Configuration: Debug .NET -----
Connecting to site http://localhost/WSSQLSelectCompile...
Publishing directory /...
Publishing directory bin...
----- Build: 1 succeeded or up-to-date, 0 failed, 0 skipped -----
----- Publish: 1 succeeded, 0 failed, 0 skipped -----

```

Figure 9.42 Processing result of the Web Service publishing.

created above, which is **http://localhost/WSSQLSelectCompile**, into the Target Location box as our target directory, keep the default setups unchanged, and click on the OK button to begin the publishing process.

As the publishing process is completed, the processing and the output of this publishing process are displayed in the Output window. To see what happened to this process, open this Output window by going to View|Other Windows|Output. A sample processing result is shown in Figure 9.42.

Another way to check this publishing result is to open the virtual directory we created for this published Web Service to inspect the associated compiled files. To do that, open the Windows Explorer window and browse to our virtual directory `C:\inetpub\wwwroot\WSSQLSelectCompile`. You can find that two terminal files named `App_Code.compiled` and `App_Code.dll` are located in the bin folder under this virtual directory. The first file corresponds to pages and the second file contains the executable code for the Web Service, such as the class file that you created. Remember that the page, its code, and the separate class file that you created have all been compiled into the executable code.

To test this published Web Service, you can open the Microsoft Internet Explorer (IE) and type our virtual directory **http://localhost/WSSQLSelectCompile** as the URL into the Address box to try to open our service page.

At this point, we have finished the discussion about how to create and consume a Web Service using a Windows-based and a Web-based Web service client project. In the following sections, we will expand these discussions to perform the data insertion and data updating and deleting actions against our sample database through the Web Services.

9.4 BUILD ASP.NET WEB SERVICE PROJECT TO INSERT DATA INTO SQL SERVER DATABASE

In this section we want to discuss how to insert data into our sample database through a Web Service developed in Visual Studio.NET. The data table we try to use for this data action is the Course table. In other words, we want to insert a new course for the selected faculty into the Course table via a Web Service we will develop in this section.

To save time and space, we can copy and modify an existing Web Service project `WebServiceSQLSelect` we developed in the previous section as our new Web Service project `WebServiceSQLInsert`.

9.4.1 Modify Existing Web Service Project

First, let's create a new folder such as Chapter 9 in our root directory using Windows Explorer, and then copy the `WebServiceSQLSelect` project from the folder `DBProjects\Chapter 9` located at the accompanying ftp site (see Chapter 1), and paste it into our new created folder `C:\Chapter 9`. Rename it to `WebServiceSQLInsert` and open this new project to perform the following modifications to this project:

1. Change the main Web Service page from `WebServiceSQLSelect.aspx` to `WebServiceSQLInsert.aspx` in the Solution Explorer window.
2. Change the name of our base class from `SQLSelectBase`, which is located in the folder `App_Code`, to `SQLInsertBase` in the Solution Explorer window.
3. Open Visual Studio.NET and our new project `WebServiceSQLInsert`, and then open our entry page `WebServiceSQLInsert.aspx` by double-clicking on it, and change the compiler directive from

```
CodeBehind="~/App_Code/WebServiceSQLSelect.cs"
to
CodeBehind="~/App_Code/WebServiceSQLInsert.cs"
```

Also change the class name from

```
Class="WebServiceSQLSelect"
to
Class="WebServiceSQLInsert"
```

4. Remove the child class `SQLSelectResult` from this project since the data insertion has no data to be returned.
5. Change the name of our code-behind page from `WebServiceSQLSelect.cs` to `WebServiceSQLInsert.cs`.
6. Open the base class `SQLInsertBase` and perform the following modifications:
 - a. Change the class name from `SQLSelectBase` to `SQLInsertBase`.
 - b. Change two member data from `SQLRequestOK` to `SQLInsertOK` and from `SQLRequestError` to `SQLInsertError`.
 - c. Add the following seven member data into this class:
 - i. `public string FacultyID;`
 - ii. `public string[] CourseID = new string[10];`
 - iii. `public string Course;`
 - iv. `public string Schedule;`
 - v. `public string Classroom;`
 - vi. `public int Credit;`
 - vii. `public int Enrollment;`

Go to the `File|Save All` menu item to save these modifications.

9.4.2 Web Service Project Development Procedure

We try to develop four Web methods in this Web Service project; two of them are used to insert the desired course information into our sample database and two of them are used to retrieve the new inserted course information from the database to test the data insertion. The fourth Web method is used to retrieve the detailed course information based on the `course_id`. These methods are listed below:

1. Develop a Web method `SetSQLInsertSP()` to call a stored procedure to perform this new course insertion.
2. Develop a Web method `GetSQLInsert()` to retrieve the new inserted course information from the database using a joined table query.
3. Develop a Web method `SQLInsertDataSet()` to perform the data insertion by using multi-query and return a `DataSet` that contains the updated Course table.
4. Develop a Web method `GetSQLInsertCourse()` to retrieve the detailed course information based on the input `course_id`.

The reason we use two different methods to perform this data insertion is to try to compare them. As you know, there is no `faculty_name` column in the Course table, and each course is related to a faculty member identified by the `faculty_id` column in the Course table. In order to insert a new course into the Course table, you must first perform a query to the Faculty table to get the desired `faculty_id` based on the selected faculty name, and then you can perform another insertion query to insert a new course based on that `faculty_id` obtained from the first query. The first Web method combines those two queries into a stored procedure, and the third method uses a `DataSet` to return the whole Course table to make this data insertion more convenient to the user.

The main code developments and modifications are performed in our code-behind page `WebServiceSQLInsert.cs`, that is, the most modifications will be performed on the codes in four Web methods listed above.

9.4.3 Develop and Modify Codes for Code-Behind Page

Open our new project `WebServiceSQLInsert` if it has not been opened, and then open the code window of our code-behind page `WebServiceSQLInsert.cs`. Change the names of our main Web service class and constructor from `WebServiceSQLSelect` to `WebServiceSQLInsert`.

Another modification is to remove the user-defined method `FillFacultyReader()` since we do not need to return any data for this data insertion operation. The last modification to this page is to modify the codes of the method `ReportError()`. Perform the following modifications to this method:

1. Change the data type of the passed argument from `SQLSelectResult` to `SQLInsertBase`.
2. Change the member data in the first coding line from `SQLRequestOK` to `SQLInsertOK`.
3. Change the member data in the second coding line from `SQLRequestError` to `SQLInsertError`.

Now let's start our modification to the first Web method.

9.4.3.1 Develop and Modify First Web Method SetSQLInsertSP

Perform the following modifications to the Web method GetSQLSelect(), as shown in Figure 9.43, to get our new Web method SetSQLInsertSP. This Web method uses a stored procedure to perform data insertion. Recall that in Section 6.10.1.2 in Chapter 6, we developed a stored procedure **dbo.InsertFacultyCourse** in the SQL Server database and used it to insert a new course into the Course table. We will use this stored procedure in this Web method to reduce our coding load. Refer to that section to get more detailed information in how to develop this stored procedure. Seven input parameters are used for this stored procedure: @FacultyName, @CourseID, @Course, @Schedule, @Classroom,

The screenshot shows the Visual Studio code editor with the following content:

```

WebServiceSQLInsert | SetSQLInsertSP()
public class WebServiceSQLInsert : System.Web.Services.WebService
{
    public WebServiceSQLInsert()
    {
        //Uncomment the following line if using designed components
        //InitializeComponent();
    }
    [WebMethod]
    public SQLInsertBase SetSQLInsertSP(string FacultyName, string CourseID, string Course, string Schedule,
        string Classroom, int Credit, int Enroll)
    {
        string cmdString = "dbo.InsertFacultyCourse";
        SqlConnection sqlConnection = new SqlConnection();
        SQLInsertBase SetSQLResult = new SQLInsertBase();
        SqlCommand sqlCommand = new SqlCommand();
        int intInsert = 0;
        SetSQLResult.SQLInsertOK = true;
        sqlConnection = SQLConn();
        if (sqlConnection == null)
        {
            SetSQLResult.SQLInsertError = "Database connection is failed";
            ReportError(SetSQLResult);
            return null;
        }
        sqlCommand.Connection = sqlConnection;
        sqlCommand.CommandType = CommandType.StoredProcedure;
        sqlCommand.CommandText = cmdString;
        sqlCommand.Parameters.Add("@FacultyName", SqlDbType.Text).Value = FacultyName;
        sqlCommand.Parameters.Add("@CourseID", SqlDbType.Char).Value = CourseID;
        sqlCommand.Parameters.Add("@Course", SqlDbType.Text).Value = Course;
        sqlCommand.Parameters.Add("@Schedule", SqlDbType.Char).Value = Schedule;
        sqlCommand.Parameters.Add("@Classroom", SqlDbType.Text).Value = Classroom;
        sqlCommand.Parameters.Add("@Credit", SqlDbType.Int).Value = Credit;
        sqlCommand.Parameters.Add("@Enroll", SqlDbType.Int).Value = Enroll;
        intInsert = sqlCommand.ExecuteNonQuery();
        sqlConnection.Close();
        sqlCommand.Dispose();
        if (intInsert == 0)
        {
            SetSQLResult.SQLInsertError = "Data insertion is failed";
            ReportError(SetSQLResult);
        }
        Return SetSQLResult;
    }
}

```

Figure 9.43 Modification to the first Web method.

@Credit, and @Enroll. All of these parameters will be input by the user as this Web Service project runs.

Let's take a closer look at these codes to see how they work.

- A.** The name of our Web Service class and the constructor of this class is changed to `WebServiceSQLInsert` to distinguish it from the original items.
- B.** The Web method's name is also changed to `SetSQLInsertSP`, which means that this Web method will call a stored procedure to perform the data insertion action. Seven input parameters are passed into this method as the new data for a new inserted course record. The returned object should be an instance of our modified base class `SQLInsertBase`.
- C.** The content of the query string must be equal to the name of the stored procedure we developed in Section 6.10.1.2 in Chapter 6. Otherwise a possible running error may be encountered as this Web Service is executed since the stored procedure is identified by its name when it is called.
- D.** A returned object `SetSQLResult` is created based on our modified base class `SQLInsertBase`; that is, no data is supposed to be returned for this data insertion action. However, in order to enable our client project to get a clear feedback from executing this Web Service, we prefer to return an object that contains the information indicating whether this Web Service is executed successfully or not.
- E.** A local integer variable `intInsert` is declared, and this variable is used to stop the returned value from calling the `ExecuteNonQuery()` method of the `Command` class, and that method will run the stored procedure to perform the data insertion action. This returned value is equal to the number of rows that have been successfully inserted into our database.
- F.** Initially we set the member data `SQLInsertOK` that is located in our modified base class `SQLInsertBase` to `true` to indicate our Web service running status is good.
- G.** If the connection to our sample database has failed, which is indicated by a returned `Connection` object containing a `null`, an error message is assigned to another member data `SQLInsertError` that is also located in our modified base class `SQLInsertBase` to log on this error, and the user-defined method `ReportError()` is called to report this error.
- H.** The property value `CommandType.StoredProcedure` must be assigned to the `CommandType` property of the `Command` object to tell the project that a stored procedure should be called as this `Command` object is executed.
- I.** Seven input parameters are assigned to the `Parameters` collection property of the `Command` object, and the last six parameters work as the new course data to be inserted into the `Course` table. One important point to be note is that these input parameters' names must be identical with those names defined in the stored procedure `dbo.InsertFacultyCourse` developed in Section 6.10.1.2. Refer to that section to get a detailed description of those parameters' names defined in that stored procedure.
- J.** The `ExecuteNonQuery()` method is called to run the stored procedure to perform this data insertion. This method returns an integer that is stored in our local variable `intInsert`.
- K.** A cleaning job is performed to release data objects used in this method.
- L.** The returned value from calling of the `ExecuteNonQuery()` method, which is stored in the variable `intInsert`, is equal to the number of rows that have been successfully inserted into the `Course` table. If this value is zero, which means that no row has been inserted into our database and this data insertion has failed, a warning message is assigned to the member data `SQLInsertError` that will be reported by using our user-defined `ReportError()` method later.

M. Finally the instance of our base class, `SetSQLResult`, is returned to the calling procedure to indicate the running result of this Web method.

At this point we have finished the coding development and modification to this Web method. Now we can run this Web Service project to test inserting new course information to our sample database via this Web Service. However, before we can build and run this project, we have to comment out all other Web methods we created in the project `WebServiceSQLSelect` we developed in the previous sections since we have modified some user-defined classes such as `SQLSelectBase` that are still used in those Web methods in this project. The Web methods to be commented out are:

- `GetSQLSelectSP()`
- `GetSQLSelectDataSet()`

After comment out these Web methods, now we can build and run this project to test the data insertion action. Click on the Start Debugging button to run the project. The built-in Web interface is shown in Figure 9.44.

Click on the Web method `SetSQLInsertSP` to select it to open another built-in Web interface to display input parameters window, which is shown in Figure 9.45.

Enter the following parameters to this Web method:

- FacultyName Ying Bai
- CourseID CSE-556
- Course Advanced Fuzzy Systems
- Schedule M-W-F: 1:00-1:55 PM
- Classroom TC-315
- Credit 3
- Enroll 28



Figure 9.44 Running built-in Web interface.



Figure 9.45 Input parameter interface.

Click on the Invoke button to run this Web method to call the stored procedure to perform this data insertion. The running result is displayed in the built-in Web interface, which is shown in Figure 9.46.

Based on the returned member data `SQLInsertOK = true`, it indicates that our data insertion is successful. To confirm this, first click on the Close button located at the upper-right corner of this Web interface to terminate our Web Service, and then you can open our sample database `CSE_DEPT` using the SQL Server Management Studio to check this new inserted course.

It can be found from this running result that the values for both attributes `<Credit>` and `<Enrollment>` are zero. This makes sense since we did not query any data from our Course table and assign any returned course information to them, and the default value is zero for any created integer variable. Ten string variables belong to the string array `CourseID[]` that is used to store `course_id`. A default value `true` is assigned to each of them since we did not need to return them in this application.

Next let's develop the second Web method `GetSQLInsert()` to retrieve all courses taught by the selected faculty member, that is, all `course_id`, to confirm our data insertion action.

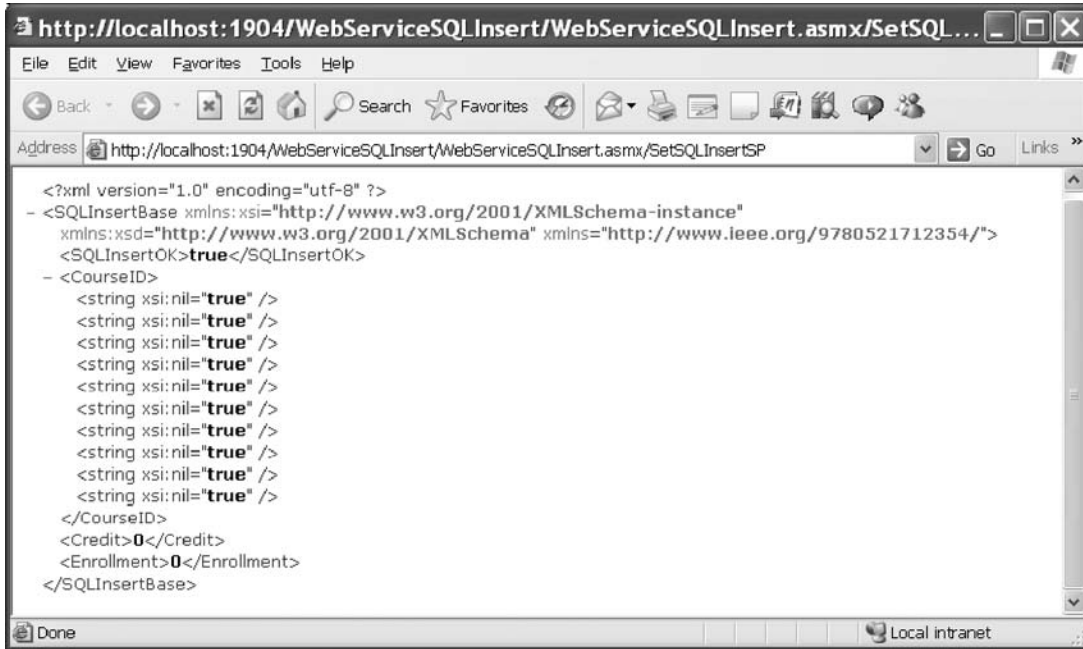


Figure 9.46 Running result of the first Web method.

9.4.3.2 Develop Second Web Method GetSQLInsert

The function of this Web method is to retrieve all `course_id`, which includes the original and the new inserted `course_id`, from the Course table based on the input faculty name. This Web method will be called or consumed by a client project later to get back and display all `course_id` in a listbox control in the client project.

Recall that in Section 5.19.2.5 in Chapter 5, we developed a joined-table query to perform the data query from the Course table to get all `course_id` based on the faculty name. The reason for that is because there is no faculty name column available in the Course table, and each course or `course_id` is related to a `faculty_id` in the Course table. In order to get the `faculty_id` that is associated with the selected faculty name, one must first go to the Faculty table to perform a query to obtain it. In this situation, a join query is a desired method to complete this function.

We will use the same strategy to perform this data query in this section. Open the code window of our code-behind page `WebServiceSQLInsert.cs` and enter the codes shown in Figure 9.47 into this page to create our new Web method `GetSQLInsert()`.

Let's take a closer look at the codes in this Web method to see how they work.

- A.** The returning data type for this Web method is our modified base class `SQLInsertBase`, and all course information is stored in the different member data in this class. The input parameter to this Web method is a selected faculty name.
- B.** The joined-table query string is defined here, and an ANSI92 standard, which is an up-to-date standard, is used for the syntax of this query string. The ANSI 89, which is an out-of-date syntax standard, can still be used for this query string definition. However, the

WebServiceSQLInsert		GetSQLInsert()
	[WebMethod]	
A	public SQLInsertBase GetSQLInsert(string FacultyName)	
	{	
B	string cmdString = "SELECT Course.course_id FROM Course JOIN Faculty " +	
	"ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.faculty_name LIKE @name)";	
C	SqlConnection sqlConnection = new SqlConnection();	
	SQLInsertBase GetSQLResult = new SQLInsertBase();	
	SqlCommand sqlCommand = new SqlCommand();	
	SqlDataReader sqlReader;	
D	GetSQLResult.SQLInsertOK = true;	
E	sqlConnection = SQLConn();	
	if (sqlConnection == null)	
	{	
	GetSQLResult.SQLInsertError = "Database connection is failed";	
	ReportError(GetSQLResult);	
	return null;	
	}	
F	sqlCommand.Connection = sqlConnection;	
	sqlCommand.CommandType = CommandType.Text;	
	sqlCommand.CommandText = cmdString;	
G	sqlCommand.Parameters.Add("@name", SqlDbType.Text).Value = FacultyName;	
H	sqlReader = sqlCommand.ExecuteReader();	
I	if (sqlReader.HasRows == true)	
	FillCourseReader(ref GetSQLResult, sqlReader);	
J	else	
	{	
	GetSQLResult.SQLInsertError = "No matched course found";	
	ReportError(GetSQLResult);	
	}	
K	sqlReader.Close();	
	sqlConnection.Close();	
	sqlCommand.Dispose();	
L	return GetSQLResult;	
	}	

Figure 9.47 Codes for our second Web GetSQLInsert method.

up-to-date standard is recommended. Refer to Section 5.19.2.5 to get more detailed discussions for this topic. The nominal name of the input dynamic parameter to this query is @name.

- C. All used data objects are declared here, such as the Connection, Command, and DataReader objects. A returned object GetSQLResult that is instantiated from our base class SQLInsertBase is also created, and it will be returned to the calling procedure to send back the queried course information.
- D. Initially we set the running status of our Web method to OK.
- E. The user-defined method SQLConn() is called to connect to our sample database. A warning message is assigned to the member data in our returned object and the user-defined method ReportError() is executed to report this error if an error occurs for this connection.
- F. The Command object is initialized with appropriate properties such as the Connection object, Command type, and Command text.
- G. The real input parameter FacultyName is assigned to the dynamic parameter @name using the Add() method.

WebServiceSQLInsert	FillCourseReader()
<pre>protected void FillCourseReader(ref SQLInsertBase sResult, SqlDataReader sReader) { int pos = 0; while (sReader.Read()) { sResult.CourseID[pos] = Convert.ToString(sReader.GetSqlString(0)); //the 1st column is course_id pos++; } }</pre>	

Figure 9.48 Codes for the FillCourseReader method.

- H.** The ExecuteReader() method is called to trigger the DataReader and perform the data query. This method is a read-only method and the returned reading result is assigned to the DataReader object sqlReader.
- I.** By checking the HasRows property of the DataReader, we can determine whether this reading is successful or not. If this reading is successful (HasRows = true), the user-defined FillCourseReader() method, whose detailed codes will be discussed in Figure 9.48, is called to assign the returned course_id to each associated member data in our returned object GetSQLResult.
- J.** Otherwise if this reading has failed, a warning message is assigned to our member data SQLInsertError in our returned object and this error is reported by calling the user-defined ReportError() method.
- K.** A cleaning job is performed to release all data objects used in this Web method.
- L.** The returned object that contains all queried course_id is returned to the calling procedure.

The detailed codes for our user-defined FillCourseReader() method are shown in Figure 9.48.

The function of this piece of code is straightforward and without tricks. A while loop is used to continuously pick up each course_id whose column index is zero from the Course table, convert it to a string, and assign it to the CourseID string array defined in our base class SQLInsertBase.

Now let's test this Web method by running this project. Build this project and click on the Start Debugging button to run our project. The built-in Web interface is displayed in Figure 9.49.

Click on the first Web method GetSQLInsert and enter the faculty name Ying Bai into the FacultyName box in the next built-in Web interface, which is shown in Figure 9.50.

Click on the Invoke button to execute this Web method, and the running result of this method is shown in Figure 9.51. It can be seen that all courses (that is, all course_id), including our new inserted course CSE-556, taught by the selected faculty Ying Bai are listed in an XML format.

Our second Web method is successful, too. Click on the Close button located at the upper-right corner of this page to terminate our Web Service project. Also go to File|Save All to save all methods we have developed.

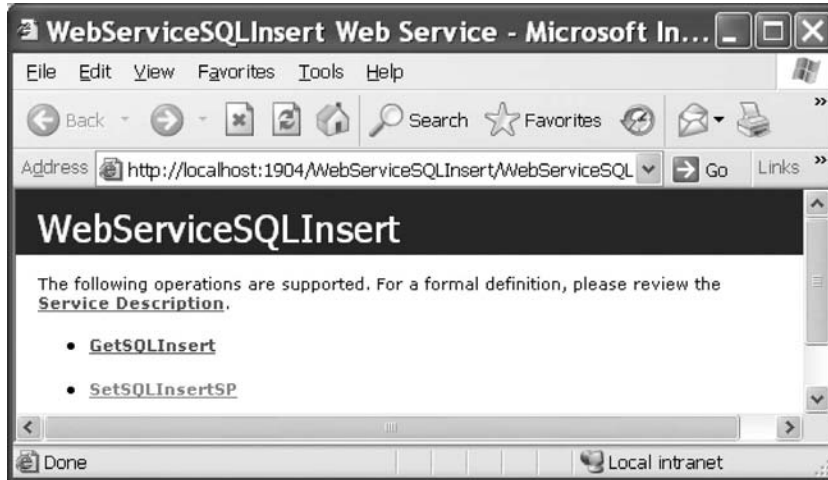


Figure 9.49 Running status of our Web Service project.

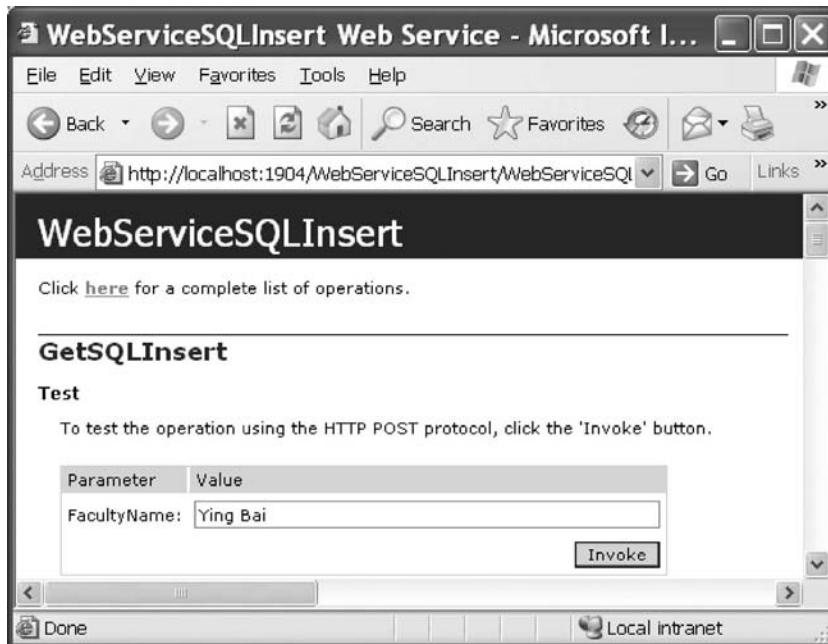


Figure 9.50 Running status of our Web Service project.

9.4.3.3 Develop and Modify Third Web Method SQLInsertDataSet

The function of this Web method is similar to the first one: to insert a new course into the Course table based on the selected faculty member. The difference is that this Web method uses multiquery to insert a new course record into the Course table and uses a DataSet as the returned object. The returned DataSet contains the updated Course table

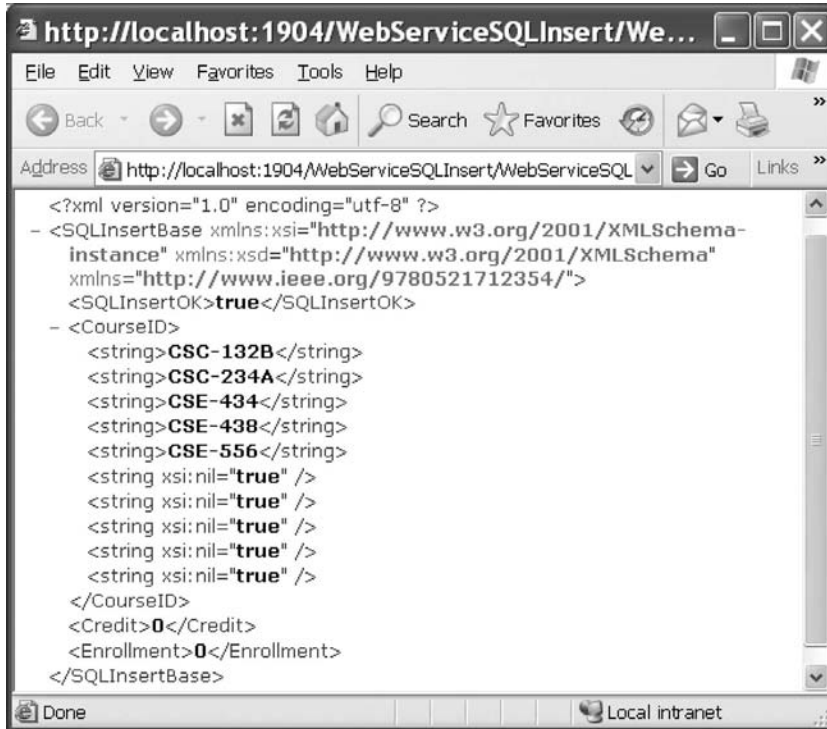


Figure 9.51 Running result of our Web method GetSQLInsert.

that includes the new inserted data. The advantages of using a DataSet as the returned object are:

1. Unlike Web methods 1 and 2, which are a pair of methods with the first used to insert data into the database and the second used to retrieve the new inserted data from the database to confirm the data insertion, method 3 contains both inserting data into and retrieving data back functions. Later when a client project is developed to consume this Web Service, methods 1 and 2 must be called together from that client project to perform both data insertion and data validation jobs. However, method 3 has both data insertion and data validation functions; therefore, it can be called independently.
2. Because a DataSet is returned, we do not need to create any new instance for our base class as the returned object. However, in order to report or log on any exception encountered when the project runs, we still need to create and use an instance of our base class to handle those error-processing issues.

Modify our existing Web method GetSQLSelectDataSet() in this project and make it our new Web method SQLInsertDataSet. Our finished Web method should match the one shown in Figure 9.52.

Let's take a closer look at the codes in this Web method to see how they work.

- A. The name of the Web method is SQLInsertDataSet(). Seven input parameters are passed into this method as the data for a new inserted record, and the returned data type is DataSet.

WebServiceSQLInsert	SQLInsertDataSet()
	[WebMethod]
A	public DataSet SQLInsertDataSet(string FacultyName, string CourseID, string Course, string Schedule, string Classroom, int Credit, int Enroll)
	{
B	string cmdString = "INSERT INTO Course VALUES (@course_id, @course, @credit, @classroom, " + "@schedule, @enrollment, @faculty_id)";
C	SqlConnection sqlConnection = new SqlConnection(); SQLInsertBase SetSQLResult = new SQLInsertBase(); SqlCommand sqlCommand = new SqlCommand(); SqlDataAdapter CourseAdapter = new SqlDataAdapter(); DataSet dsCourse = new DataSet(); int intResult = 0; string FacultyID;
D	SetSQLResult.SQLInsertOK = true;
E	sqlConnection = SQLConn(); if (sqlConnection == null) { SetSQLResult.SQLInsertError = "Database connection is failed"; ReportError(SetSQLResult); return null; }
F	sqlCommand.Connection = sqlConnection; sqlCommand.CommandType = CommandType.Text;
G	sqlCommand.CommandText = "SELECT faculty_id FROM Faculty WHERE faculty_name LIKE @Name";
H	sqlCommand.Parameters.Add("@Name", SqlDbType.Text).Value = FacultyName;
I	FacultyID = (string)sqlCommand.ExecuteScalar();
J	sqlCommand.CommandText = cmdString;
K	sqlCommand.Parameters.Add("@faculty_id", SqlDbType.Text).Value = FacultyID; sqlCommand.Parameters.Add("@course_id", SqlDbType.Char).Value = CourseID; sqlCommand.Parameters.Add("@course", SqlDbType.Text).Value = Course; sqlCommand.Parameters.Add("@schedule", SqlDbType.Char).Value = Schedule; sqlCommand.Parameters.Add("@classroom", SqlDbType.Text).Value = Classroom; sqlCommand.Parameters.Add("@credit", SqlDbType.Int).Value = Credit; sqlCommand.Parameters.Add("@enrollment", SqlDbType.Int).Value = Enroll;
L	CourseAdapter.InsertCommand = sqlCommand;
M	intResult = CourseAdapter.InsertCommand.ExecuteNonQuery();
N	if (intResult == 0) { SetSQLResult.SQLInsertError = "No matched course found"; ReportError(SetSQLResult); }
O	sqlCommand.CommandText = "SELECT * FROM Course WHERE faculty_id LIKE @FacultyID";
P	sqlCommand.Parameters.Add("@FacultyID", SqlDbType.Text).Value = FacultyID;
Q	CourseAdapter.SelectCommand = sqlCommand;
R	CourseAdapter.Fill(dsCourse, "Course");
S	CourseAdapter.Dispose(); sqlConnection.Close(); sqlCommand.Dispose();
T	return dsCourse; }

Figure 9.52 Codes for the Web SQLInsertDataSet method.

- B.** The data insertion query string is declared here. In fact, in total, we have three query strings in this method. The first two queries are used to perform the data insertion, and the third one is used to retrieve the new inserted data from the database to validate the data insertion. For the data insertion, first we need to perform a query to the Faculty table to get the matched `faculty_id` based on the input faculty name since there is no faculty name

column available in the Course table and each course is related to a `faculty_id`. Second, we can insert a new course record into the Course table by executing another query based on the `faculty_id` obtained from the first query. The query string declared here is the second string.

- C. All data objects and variables used in this Web method are declared here, which include the Connection, Command, DataAdapter, DataSet, and an instance of our base class `SQLInsertBase`. The local integer variable `intResult` is used to hold the returned value from calling the `ExecuteNonQuery()` method, and the local string variable `FacultyID` is used to reserve the `faculty_id` that is obtained from the first query.
- D. The member data `SQLInsertOK` is initialized to the normal case.
- E. The user-defined `SQLConn()` method is called to perform the database connection. A warning message will be displayed and reported using the method `ReportError()` if this connection encountered any error.
- F. The Command object is first initialized to perform the first query—get the desired `faculty_id` from the Faculty table based on the input faculty name.
- G. The first query string is assigned to the `CommandText` property.
- H. The dynamic parameter `@Name` is assigned with the actual input parameter `FacultyName`.
- I. The `ExecuteScalar()` method of the Command object is called to perform the first query to pick up the `faculty_id`, and the returned `faculty_id` is assigned to the local string variable `FacultyID`. One point to be noted is the data type that the `ExecuteScalar()` method returned. An Object type is returned from calling this method in the normal case. Therefore, a `string` casting is used to convert this object to a string and assign it to the local string variable `FacultyID`.
- J. The second query string is assigned to the `CommandText` property of the Command object to make it ready to perform the second query; insert a new course record into the Course table.
- K. All seven input parameters to the INSERT command are initialized by assigning them with the actual input values. The point to note is the data types of the last two parameters. Both `credit` and `enrollment` are integers therefore the data type `SqlDbType.Int` is used for both of them.
- L. The initialized Command object is assigned to the `InsertCommand` property of the `CourseDataAdapter`.
- M. The `ExecuteNonQuery()` method is called to perform this data insertion query to insert a new course record into the Course table in our sample database. This method will return an integer to indicate the number of rows that have been successfully inserted into the database.
- N. If this returned integer is zero, which means that no row has been inserted into the database and this insertion has failed, a warning message is assigned to the member data `SQLInsertError`, and our method `ReportError()` is called to report this error.
- O. The third query string, which is used to retrieve all courses including the new inserted courses from the database based on the input `faculty_id`, is assigned to the `CommandText` property of the Command object.
- P. The dynamic parameter `faculty_id` is initialized with the actual `faculty_id` obtained from the first query as we did above.
- Q. The initialized Command object is assigned to the `SelectCommand` property of the `DataAdapter`.

- R. The Fill() method of the DataAdapter is executed to retrieve all courses, including the new inserted courses, from the database and add them into the DataSet dsCourse.
- S. A cleaning job is performed to release all objects used in this Web method.
- T. Finally the DataSet that contains the updated course information is returned to the calling procedure.

Compared to the first Web method, it looks like more codes are involved in this method. Yes, it is true. However, this method has two functions: inserting data into the database and validating the inserted data from the database. In order to validate the data insertion for the first method, the second Web method must be executed. Therefore from the point of view of data insertion and data validation processes, the third Web method has less coding compared to the first one.

Now let's build and run our Web Service project to test this Web method using the built-in Web interface. Click on the Start Debugging button to run the project and click on our Web method SQLInsertDataSet from the built-in Web interface to start it. The parameters dialog box is displayed, which is shown in Figure 9.53. Enter the following parameters into each associated Value box as the data item of a new course:



Figure 9.53 Finished parameter dialog box.

- FacultyName Ying Bai
- CourseID CSE-665
- Course Neural Network Systems
- Schedule T-H: 1:00-2:25 PM
- Classroom TC-309
- Credit 3
- Enroll 32

Your finished parameter dialog box should match the one shown in Figure 9.53.

Click on the Invoke button to run this Web method to perform this new course insertion. The running result is shown in Figure 9.54.

All six courses, including the sixth course CSE-665, which is the new inserted course, are displayed in the XML format or tags in this running result dialog box.

A point to note is that you can only insert this new course record into the database once, which means that after this new course has been inserted into the database, you cannot continue to click on the Invoke button to perform another insertion with the same course information since the data to be inserted into the database must be unique.

Click on the Close button located at the upper-right corner of this Web interface to terminate our service. Next let's develop our fourth Web method.

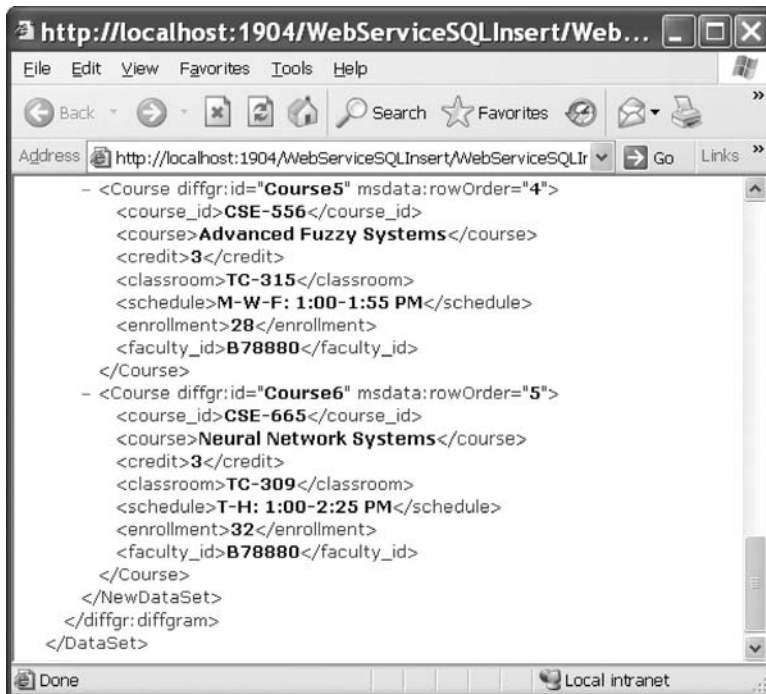


Figure 9.54 Running result of our third Web method.

9.4.3.4 Develop Fourth Web Method GetSQLInsertCourse

The function of this method is to retrieve the detailed course information from the database based on the input `course_id`. This method can be used by a client project when users want to get the detailed course information such as the course name, schedule, classroom, credit, enrollment, and `faculty_id` when a `course_id` is selected from a listbox control.

Because this query is a single query, you can use either a normal query or a stored procedure if you want to reduce the coding load in this method. Relatively speaking, the stored procedure is more efficient compared to the normal query; therefore we prefer to use a stored procedure to perform this query.

Let's first create our stored procedure `WebSelectCourseSP`.

9.4.3.4.1 Create Stored Procedure `WebSelectCourseSP` Open the Visual Studio .NET 2008 and the Server Explorer window. Click on our sample database folder `CSE_DEPT.mdf` to connect it. Then expand to the `Stored Procedures` folder. To create a new stored procedure, right-click on this folder and select the item `Add New Stored Procedure` to open the `Add New Stored Procedure` dialog box.

Enter the codes shown in Figure 9.55 into this dialog box to create our new stored procedure. Go to `File|Save StoredProcedure1` to save this stored procedure.

To test this stored procedure, go to the Server Explorer window and right-click on this new created stored procedure, and then select the item `Execute` from the pop-up menu to open the `Run Stored Procedure` dialog box. Enter `CSE-438` into the `Value` box in this dialog box as the input `course_id` and click on the `OK` button to run this stored procedure. The running result is displayed in the `Output` window, which is shown in Figure 9.56.

One row is found and returned from the `Course` table in our sample database. To view all returned columns, move the horizontal bar at the bottom of this dialog box to the right. Our stored procedure works fine.

Right-click on our database folder `CSE_DEPT.mdf` and select the item `Close Connection` from the pop-up menu to close this database connection.

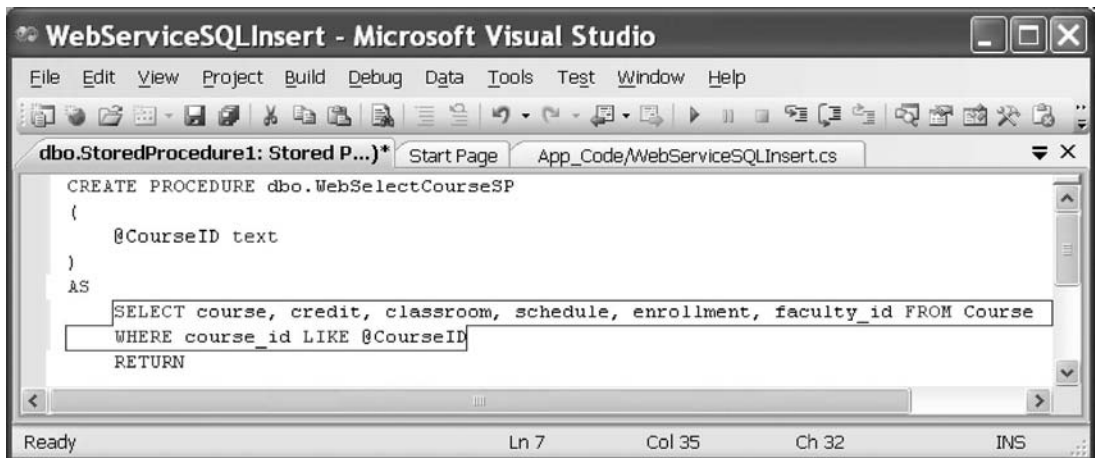


Figure 9.55 Codes for the stored procedure `WebSelectCourseSP`.

```

Output
Show output from: Database Output
Running [dbo].[WebSelectCourseSP] ( @CourseID = CSE-438 ).

course
-----
Advd Logic & Microprocessor
No rows affected.
(1 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[WebSelectCourseSP].

```

Figure 9.56 Running result of the stored procedure WebSelectCourseSP.

```

WebServiceSQLInsert GetSQLInsertCourse()
[WebMethod]
public SQLInsertBase GetSQLInsertCourse(string CourseID)
{
    string cmdString = "dbo.WebSelectCourseSP";
    SqlConnection sqlConnection = new SqlConnection();
    SQLInsertBase GetSQLResult = new SQLInsertBase();
    SqlDataReader sqlReader;

    GetSQLResult.SQLInsertOK = true;
    sqlConnection = SQLConn();
    if (sqlConnection == null)
    {
        GetSQLResult.SQLInsertError = "Database connection is failed";
        ReportError(GetSQLResult);
        return null;
    }
    SqlCommand sqlCommand = new SqlCommand(cmdString, sqlConnection);
    sqlCommand.CommandType = CommandType.StoredProcedure;
    sqlCommand.Parameters.Add("@CourseID", SqlDbType.Text).Value = CourseID;
    sqlReader = sqlCommand.ExecuteReader();
    if (sqlReader.HasRows == true)
        FillCourseDetail(ref GetSQLResult, sqlReader);
    else
    {
        GetSQLResult.SQLInsertError = "No matched course found";
        ReportError(GetSQLResult);
    }
    sqlReader.Close();
    sqlConnection.Close();
    sqlCommand.Dispose();
    return GetSQLResult;
}

```

Figure 9.57 Codes for the Web method GetSQLInsertCourse.

9.4.3.4.2 Develop Codes to Call This Stored Procedure Now let's develop the codes for our fourth Web method GetSQLInsertCourse() to call this stored procedure to perform the course information query. Open the code-behind page `WebServiceSQLInsert.cs` and add the codes shown in Figure 9.57 into this page to create this Web method.

Let's take a look at the codes in this Web method to see how they work.

- A.** The name of the Web method is `GetSQLInsertCourse()`, and it returns an instance of our base class `SQLInsertBase`. The returned instance contains the detailed course information.
- B.** The content of the query string is the name of the stored procedure we developed in the last section. This is required if a stored procedure is used and called later to perform a data query. This name must be exactly identical with the name of the stored procedure we developed, otherwise a running error may be encountered since the stored procedure is identified by its name when the project runs.
- C.** Some data objects such as the `Connection` and the `DataReader` are created here. Also a returned instance of our base class is also created.
- D.** The user-defined `SQLConn()` method is called to perform the database connection. A warning message is displayed and reported using the `ReportError()` method if any error is encountered during the database connection process.
- E.** The `Command` object is created with two arguments: query string and connection object. The coding load can be reduced but the working load cannot when creating a `Command` object in this way. As you know, the `Command` class has four kinds of constructors and we used the third one here.
- F.** The `CommandType` property of the `Command` object must be set to the value of `StoredProcedure` since we need to call a stored procedure to perform this course information query in this method.
- G.** The dynamic parameter `@CourseID` is assigned with the actual parameter `CourseID` that will be entered as an input by the user as the project runs. One point to note is that the nominal name of the this dynamic parameter must be identical with the name of the input parameter defined in the stored procedure we developed in the last section.
- H.** After the `Command` object is initialized, the `ExecuteReader()` method is called to trigger the `DataReader` and to run the stored procedure to perform the course information query. The returned course information is stored to the `DataReader`.
- I.** By checking the `HasRows` property of the `DataReader`, we can determine whether the course information query is successful or not. If this property is `true`, which means that at least one row has been found and returned from our database, the user-defined `FillCourseDetail()` method, whose codes are shown in Figure 9.58, is executed to assign

WebServiceSQLInsert	FillCourseDetail()
<pre> A protected void FillCourseDetail(ref SQLInsertBase sResult, SqlDataReader sReader) B { C sReader.Read(); sResult.FacultyID = Convert.ToString(sReader["faculty_id"]); sResult.Course = Convert.ToString(sReader["course"]); sResult.Schedule = Convert.ToString(sReader["schedule"]); sResult.Classroom = Convert.ToString(sReader["classroom"]); sResult.Credit = Convert.ToInt32(sReader["credit"]); sResult.Enrollment = Convert.ToInt32(sReader["enrollment"]); } </pre>	

Figure 9.58 Codes for the `FillCourseDetail` method.

each piece of course information to the associated member data defined in our base class, and an instance of this class will be returned as this method is done.

- J.** Otherwise, if this property returns `false`, which means that no row has been selected and returned from our database, a warning message is displayed and reported using the `ReportError()` method.
- K.** A cleaning job is performed to release all data objects used in this Web method.
- L.** Finally an instance of our base class `SQLInsertBase`, `GetSQLResult` that contains the queried course detailed information, is returned to the calling procedure.

The codes for the `FillCourseDetail()` method are shown in Figure 9.58.

Let's take a closer look at this piece of code to see how it works.

- A.** Two arguments are passed into this method: The first one is our returned object, which contains all member data, and the second one is the `DataReader`, which contains queried course information. The point is that the passing mode for the first argument is passing by reference, which means that an address of our returned object is passed into this method. In this way, all modified member data that contain the course information in this returned object can be returned to the calling procedure or our Web method—`GetSQLInsertCourse()`. From this point of view, this method works just as a function, and our object can be returned as this method is completed.
- B.** The `Read()` method of the `DataReader` is executed to read course record from the `DataReader`.
- C.** Each column of queried course record is assigned to the associated member data in our base class. Two system methods `Convert.ToString()` and `Convert.ToInt32()` are used to convert all data to the associated data types for this assignment.

Now let's build and run our project to test this Web method. Click on the **Build/Build Web Site** menu item to build this project. Then click on the **Start Debugging** button to run the project. Select our Web method `GetSQLInsertCourse` from the built-in Web interface and enter `CSE-665` as the `course_id` into the **Value** box, and then click on the **Invoke** button to run this Web method. The running result of this Web method is shown in Figure 9.59.

Six pieces of course information are displayed in XML tags except the `course_id`. We defined this member data as a string array `CourseID[]` with a dimension of 10. This member data is used for our second Web method—`GetSQLInsert()`—that returns an array that contains all `course_id`. Since we did not use it in this method, 10 elements on this `CourseID[]` array are set to `true` and displayed in this resulting file.

Click on the **Close** button located at the upper-right corner of this Web interface to terminate our service.

At this point, we have finished developing jobs in our Web Service project on the server side. In the following sections, we want to develop some professional Windows-based and Web-based applications with beautiful graphic user interfaces to use the Web Service application we developed in this section. Those Windows-based and Web-based applications can be considered as Web Service clients.

A complete Web Service project `WebServiceSQLInsert` that contains all four Web methods can be found at the folder `DBProjects\Chapter 9` located at the accompanying ftp site (see Chapter 1).

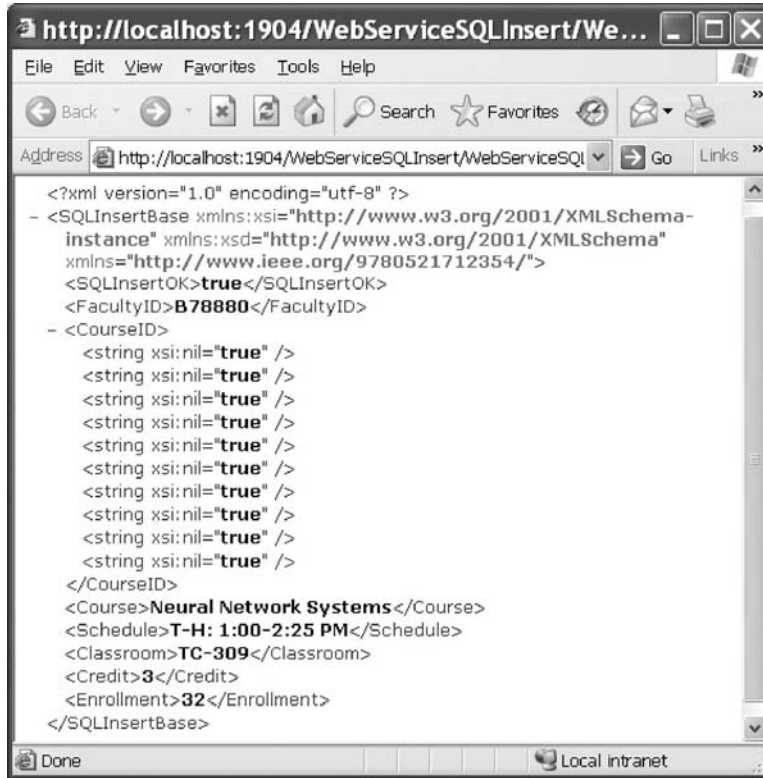


Figure 9.59 Running result of our Web method—GetSQLInsertCourse.

9.4.4 Build Windows-Based Web Service Clients to Use Web Services

To use the Web Service `WebServiceSQLInsert` we developed in the last section, we need first to create a Web Service proxy class in our Windows-based or Web-based applications. Then we can create a new instance of the Web Service proxy class and execute the desired Web methods located in that Web Service class to perform the data insertion actions against our sample database via the Web server. The process of creating a Web Service proxy class is equivalent to adding a Web reference to our Windows-based or Web-based applications. We provided a detailed discussion on how to create a Web Service proxy class in Section 9.3.10.1. Refer to that Section to get more detailed information in creating a proxy class. In order to save space, Sections 9.4.4.1 to 9.4.4.3.4, which provide a detailed discussion in how to build a Windows-based client project `WinClientSQLInsert` to consume our Web Service project `WebServiceSQLInsert`, have been moved to the accompanying ftp site with a file named `WinClientSQLInsert.pdf` that can be found from the folder `DBProjects\Chapter 9\Doc` that is located at the site `ftp://ftp.wiley.com/public/sci_tech_med/practical_database`. For your convenience, a completed Windows-based client project, `WinClientSQLInsert`, has also been developed and debugged, which can be found from the folder `DBProjects\Chapter 9` at the same ftp site.

9.4.5 Build Web-Based Web Service Clients to Use Web Services

As we discussed in Section 9.3.11, there is no significant difference between developing a Web-based client application and a Windows-based client project to use a Web Service. As long as the Web Service is referenced by the Web-based client project, one can access and call any Web method developed in that Web Service to perform the desired data queries via the Web-based client project without any problem. Visual Studio.NET will create the same document files such as the Discovery Map file, the WDSL file, and the DISCO file for the client project no matter if this Web Service is used by a Windows-based or a Web-based client application.

To save time and space, we can modify an existing ASP.NET Web application SQLWebInsert we developed in Chapter 8 to make it our new Web-based Web Service client project named WebClientSQLInsert; that is, we can copy and rename that entire project as our new Web-based client project. However, we prefer to create a new ASP.NET Web site project and only copy and modify the Course page.

This section can be developed in the following sequences:

1. Create a new ASP.NET website project WebClientSQLInsert and add an existing website page `Course.aspx` from the project SQLWebInsert into our new project with a few modifications.
2. Add a Web Service reference to our new project and modify the Web form window of the `Course.aspx` to meet our data insertion requirements.
3. Modify codes in the related methods of the `Course.aspx.cs` file to call the associated Web method to perform our data insertion. The code modifications include the following sections:
 - a. Modify the codes in the `Page_Load()` method of the `CourseForm` class.
 - b. Develop the codes for the Insert button's Click method.
 - c. Develop the codes for the TextChanged event method of the Course ID textbox.
 - d. Modify the codes in the Select button's Click method. Also add four user-defined methods: `ProcessObject()`, `FillCourseListBox()`, `FillCourseDataSet()`, and `FillCourseDetail()`. These four methods are basically identical with those we developed in the last Windows-based Web Service client project WinClientSQLInsert. One can copy and paste them into our new project with a few modifications.
 - e. Modify the codes in the SelectedIndexChanged event method.
 - f. Modify the codes in the Back button's Click method.

Now let's start with the first step listed above.

9.4.5.1 Create a New Website Project and Add Existing Web Page

Open Visual Studio.NET and go to the File|New Web Site menu item to create a new website project. Enter `C:\Chapter 9\WebClientSQLInsert` into the name box that is next to the Location box, and click on OK to create this new project.

On the opened new project window, right-click on our new project icon WebClientSQLInsert from the Solution Explorer window, and select the item Add Existing Item from the pop-up menu to open the Add Existing Item dialog. Browse to

our Web project SQLWebInsert located at the folder DBProjects\Chapter 8 at the accompanying ftp site (see Chapter 1). Select this project and then click on the Add button to open all existing items for this website project. Select both items, *Course.aspx* and *Course.aspx.cs*, from the list and click on the Add button to add these two items into our new website project.

9.4.5.2 Add Web Service Reference and Modify Web Form Window

To add a Web reference of our Web Service to this new website project, right-click on our new project icon from the Solution Explorer window and select the item **Add Web Reference** from the pop-up menu. Open our Web Service project WebServiceSQLInsert and click on the Start Debugging button to run it. As the project runs, copy the URL from the Address box and paste it into the URL box in our **Add Web Reference** dialog box. Then click on the green button **Go** to add this Web Service as a reference to our client project. You can modify this Web reference name to any name you want. In this application, we prefer to change it to *WS_SQLInsert*. Your finished **Add Web Reference** dialog box should match the one shown in Figure 9.60.

Click on the **Add Reference** button to finish this adding Web reference process. Immediately you can see that the following three files are created in the Solution Explorer window under the folder of the new added Web reference:

- WebServiceSQLInsert.disco
- WebServiceSQLInsert.discomap
- WebServiceSQLInsert.wsdl

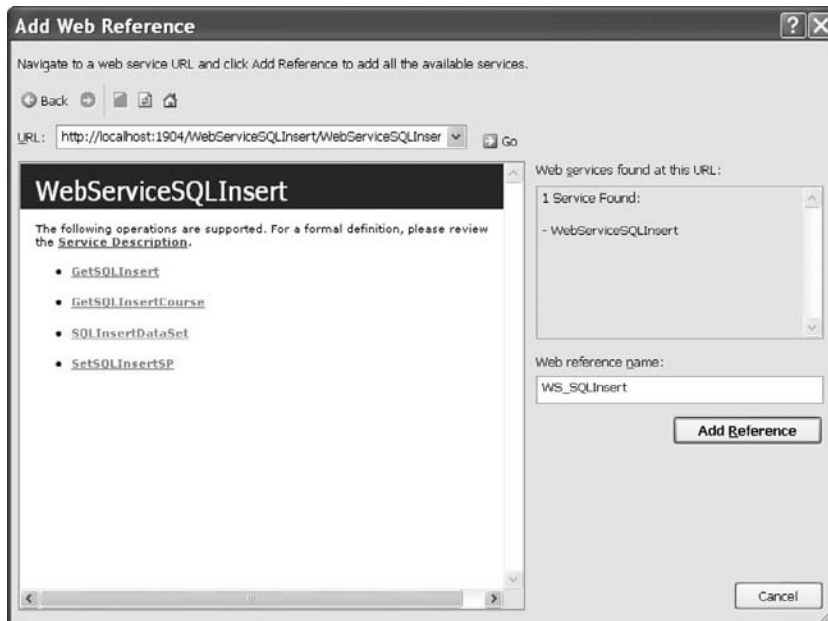


Figure 9.60 Finished Add Web Reference dialog box.

The modifications to the Web page of the `Course.aspx` include three steps:

1. Add a new textbox and name it `txtCourseID` and a new label with the `Text` property as `Course ID`. The positions of this textbox and label are above the `Course Title` textbox. Also set the `AutoPostBack` property of this textbox to `true`. **This is very important** since when the content of this textbox is changed when the project runs, a `TextChanged` event occurs. However, this event only occurs on the client side, not the server side. Our Web-based client project is running at a Web server or server side; therefore, this event cannot be responded to by the server. The result is that the command inside this event method cannot be executed (the `Insert` button cannot be enabled); even the content of the `Course ID` textbox is changed when the project runs. To solve this problem, we must set the `AutoPostBack` property of this textbox to `true` to allow it post back a `TextChanged` event to the client automatically as the content of this textbox is changed.
2. Add one more `DropDownList` control and the associated label to the left of the `Faculty Name` combobox control. Name this `DropDownList` `ComboMethod` and the label with the `Text` property as `Method`. This `DropDownList` control is used to store two Web methods developed in our Web Service and enable users to select one of them to perform the associated data insertion as the project runs.
3. Change the names of the following textboxes to make them match the names used in the codes on this page:
 - a. `Course Title` textbox from `txtCourse` to `txtCourseName`
 - b. `Credit` textbox from `txtCredit` to `txtCredits`
 - c. `Enrollment` textbox from `txtEnrollment` to `txtEnroll`
 - d. `Classroom` textbox from `txtClassroom` to `txtClassRoom`

Your modified `Course.aspx` Web form window is shown in Figure 9.61. Go to the `FileSave All` menu item to save these modifications.

9.4.5.3 Modify Codes for Related Methods

The first modification is to change the codes in the `Page_Load()` method and some field-level variables.

9.4.5.3.1 Modify Codes in Page_Load Method Perform the following changes to complete this modification:

1. Remove the field-level textbox array variables `CourseTextBox[]` since we do not need it in this application.
2. Add the following three field-level variables into this page:


```
private bool dsFlag = false;
private DataSet wsDataSet = new DataSet();
WS_SQLInsert.SQLInsertBase wsSQLResult = new WS_SQLInsert.
    SQLInsertBase();
```
3. Remove the `if` block inside the `Page_Load()` method and the associated global connection object that is stored in the Application state function `Application["sqlConnection"]`.
4. Add the codes to display two Web methods in the combobox control `ComboMethod`.

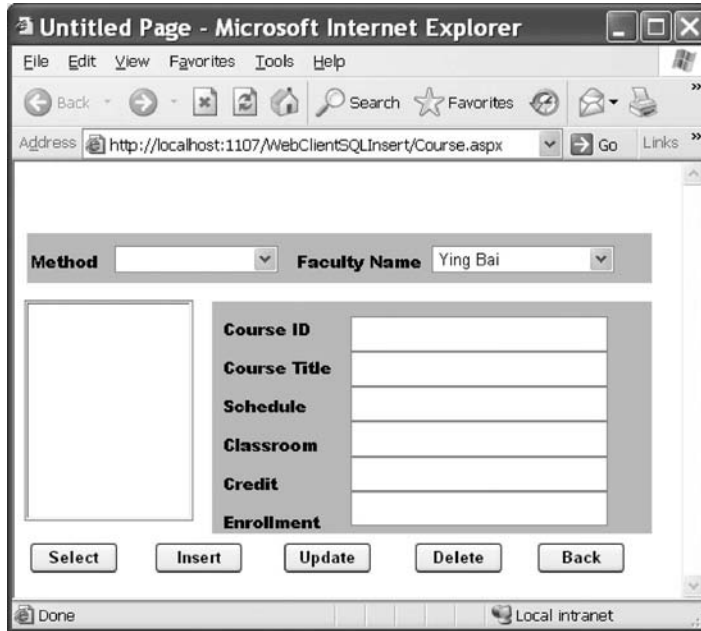


Figure 9.61 Modified Course page window.

Your finished codes for the Page_Load() method should match the one shown in Figure 9.62. The new adding codes have been highlighted in bold. The next step is to develop the codes for the Insert button's Click method.

9.4.5.3.2 Develop Codes for Insert Button's Click Method The function of this piece of code is to insert a new course record that is stored in six textboxes in the Web page into the database as this Insert button is clicked. This coding is basically identical with that in the same method of the Windows-based client project we developed in the last section. Therefore we can copy those codes, from that method and paste them into our current method with a few modifications.

Open our Windows-based client project WinClientSQLInsert from the folder DBProjects\Chapter 9 at the accompanying ftp site (see Chapter 1) and browse to the Insert button's Click method. Then copy all codes from that method and paste them into the Insert button's Click method in our current Web-based client project.

The only modification to this method is to add one more string variable `errMsg` that is used to store the returned error information from calling different Web methods. Also all message box functions `MessageBox()` should be replaced by the `Write()` method of the `Response` object of the server class since the `MessageBox()` method can only be used in the Windows-based applications.

Your finished codes for the Insert button's Click method should match the one shown in Figure 9.63. The modification parts have been highlighted in bold.

Course	Page_Load()
<pre> public partial class Course : System.Web.UI.Page { private bool dsFlag = false; private DataSet wsDataSet = new DataSet(); WS_SQLInsert.SQLInsertBase wsSQLResult = new WS_SQLInsert.SQLInsertBase(); protected void Page_Load(object sender, EventArgs e) { if (!IsPostBack) //these items can only be added into the combo box in one time { ComboName.Items.Add("Ying Bai"); ComboName.Items.Add("Satish Bhalla"); ComboName.Items.Add("Black Anderson"); ComboName.Items.Add("Steve Johnson"); ComboName.Items.Add("Jenney King"); ComboName.Items.Add("Alice Brown"); ComboName.Items.Add("Debby Angles"); ComboName.Items.Add("Jeff Henry"); ComboMethod.Items.Add("Stored Procedure Method"); ComboMethod.Items.Add("DataSet Method"); } } } </pre>	

Figure 9.62 Modified Page_Load method.

Course	cmdInsert_Click()
<pre> protected void cmdInsert_Click(object sender, EventArgs e) { WS_SQLInsert.WebServiceSQLInsert wsSQLInsert = new WS_SQLInsert.WebServiceSQLInsert(); string errMsg; if (ComboMethod.Text == "Stored Procedure Method") { try { wsSQLResult = wsSQLInsert.SetSQLInsertSP(ComboName.Text, txtCourseID.Text, txtCourseName.Text, txtSchedule.Text, txtClassRoom.Text, Convert.ToInt32(txtCredits.Text), Convert.ToInt32(txtEnroll.Text)); } catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert(" + errMsg + ")</script>"); } if (wsSQLResult.SQLInsertOK == false) Response.Write("<script>alert(" + wsSQLResult.SQLInsertError + ")</script>"); } else { dsFlag = true; Application["dsFlag"] = dsFlag; //indicate the DataSet insert is performed //reserve this flag as a global flag try { wsDataSet = wsSQLInsert.SQLInsertDataSet(ComboName.Text, txtCourseID.Text, txtCourseName.Text, txtSchedule.Text, txtClassRoom.Text, Convert.ToInt32(txtCredits.Text), Convert.ToInt32(txtEnroll.Text)); } catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert(" + errMsg + ")</script>"); } } Application["wsDataSet"] = wsDataSet; //reserve the global DataSet cmdInsert.Enabled = false; } </pre>	

Figure 9.63 Coding for the Insert button's Click method.

Let's take a quick review of this piece of code to see how it works.

- A. A string variable `errMsg` is created, and this variable is used to store the possible error message to be displayed via a Java `alert()` method if any error occurs during the data insertion action processing.
- B. If the user selected the Stored Procedure method to perform this data insertion, the Web method `SetSQLInsertSP()`, which is developed in the Web Service, is executed to call the associated stored procedure to insert a new course record into our sample database. Any error encountered during the execution of this Web method will be displayed and reported.
- C. If the user chose the DataSet method to perform this data insertion, we need to set a flag to tell the project that a DataSet data insertion has been performed.
- D. This flag is stored into a global variable using the Application state function. The reason we need to make this step is that the Web method `SQLInsertDataSet()` has two functions: insert data into the database and retrieve data from the database. However, in order to perform data retrieval using this method, first we must insert data using this method. Otherwise no data can be retrieved if no data insertion is performed using this DataSet method. The reason we use an Application state to store this flag is that our Web client project will run on a Web server, and the server will send back a refreshed page to the client each time a request is sent to the server. Therefore all global variable values will also be refreshed when a refreshed page is sent back. However, the Application state is never changed, no matter how many times our client page is refreshed.
- E. The associated Web method `SQLInsertDataSet()` is called to insert this new course record into the database. Similarly, if any error is encountered during this calling process, it will be displayed and reported immediately.
- F. The returned DataSet object `wsDataSet` that contains all `course_ids` is a field-level variable. Because of the same reason we discussed in step D, we need to use an Application state to store this DataSet since we need to pick up all `course_ids` from it when we perform the validation process later by clicking the Select button. Otherwise the content of this DataSet will be refreshed each time when a refreshed Course page is sent back.
- G. Finally the Insert button is disabled to avoid multiple insertion of the same data into the database.

9.4.5.3.3 Develop Codes for TextChanged Method of CourseID Textbox The codes for this event method are very simple. Open this method by double-clicking on the Course ID textbox from the Web page window and enter the following code into this method:

```
cmdInsert.Enabled = true;
```

As we mentioned, after a new course record has been inserted into the database, the Insert button must be disabled to avoid the possible multiple insertion of the same record into the database. However, as the next new course record is ready to be inserted into the database, this Insert button should be enabled to allow users to do that insertion. To distinguish between the existing and a new course record, the content of the Course ID textbox or the `course_id` column is a good candidate since it is a primary key in our Course data table. Each `course_id` is a unique identifier for each course record, and therefore as long as the content of this Course ID textbox changed, which means that a new course record is ready to be inserted, the Insert button should be enabled for this situation.

Another important point is to make sure that the `AutoPostBack` property of this Course ID textbox should be set to `true` to allow it to post back a `TextChanged` event to the client when its content is changed.

9.4.5.3.4 Modify Codes in Select Button's Click Method The codes in this method are similar to those codes, we developed in the same method in our Windows-based client project `WinClientSQLInsert`. Therefore we can copy those codes, and paste them into our current Select button's Click method with a few modifications. Open the Select button's Click method from our Windows-based client project `WinClientSQLInsert`, copy those codes and paste them into our Select button's Click method. The only modifications to this piece of copied codes are to change the Windows-based message box function `MessageBox()` to the Web-based message box function. Your finished codes for this method are shown in Figure 9.64. The modified parts have been highlighted in bold.

Let's take a quick review for this piece of code to see how it works.

- A. An instance of our Web Service reference `WebServiceSQLInsert` is created first, and this instance works as a bridge to connect this client project with associated Web methods developed in the Web Service. Also an `errMsg` string variable is created, and it is used to store the error message to be displayed and reported later.
- B. If the stored procedure method is selected by the user, the associated Web method `GetSQLInsert()` is executed to call the stored procedure to pick up all `course_ids` taught by the selected faculty based on the input faculty name. If any error had occurred during the execution of this Web method, the error source is reported and displayed with a `Java alert()` script method.

```

Course ▼ cmdSelect_Click() ▼
protected void cmdSelect_Click(object sender, EventArgs e)
{
A   WS_SQLInsert.WebServiceSQLInsert wsSQLInsert = new WS_SQLInsert.WebServiceSQLInsert();
   string errMsg;
B   if (ComboMethod.Text == "Stored Procedure Method")
   {
   try { wsSQLResult = wsSQLInsert.GetSQLInsert(ComboName.Text); }
   catch (Exception err) { errMsg = "Web service is wrong: " + err.Message;
   Response.Write("<script>alert('" + errMsg + "')</script>"); }
C   if (wsSQLResult.SQLInsertOK == false)
   Response.Write("<script>alert('" + wsSQLResult.SQLInsertError + "')</script>");
D   ProcessObject(ref wsSQLResult);
E   }
   else
   {
   dsFlag = (bool)Application["dsFlag"];
   if (dsFlag == false)
   {
   errMsg = "No DataSet Insertion performed, do that first";
   Response.Write("<script>alert('" + errMsg + "')</script>");
   }
   wsDataSet = (DataSet)Application["wsDataSet"];
F   FillCourseDataSet(ref wsDataSet);
G   Application["dsFlag"] = false;
   }
}

```

Figure 9.64 Codes for the Select button Click method.

- C. Besides the system error checking, we also need to inspect any application error, and this can be performed by checking the status of the member data `SQLInsertOK` that is defined in the base class `SQLInsertBase` in our Web Service project.
- D. If no error is detected, the user-defined `ProcessObject()` method, whose detailed codes are shown in Figure 9.65, is called to extract all retrieved `course_ids` from the returned instance and add them into the listbox control `CourseList` in our Course page window.
- E. If a user selected the `DataSet` method, first we need to check the `dsFlag` stored in an Application state to make sure that the Web method `SQLInsertDataSet()` has been performed once since our current data query needs to extract all `course_ids` from the `DataSet` that is returned from the last execution of the Web method `SQLInsertDataSet()`. If this `dsFlag` is `false`, which means that this Web method has not been called and executed, we do not have any returned `DataSet` available. A warning message is displayed to remind users to run the Web method `SQLInsertDataSet()` first.
- F. If the `dsFlag` is `true`, the Web method `SQLInsertDataSet()` has been executed and a returned `DataSet` that contains all `course_ids` is available. A user-defined `FillCourseDataSet()` method is executed to extract all `course_ids` from that returned `DataSet` from the last calling of the Web method and add them into the listbox control in our client page window. The global `DataSet` object `wsDataSet` that is stored in an Application state is passed as an argument for this method calling.
- G. Finally the `dsFlag` stored in an Application state function is reset to `false`.

The detailed codes for two user-defined methods `ProcessObject()` and `FillCourseListBox()` are shown in Figure 9.65.

Let's take a closer look at this piece of code to see how it works.

- A. A local string variable `errMsg` is declared, and it is used to hold any error message to be displayed and reported later.

```

private void ProcessObject(ref WS_SQLInsert.SQLInsertBase wsResult)
{
  A   string errMsg;
  B   if (wsResult.SQLInsertOK == true)
      FillCourseListBox(ref wsResult);
  C   else
      {
        errMsg = "Course information cannot be retrieved: " + wsResult.SQLInsertError;
        Response.Write("<script>alert('" + errMsg + "')</script>");
      }
}

private void FillCourseListBox(ref WS_SQLInsert.SQLInsertBase sqlResult)
{
  D   int index = 0;
  E   CourseList.Items.Clear();           //clean up the course listbox
  F   for (index = 0; index <= sqlResult.CourseID.Length - 1; index++)
      {
        if (sqlResult.CourseID[index] != null)
            CourseList.Items.Add(sqlResult.CourseID[index]);
      }
}

```

Figure 9.65 Codes for the `ProcessObject` and `FillCourseListBox` methods.

- B.** First, we need to check the member data SQLInsertOK to make sure that the Web method is executed successfully. If it is, the user-defined method FillCourseListBox() is called to fill all `course_ids` contained in the returned instance to the listbox control in our client page.
- C.** A warning message is displayed if any error was encountered during the execution of that Web method.
- D.** In the FillCourseListBox() method, first a local integer variable `index` is created, and it works as a loop counter for a `for` loop to continuously pick up all `course_ids` from the returned instance and add them into the listbox control.
- E.** The course listbox control is cleaned up first before any `course_id` can be added into it. This process is very important in displaying all `course_ids`, otherwise any new `course_id` will be attached at the end of the original `course_id` in this control and the displaying result is messy.
- F.** A `for` loop is used to continuously pick up the `course_id` from the `CourseID[]` array defined in our base class `SQLInsertBase`. Note the upper bound and the length of this array. The length or the size of this array is 10, but the upper bound of this array is 9 since the index of this array starts from 0, not 1. Therefore the upper bound of this array is equal to the length of this array minus 1. As long as the content of the `CourseID[index]` is not null, that element or `course_id` is added to the listbox control by using the `Add()` method.

The codes for the user-defined `FillCourseDataSet()` method are shown in Figure 9.66. This coding is identical with that coding in the same method we developed in our Windows-based client project `WinClientSQLInsert`. You can copy and paste it into our current project.

Let's take a look at the codes in this subroutine to see how they work.

- A.** A `DataTable` object is declared at the beginning of this method since we need to use it to perform the data extraction from the returned instance and the data addition to the listbox control later.
- B.** The listbox control `CourseList` is first cleaned up to avoid messy displaying of multiple `course_ids`.
- C.** The `CourseTable` object is initialized by adding a new data table named `Course` and is attached to the `DataSet` object `ds`.

Course	FillCourseDataSet()
<pre> private void FillCourseDataSet(ref DataSet ds) { DataTable CourseTable = new DataTable(); CourseList.Items.Clear(); //clean up the course listbox CourseTable = ds.Tables["Course"]; foreach (DataRow CourseRow in CourseTable.Rows) { CourseList.Items.Add(CourseRow[0].ToString()); //the 1st column is course_id } } </pre>	

Figure 9.66 Codes for the `FillCourseDataSet` method.

- D.** A `foreach` loop is used to continuously pick up the first column that is the `course_id` column from all returned rows, and add each of them into the list box control. One point to be noticed is that the first column has an index value of 0, not 1, since the index starts from 0.

Next we need to modify the codes in the `SelectedIndexChanged` event method and add the fourth method, `FillCourseDetail()`. Before we can continue to do these jobs, first we need to delete the following methods from our current project since we do not need to use them in this application:

- `FillCourseReader()`
- `FillCourseReaderTextBox()`
- `MapCourseTable()`

Now let's continue to go to the next step.

9.4.5.3.5 Modify Codes in SelectedIndexChanged Method The function of this coding is that the detailed course information such as the course name, schedule, classroom, credit, and enrollment will be displayed in the associated textbox control as the user clicked on and selected one `course_id` from the listbox control; that is, the main coding job is performed inside the `SelectedIndexChanged` event method of the listbox control. Because as users click on or select a `course_id` from the listbox control, a `SelectedIndexChanged` event is issued, and this event is passed to the associated `SelectedIndexChanged` event method to be processed.

To pick up the detailed course information for a selected course, the Web method `GetSQLInsertCourse()` we developed in our Web Service project `WebServiceSQLInsert` is called, and this method returns an instance of the base class `SQLInsertBase` to our client project. The detailed course information is stored in that returned instance.

The codes in this event method are identical with that we did for the same event method in our Windows-based client project `WinClientSQLInsert`. Therefore we can copy those codes from that method and paste them into our current project with a few modifications.

Double-click on the listbox control `CourseList` from our client page window to open the `SelectedIndexChanged` event method of the listbox control. Then copy and paste those codes from the same event method in our Windows-based project `WinClientSQLInsert`. The only modifications are those `MessageBox()` methods, and these methods should be replaced by a Java script message method `alert()`. Your finished `SelectedIndexChanged` event method should match that shown in Figure 9.67. The modified parts have been highlighted in bold.

Let's take a closer look at this piece of code to see how it works.

- A.** An instance of our Web Service reference or the proxy class `wsSQLInsert` is created here. This instance works as a bridge between our client project and the Web methods developed in our Web Service project. Also a local string variable `errMsg` is declared, and it is used to hold the error message to be displayed and reported later.
- B.** A `try...catch` block is used to call the Web method `GetSQLInsertCourse()` with the selected `course_id` from the listbox control as the argument to perform this course information retrieving. The selected `course_id` is stored in the `Text` property of the

Course	CourseList_SelectedIndexChanged()
<p>A</p> <p>B</p> <p>C</p> <p>D</p>	<pre>protected void CourseList_SelectedIndexChanged(object sender, EventArgs e) { WS_SQLInsert.WebServiceSQLInsert wsSQLInsert = new WS_SQLInsert.WebServiceSQLInsert(); string errMsg; try { wsSQLResult = wsSQLInsert.GetSQLInsertCourse(CourseList.Text); } catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert('" + errMsg + "')</script>"); } if (wsSQLResult.SQLInsertOK == false) Response.Write("<script>alert('" + wsSQLResult.SQLInsertError + "')</script>"); FillCourseDetail(ref wsSQLResult); }</pre>

Figure 9.67 Modified codes for the SelectedIndexChanged method.

Course	FillCourseDetail()
	<pre>private void FillCourseDetail(ref WS_SQLInsert.SQLInsertBase sqlResult) { txtCourseID.Text = CourseList.Text; txtCourseName.Text = sqlResult.Course; txtSchedule.Text = sqlResult.Schedule; txtClassRoom.Text = sqlResult.Classroom; txtCredits.Text = sqlResult.Credit.ToString(); txtEnroll.Text = sqlResult.Enrollment.ToString(); }</pre>

Figure 9.68 Codes for the FillCourseDetail method.

CourseList control. An exception message is displayed if any error has encountered during the execution of this Web method.

- C. In addition to the error checking performed by the system, we also need to perform our exception checking by inspecting the member data SQLInsertOK in the base class SQLInsertBase. If this data value is false, which means that an application error has occurred during the running of this Web method, an error message is displayed to indicate this situation.
- D. If everything is fine, the user-defined method FillCourseDetail() is executed to extract the detailed course information from the returned instance and assign it to each associated textbox control in our client page form.

The detailed codes for the FillCourseDetail() method are shown in Figure 9.68.

This piece of code is identical with that developed in the same method in our Windows-based client project WinClientSQLInsert. You can copy and paste it into this project.

The function of this coding is straightforward and without tricks. Each piece of course data is extracted from the returned instance and assigned to the associated textbox control in our client page window.

9.4.5.3.6 Modify Codes in Back Button's Click Method The final modification is to change the codes for the Back button's Click method. When this button is clicked on

by the user, our client project should be terminated. Open this method and replace the original codes with the following codes to this method to close our client project:

```
Response.Write("<script>window.close()</script>");
```

In this way, our client page will be terminated when the script command close() is executed.

At this point, we have finished all the coding operations for this Web-based client project. Before we can run this project to test the data insertion and validation functions, make sure that the following operations have been performed:

- Our main Web page `Course.aspx` has been set as the starting page. This can be done by right-clicking on our main Web page and selecting the item `Set As Start Page` from the pop-up menu.
- Our Web Service `WebServiceSQLInsert` is in the open status, and this can be checked by locating a small white icon on the status bar at the bottom of the screen. If you cannot find this icon, open our Web Service project and click on the `Start Debugging` button to run it. After the Web Service starts to run, you can close it if you like, but it is still in the open status.
- Two new course records, `CSE-535` and `CSE-526`, which we inserted before by testing the `Insert` button's `Click` method, should have been deleted from our sample database since we want to insert the same course records in this test.

Now click on the `Start Debugging` button to run our client project. First, let's test the data insertion function. Select the `Stored Procedure` method from the `Method` combobox control, select the default faculty `Ying Bai` from the `Faculty Name` combo box control, and enter the first new course information as shown in Table 9.2 into the associated textboxes, and then click on the `Insert` button. Perform the similar operation to insert the second new course information as shown in Table 9.3 with the `DataSet` method selected. Your running Web page should match the one shown in Figure 9.69.

Table 9.2 The first course record to be inserted

Controls	Input Parameters
Method:	Stored Procedure Method
Faculty Name:	Ying Bai
Course ID:	CSE-535
Course Name:	Dynamic Modeling Systems
Schedule:	T-H: 11:00-12:25 PM
Classroom:	TC-325
Credits:	3
Enrollment:	25

Table 9.3 The second course record to be inserted

Controls	Input Parameters
Method:	DataSet Method
Faculty Name:	Ying Bai
Course ID:	CSE-526
Course Name:	Embedded Microcontrollers
Schedule:	M-W-F: 9:00-9:55 AM
Classroom:	TC-308
Credits:	3
Enrollment:	32

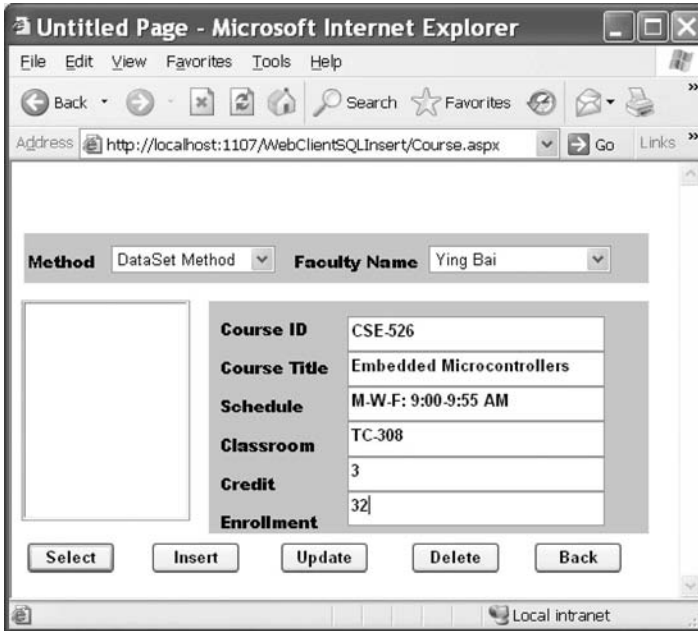


Figure 9.69 Running status of inserting new course records.

To validate these data insertions, click on the Select button for DataSet method and then the Stored Procedure method, respectively. The running result is shown in Figure 9.70. You can find that our two new inserted courses CSE-535 and CSE-526 have been added into and retrieved back from our database and displayed in the listbox control.

To get detailed course information for a specific course, click on a desired `course_id` from the listbox control. Immediately the detailed course information for the selected `course_id` is displayed on each associated textbox, which is shown in Figure 9.71.

You can try to get the detailed information for different course by selecting different `course_id` from the listbox control via either the DataSet or the Stored Procedure method. Click on the Back button to terminate our Web client project when you finish this test.

A completed Web-based Web Service client project WebClientSQLInsert can be found in the folder DBProjects\Chapter 9 located at the accompanying ftp site (see Chapter 1).

Next we need to take care of updating and deleting data via Web services.

9.5 BUILD ASP.NET WEB SERVICE TO UPDATE AND DELETE DATA FOR SQL SERVER DATABASE

In this section we discuss how to update and delete a record in the Course table in our sample database via the Web Services. Two major Web methods are developed in this Web Service project: `SQLUpdateSP()` and `SQLDeleteSP()`. Both methods call the associated stored procedure to perform the data updating and deleting operations in our sample database CSE_DEPT.mdf.

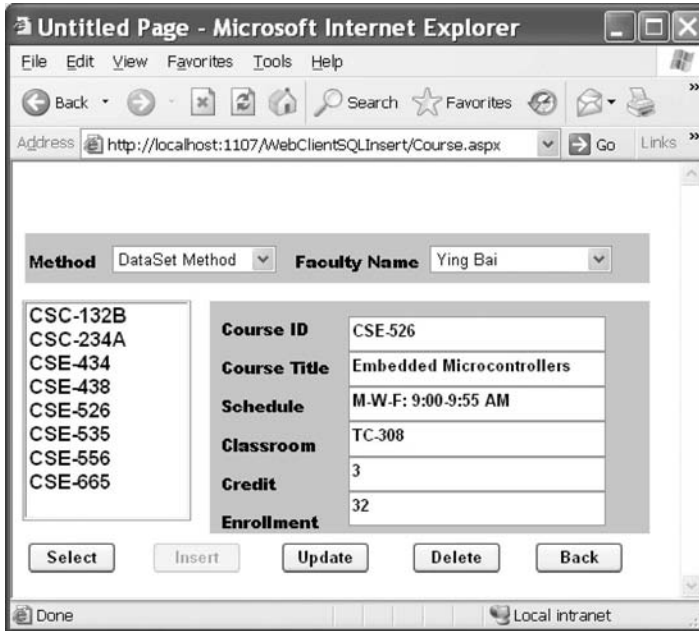


Figure 9.70 Running status of the data validation process.

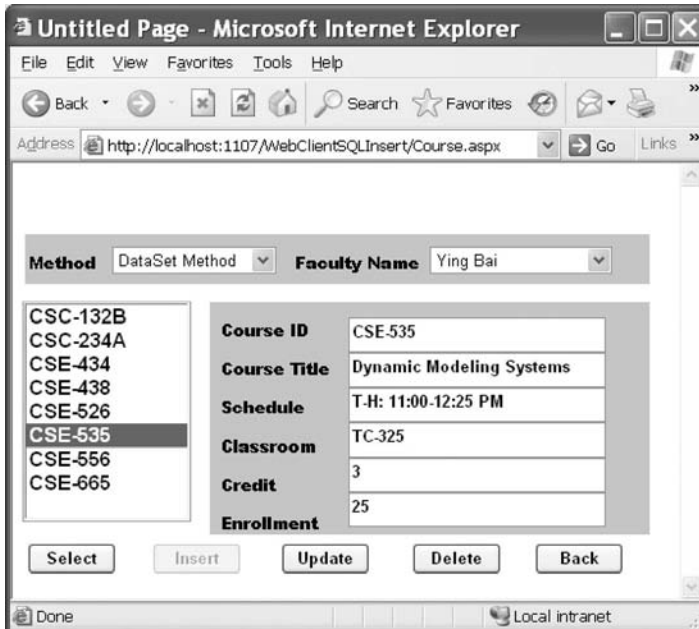


Figure 9.71 Running status of getting the detailed course information.

To save time and space, we can modify an existing Web Service project `WebServiceSQLInsert` we developed in Section 9.4 to make it our new Web Service project `WebServiceSQLUpdateDelete`.

9.5.1 Modify Existing Web Service Project

Open Windows Explorer and create a new folder Chapter 9 if you have not created it. Then browse to the folder `DBProjects\Chapter 9` located at the accompanying ftp site (see Chapter 1), and select the Web Service project `WebServiceSQLInsert`. Copy this project and paste it into our new folder `C:\Chapter 9` on our computer. Rename this project `WebServiceSQLUpdateDelete`.

Open Visual Studio.NET 2008 and our new Web Service project `WebServiceSQLUpdateDelete`. Perform the following modifications to this project:

1. Expand the `App_Code` folder from the Solution Explorer window. Find and rename our base class from `SQLInsertBase.cs` to `SQLBase.cs` by right-clicking on this class file and select the `Rename` item from the pop-up menu.
2. In a similar way, rename our code-behind page from `WebServiceSQLInsert.cs` to `WebServiceSQLUpdateDelete.cs`.
3. Rename our main Web Service page from `WebServiceSQLInsert.asmx` to `WebServiceSQLUpdateDelete.asmx` in a similar way.
4. Double-click on our new Web main page `WebServiceSQLUpdateDelete.asmx` to open it, and make the following changes to the top coding line:
 - a. From: `CodeBehind="~/App_Code/WebServiceSQLInsert.cs"`
To: `CodeBehind="~/App_Code/WebServiceSQLUpdateDelete.cs"`
 - b. From: `Class="WebServiceSQLInsert"`
To: `Class="WebServiceSQLUpdateDelete"`
5. Open Visual Studio.NET 2008 and our new Web Service project. Then double-click on our new base class file `SQLBase` and perform the following modifications to the class name and the first two member data:
 - a. Change the name of the class and the constructor from `SQLInsertBase` to `SQLBase`.
 - b. Change `public bool SQLInsertOK;` to `public bool SQLOK;`
 - c. Change `public string SQLInsertError;` to `public string SQLError;`
6. Double-click on our new code-behind page `WebServiceSQLUpdateDelete.cs` from the Solution Explorer window to open it. Change the name of our Web class, which is located after the access mode `public class`, and the name of the constructor from `WebServiceSQLInsert` to `WebServiceSQLUpdateDelete`.

Go to the `File\Save All` menu item to save these modifications. Next let's concentrate on the modifications to the related Web methods.

9.5.2 Modify Related Web Methods

These modifications include:

1. Remove the Web method `SQLInsertDataSet()` from this project since we do not need this method to perform either data updating or deleting actions.

2. Modify the Web method `SetSQLInsertSP()` to make it our new Web method `SQLUpdateSP()`. This method will call a stored procedure to perform the data updating for our `Course` table.
3. Modify the Web method `GetSQLInsert()` to make it our new Web method `GetSQLCourse()`, which will return all `course_id`, including the original and the updating `course_id`, to the calling procedure. This method will be called by the client project to perform a data updating or deleting validation.
4. Modify the Web method `GetSQLInsertCourse()` to make it as our new Web method `GetSQLCourseDetail()`, which will return detailed information for a specific `course_id` to the calling procedure. This method will be called by the client project to perform a data updating validation.
5. Add a new Web method named `SQLDeleteSP()`, and this method will delete a course record based on the input `course_id`.

Now let's detail these modifications starting from step 2.

9.5.2.1 Modify Web Method from *SetSQLInsertSP* to *SQLUpdateSP*

The function of this Web method is to call a stored procedure `dbo.WebUpdateCourseSP()`, which will be developed in Section 9.5.3.1, to perform the data updating for a course record based on the `course_id`.

Normally we do not need to update the primary key for a record to be updated because it is better to insert a new record with a new primary key than to update that record with a new primary key. Another reason for this issue is that it would be a very complicated process if one wants to update a primary key in a table since it may be related to many other child tables using the foreign keys. Therefore one has to update those foreign keys first in many child tables before the primary key can be updated in the parent table.

Open our new Web Service project `WebServiceSQLUpdateDelete` and the Web method `SetSQLInsertSP()`, and then perform the modifications shown in Figure 9.72 to this method. The modified parts have been highlighted in bold.

Let's take a closer look at these modified codes to see how they work.

- A. The name of this Web method is changed to `SQLUpdateSP`. Also the returned data type is our modified base class whose name is changed to `SQLBase`.
- B. The content of the query string is equal to the name of the stored procedure, which will be developed in Section 9.5.3.1. Keep in mind that this name must be identical with the name used for the stored procedure to be developed later.
- C. An instance of our modified base class `SQLBase`, `SQLResult`, is created, and this instance contains the running status of this Web method and will be returned to the calling procedure.
- D. A local integer variable `intUpdate` is declared here, and it is used to hold the returned value from calling the `ExecuteNonQuery()` method.
- E. First, we preset a good running status of this Web method to the member data `SQLOK` to indicate that so far our Web method is running fine.
- F. If any error is encountered during the database connection process, the error information is stored in the member data `SQLError` and reported using the user-defined `ReportError()` method.

WebServiceSQLUpdateDelete		SQLUpdateSP()
	[WebMethod]	
A	public SQLBase SQLUpdateSP (string FacultyName, string CourseID, string Course, string Schedule, string Classroom, int Credit, int Enroll)	
	{	
B	string cmdString = "dbo.WebUpdateCourseSP";	
	SqlConnection sqlConnection = new SqlConnection();	
C	SQLBase SQLResult = new SQLBase ();	
	SqlCommand sqlCommand = new SqlCommand();	
D	int intUpdate = 0;	
E	SQLResult.SQLOK = true;	
	sqlConnection = SQLConn();	
F	if (sqlConnection == null)	
	{	
	SQLResult.SQLError = "Database connection is failed";	
	ReportError(SQLResult);	
	return null;	
	}	
G	sqlCommand.Connection = sqlConnection;	
	sqlCommand.CommandType = CommandType.StoredProcedure;	
	sqlCommand.CommandText = cmdString;	
	sqlCommand.Parameters.Add("@FacultyName", SqlDbType.Text).Value = FacultyName;	
	sqlCommand.Parameters.Add("@CourseID", SqlDbType.Char).Value = CourseID;	
	sqlCommand.Parameters.Add("@Course", SqlDbType.Text).Value = Course;	
	sqlCommand.Parameters.Add("@Schedule", SqlDbType.Char).Value = Schedule;	
	sqlCommand.Parameters.Add("@Classroom", SqlDbType.Text).Value = Classroom;	
	sqlCommand.Parameters.Add("@Credit", SqlDbType.Int).Value = Credit;	
	sqlCommand.Parameters.Add("@Enroll", SqlDbType.Int).Value = Enroll;	
H	intUpdate = sqlCommand.ExecuteNonQuery();	
I	sqlConnection.Close();	
	sqlCommand.Dispose();	
J	if (intUpdate == 0)	
	{	
	SQLResult.SQLError = "Data updating is failed";	
	ReportError(SQLResult);	
	}	
K	return SQLResult ;	
	}	

Figure 9.72 Modified codes for the Web method SQLUpdateSP.

- G.** The Command object is initialized with associated data objects such as Connection object, Command text, and Command type. Note that the Command type must be set to the `StoredProcedure` since this method will call a stored procedure, not a data query, to perform the data updating. The last initialization step for the Command object is to assign all input or updating parameters to the associated dynamic parameter in the UPDATE command.
- H.** The `ExecuteNonQuery()` method is executed to call the stored procedure to perform the data updating. An integer value will be returned from this method, and the value of this integer is equal to the number of rows that have been successfully updated in our Course table.
- I.** A cleaning job is performed to release all objects used in this method.
- J.** If the returned value from calling of the `ExecuteNonQuery()` method is zero, which means that no row has been updated in our Course table and this data updating has failed, an error message is sent to the member data `SQLError` and reported using the user-defined `ReportError()` method.

- K. Finally the instance `SQLResult` that contains the running status of this Web method is returned to the calling procedure.

Go to FileSave All menu item to save these modifications.

9.5.2.2 Modify Web Method `GetSQLInsert` to `GetSQLCourse`

The function of this Web method is to retrieve all `course_id`, including the original and updated `course_id`, and assign them to a `CourseID[]` array in our base class `SQLBase` that will be returned as an instance to the calling procedure. A client project will extract all `course_id` from this returned instance and display them in a listbox control in the client project.

Open this Web method and perform the modifications shown in Figure 9.73 to this method. The modified parts have been highlighted in bold.

Lets' take a closer look at these modified codes to see how they work.

- A. The name of this Web method is changed from `GetSQLInsert` to `GetSQLCourse`. Also the returned data type is changed to our modified base class `SQLBase`.

```

[WebMethod]
A public SQLBase GetSQLCourse(string FacultyName)
{
    string cmdString = "SELECT Course.course_id FROM Course JOIN Faculty " +
        "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.faculty_name LIKE @name)";
    B SQLBase SQLResult = new SQLBase();
    SqlCommand sqlCommand = new SqlCommand();
    SqlDataReader sqlReader;
    C SQLResult.SQLOK = true;
    sqlConnection = SQLConn();
    D if (sqlConnection == null)
    {
        SQLResult.SQLError = "Database connection is failed";
        ReportError(SQLResult);
        return null;
    }
    E sqlCommand.Connection = sqlConnection;
    sqlCommand.CommandType = CommandType.Text;
    sqlCommand.CommandText = cmdString;
    sqlCommand.Parameters.Add("@name", SqlDbType.Text).Value = FacultyName;
    sqlReader = sqlCommand.ExecuteReader();
    F if (sqlReader.HasRows == true)
        FillCourseReader(ref SQLResult, sqlReader);
    G else
    {
        SQLResult.SQLError = "No matched course found";
        ReportError(SQLResult);
    }
    H sqlReader.Close();
    sqlConnection.Close();
    sqlCommand.Dispose();
    I return SQLResult;
}

```

Figure 9.73 Modified codes for the Web method `GetSQLCourse`.

- B.** An instance of our modified base class `SQLBase`, `SQLResult`, is created, and this instance contains all retrieved `course_id` and the running status of this Web method. This instance will be returned to the calling procedure when this method is done.
- C.** First, we preset a good running status of this Web method to the member data `SQLOK` to indicate that so far our Web method is running fine.
- D.** If any error is encountered during the database connection process, the error information is stored into the member data `SQLException` and reported using the user-defined `ReportError()` method.
- E.** The `Command` object is initialized with associated data objects and properties such as `Connection` object, `Command text`, and `Command type`. Also the dynamic parameter `@name` is assigned with the actual faculty name that is an input parameter to this method.
- F.** After the `ExecuteReader()` method is called to perform this data query, we need to check the status of the property `HasRows`. If this property is `true`, which means that at least one row has been retrieved from the `Course` table, the user-defined `FillCourseReader()` method is executed to extract all `course_id` from the `DataReader` and assign them to the associated member data in the returned instance.
- G.** Otherwise if this property is `false`, which means that no row has been retrieved from the `Course` table, an error message is displayed and reported using the user-defined `ReportError()` method.
- H.** A cleaning job is performed to release all objects used in this method.
- I.** Finally the instance containing all `course_id` is returned to the calling method.

The only modification to the method `FillCourseReader()` is to change the data type of the first input argument `sResult` from `SQLInsertBase` to `SQLBase`.

9.5.2.3 Modify Web Method `GetSQLInsertCourse` to `GetSQLCourseDetail`

The function of this Web method is to retrieve the detailed information for a specific `course_id` that works as an input parameter to this method. A stored procedure `WebSelectCourseSP`, which we developed in Section 9.4.3.4.1, is called to perform this data query as this Web method is executed.

Open this Web method and perform the modifications shown in Figure 9.74 to this method. The modified parts have been highlighted in bold.

Let's take a closer look at these modified codes to see how they work.

- A.** The name of this Web method is changed from `GetSQLInsertCourse` to `GetSQLCourseDetail`. Also the returned data type is changed to our modified base class `SQLBase`.
- B.** An instance of our modified base class `SQLBase`, `SQLResult`, is created, and this instance contains the detailed information retrieved from the `Course` table based on the specific `course_id` and the running status of this Web method. This instance will be returned to the calling procedure when this method is done.
- C.** First, we preset a good running status of this Web method to the member data `SQLOK` to indicate that so far our Web method is running fine.
- D.** If any error is encountered during the database connection process, the error information is stored into the member data `SQLException` and reported using the user-defined `ReportError()` method.

```

WebServiceSQLUpdateDelete | GetSQLCourseDetail()
[WebMethod]
A public SQLBase GetSQLCourseDetail(string CourseID)
  {
    string cmdString = "dbo.WebSelectCourseSP";
    B SqlConnection sqlConnection = new SqlConnection();
    SQLBase SQLResult = new SQLBase();
    C SqlDataReader sqlReader;
    SQLResult.SQLOK = true;
    D sqlConnection = SQLConn();
    if (sqlConnection == null)
    {
      SQLResult.SQLError = "Database connection is failed";
      ReportError(SQLResult);
      return null;
    }
    E SqlCommand sqlCommand = new SqlCommand(cmdString, sqlConnection);
    sqlCommand.CommandType = CommandType.StoredProcedure;
    sqlCommand.Parameters.Add("@CourseID", SqlDbType.Text).Value = CourseID;
    sqlReader = sqlCommand.ExecuteReader();
    F if (sqlReader.HasRows == true)
      FillCourseDetail(ref SQLResult, sqlReader);
    G else
    {
      SQLResult.SQLError = "No matched course found";
      ReportError(SQLResult);
    }
    H sqlReader.Close();
    sqlConnection.Close();
    sqlCommand.Dispose();
    I return SQLResult;
  }

```

Figure 9.74 Modified codes for the Web method GetSQLCourseDetail.

- E.** The Command object is initialized with associated data objects and properties such as Connection object, Command text, and Command type. Also the dynamic parameter @CourseID is assigned with the actual CourseID that is an input parameter to this method.
- F.** After the ExecuteReader() method is called to perform this data query, we need to check the status of the property HasRows. If this property is true, which means that at least one row has been retrieved from the Course table, the user-defined FillCourseDetail() method is executed to extract the detailed course information from the DataReader and assign it to the associated member data in the returned instance.
- G.** Otherwise if this property is false, which means that no row has been retrieved from the Course table, an error message is displayed and reported using the user-defined ReportError() method.
- H.** A cleaning job is performed to release all objects used in this method.
- I.** Finally the instance containing the detailed course information is returned to the calling procedure.

The only modification to the user-defined FillCourseDetail() method is to change the data type of the first input argument sResult from SQLInsertBase to SQLBase.

The last modification to this Web method is to modify the user-defined ReportError() method. Perform the following modifications to this method:

1. Change the data type of the passed argument `ErrSource` from `SQLInsertBase` to `SQLBase`.
2. Change the first instruction from `ErrSource.SQLInsertOK = false;` to `ErrSource.SQLOK = false;`.
3. Change the second instruction from `MessageBox.Show(ErrSource.SQLInsertError);` to `MessageBox.Show(ErrSource.SQLError);`.

Next let's develop a new Web method `SQLDeleteSP` to perform the data deleting action.

9.5.2.4 Add New Web Method `SQLDeleteSP`

As we discussed in Section 7.1.1, to delete a record from a relational database, one needs to follow the operation steps listed below:

1. Delete records that are related to the parent table using the foreign keys from child tables.
2. Delete records that are defined as primary keys from the parent table.

In other words, to delete one record from the parent table, all records that are related to that record as foreign keys and located at different child tables must be deleted first. In our case, in order to delete a record using the `course_id` as the primary key from the `Course` table (parent table), one must first delete those records using the `course_id` as a foreign key from the `StudentCourse` table (child table). Fortunately we have only one child table related to our parent table in our sample database. Refer to Section 2.5 and Figure 2.5 in Chapter 2 to get a clear relationship description among different data tables in our sample database.

From this discussion, it can be found that to delete a course record from our sample database two deleting queries need to be performed: The first query is used to delete the related records from the child table or `StudentCourse` table, and the second query is used to delete the target record from the parent table or the `Course` table. Fortunately we have set a **Cascaded** mode for the Delete Rule when we built the relationship between our data tables in our sample database. Refer to Section 2.10.4.5 in Chapter 2 to get more detailed discussions about this issue. The Cascaded Delete mode means that if a record is deleted from the parent table, all related records in the child tables will also be deleted automatically by the database engine. To save time and space as well as the efficiency, we prefer to place this deleting query into a stored procedure named `WebDeleteCourseSP()` that we will develop in the following section. A single input parameter `course_id` is passed into this stored procedure as the primary key. At this moment, we just assume that we have already developed that stored procedure and will use it in this Web method.

Open our Web Service project and our code-behind page `WebServiceSQLUpdateDelete.cs`. Create the Web method `SQLDeleteSP()`, which is shown in Figure 9.75.

Let's take a closer look at this piece of code to see how it works.

- A.** The name of this Web method is `SQLDeleteSP`, and the returned data type is our modified base class `SQLBase`.
- B.** The content of the query string is equal to the name of the stored procedure we will develop soon. The point is that the name used in this query string must be identical with the name used in our stored procedure later. Otherwise a running error may be encountered since the stored procedure is identified by its name as the project runs.

WebServiceSQLUpdateDelete		SQLDeleteSP()
	[WebMethod]	
A	public SQLBase SQLDeleteSP(string CourseID)	
	{	
B	string cmdString = "dbo.WebDeleteCourseSP";	
	SqlConnection sqlConnection = new SqlConnection();	
C	SQLBase SQLResult = new SQLBase();	
	int intDelete = 0;	
D	SQLResult.SQLOK = true;	
	sqlConnection = SQLConn();	
E	if (sqlConnection == null)	
	{	
	SQLResult.SQLError = "Database connection is failed";	
	ReportError(SQLResult);	
	return null;	
	}	
F	SqlCommand sqlCommand = new SqlCommand(cmdString, sqlConnection);	
	sqlCommand.CommandType = CommandType.StoredProcedure;	
G	sqlCommand.Parameters.Add("@CourseID", SqlDbType.Text).Value = CourseID;	
H	intDelete = sqlCommand.ExecuteNonQuery();	
I	if (intDelete == 0)	
	{	
	SQLResult.SQLError = "Data deleting is failed";	
	ReportError(SQLResult);	
	}	
J	sqlConnection.Close();	
	sqlCommand.Dispose();	
K	return SQLResult;	
	}	

Figure 9.75 Codes for the Web method SQLDeleteSP.

- C.** An instance of our modified base class `SQLBase`, `SQLResult`, is created. This instance contains the running status of this Web method and will be returned to the calling procedure when this method is done. Also a local integer variable `intDelete` is declared, and this variable is used to hold the returned value from calling the `ExecuteNonQuery()` method after this method runs.
- D.** First, we preset a good running status of this Web method to the member data `SQLOK` to indicate that so far our Web method is running fine.
- E.** If any error is encountered during the database connection process, the error information is stored into the member data `SQLError` and reported using the user-defined `ReportError()` method.
- F.** The `Command` object is created with a constructor that includes two arguments: `Command` string and `Connection` object. Then the `Command` object is initialized with associated data objects and properties such as `Command Type`. The point is that the `Command Type` property must be set to the value of `StoredProcedure` since this command will call a stored procedure to perform this data deleting later.
- G.** Also the dynamic parameter `@CourseID` is assigned with the actual `CourseID` that is an input parameter to this Web method.
- H.** The `ExecuteNonQuery()` method is executed to call our stored procedure to perform this data deleting action. This method returns an integer value to indicate the running status of this method, and the returned value is assigned to the local integer variable `intDelete`.

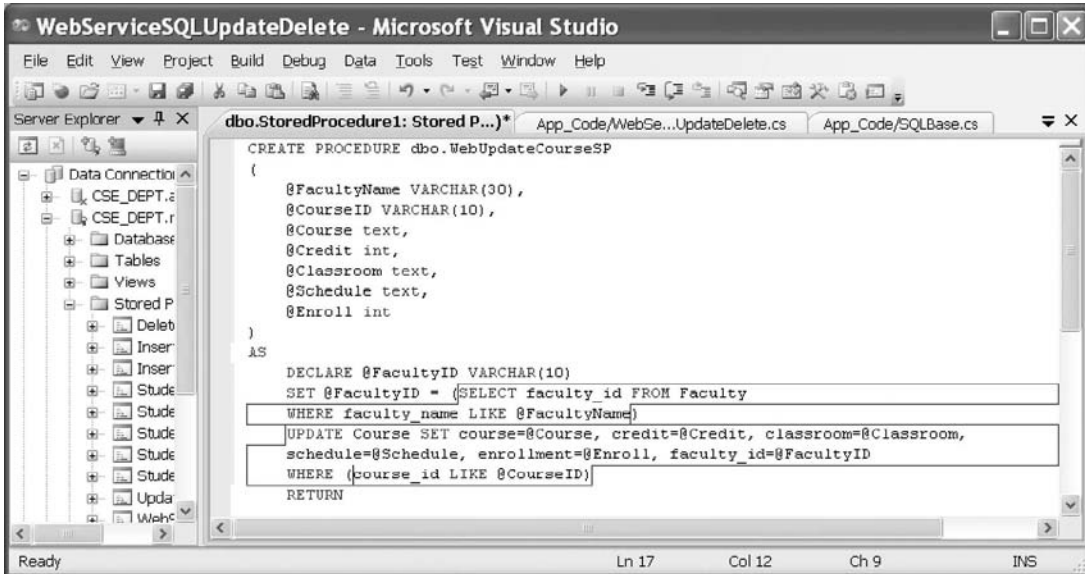


Figure 9.76 Codes for our new stored procedure WebUpdateCourseSP.

- I. The value returned from execution of the ExecuteNonQuery() method is equal to the number of rows that have been successfully deleted from the Course table. If this returned value is zero, which means that no row has been deleted from the Course table, an error message is displayed and reported using the user-defined ReportError() method.
- J. A cleaning job is performed to release all objects used in this method.
- K. Finally the instance containing the running status of this Web method is returned to the calling procedure.

At this point, we have finished all coding jobs for our Web Service project. Next let's begin to develop our two stored procedures.

9.5.3 Develop Two Stored Procedures WebUpdateCourseSP and WebDeleteCourseSP

Now we develop two stored procedures we need for this Web Service project to perform both data updating and deleting actions. Both stored procedures can be developed in the Server Explorer window in the Visual Studio.NET 2008 environment.

9.5.3.1 Develop Stored Procedure WebUpdateCourseSP

Open Visual Studio.NET 2008 and the Server Explorer window, and connect and expand our sample SQL Server database CSE_DEPT.mdf to find the Stored Procedures folder. Right-click on this folder and select the item Add New Stored Procedure from the pop-up menu to open the Add New Stored Procedure dialog box.

Enter the codes shown in Figure 9.76 into this procedure to make it our new stored procedure WebUpdateCourseSP. The actual name of this procedure is dbo.

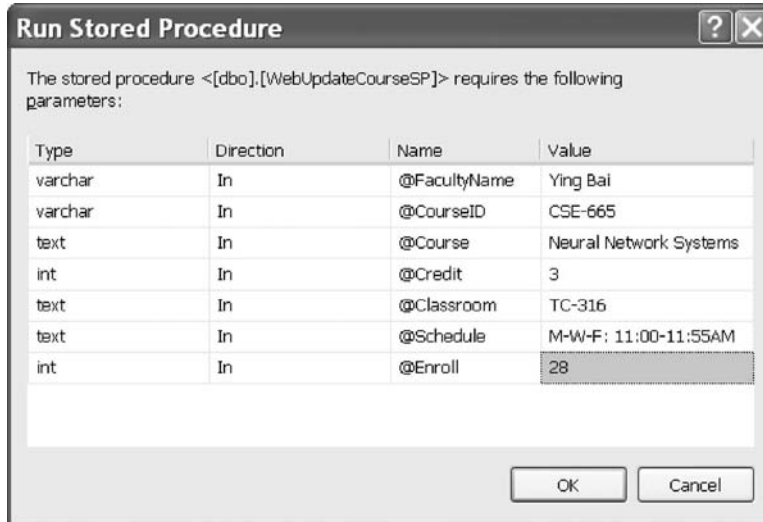


Figure 9.77 Input parameters for stored procedure WebUpdateCourseSP.

Table 9.4 The Input Parameters to the Stored Procedure

Parameter Name	Parameter Value
@FacultyName	Ying Bai
@CourseID	CSE-665
@Course	Neural Network Systems
@Credit	3
@Classroom	TC-316
@Schedule	M-W-F: 11:00-11:55 AM
@Enroll	28

WebUpdateCourseSP, however, generally we call this procedure WebUpdateCourseSP without the prefix dbo since this prefix is added automatically when a new SQL Server stored procedure is created.

Seven input parameters are listed in the parameter section with the related data types. Two queries are included in this procedure. The first one is used to pick up the desired `faculty_id` based on the input parameter `FacultyName` since there is no faculty name column available in the `Course` table. The second query is used to perform the data updating based on another six input parameters with the `course_id` as the dynamic parameter. Go to the `FileSave StoredProcedure1` menu item to save this new stored procedure.

To test this stored procedure, we can run it in the Visual Studio.NET environment. Right-click on our new created stored procedure from the Server Explorer window and select the `Execute` item from the pop-up menu to open the `Run Stored Procedure` dialog box, which is shown in Figure 9.77.

Enter the parameters shown in Table 9.4 into the `Value` box as the input parameters (refer to Figure 9.77):

Click on the `OK` button to run this stored procedure, and the running result is displayed in the `Output` windows, which is shown in Figure 9.78.


```

Output
Show output from: Database Output
Running [dbo].[WebUpdateCourseSP] ( @FacultyName = Ying Bai, @CourseID = CSE-665, @Course = 
(1 row(s) affected)
(0 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[WebUpdateCourseSP].

```

Figure 9.78 Running result of the stored procedure WebUpdateCourseSP.

Table 9.5 The Recovered Course record for CSE-665

Column Name	Column Value
course_id	CSE-665
course	Neural Network Systems
credit	3
classroom	TC-315
schedule	T-H: 1:00-2:25 PM
enrollment	26
faculty_id	B78880

The result shows that one row has been affected, which means that the selected row in the Course table has been successfully updated. To confirm this data updating at this moment, we can open our sample database CSE_DEPT.mdf in the Server Explorer window and then expand to our Course table under the Tables folder. Finally open the Course data table by right-clicking on it and select the item Show Table Data from the pop-up menu. As our Course table is fully opened, you can immediately find that this record has been updated according to the parameters we input when this procedure was executed.

It is highly recommended to recover this updated record to the original one since we will use the same input parameters later to update this record again when we test our Web Service project. So you can perform this record recovering in the opened Course table with the values shown in Table 9.5.

Next let's continue developing our second stored procedure, WebDeleteCourseSP.

9.5.3.2 Develop Stored Procedure WebDeleteCourseSP

Open Visual Studio.NET 2008 and Server Explorer window, and connect and expand our sample SQL Server database CSE_DEPT.mdf to find the Stored Procedures folder. Right-click on this folder and select the item Add New Stored Procedure from the pop-up menu to open the Add New Stored Procedure dialog box.

Enter the codes shown in Figure 9.79 into this procedure to make it our new stored procedure WebDeleteCourseSP.

The actual name of this procedure is `dbo.WebDeleteCourseSP`, however, generally we call this procedure `WebDeleteCourseSP` without the prefix `dbo` since this prefix can be added automatically when a new SQL Server stored procedure is created.

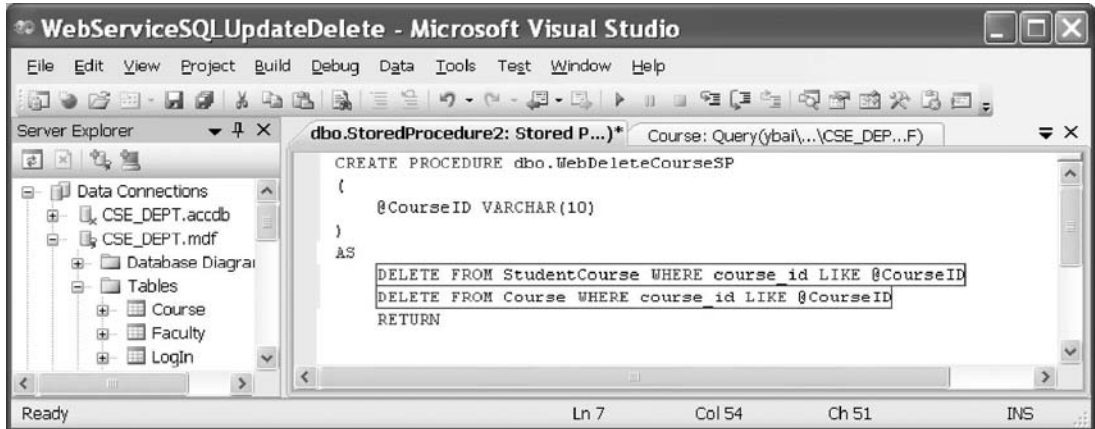


Figure 9.79 Codes for the stored procedure WebDeleteCourseSP.

One input parameter @CourseID is listed in the parameter section with the related data type. Two deleting queries are included in this procedure. The first one is used to delete all records related to the `course_id` from the child table `StudentCourse` based on the input parameter @CourseID. The second query is used to delete the target course from the parent table `Course` with the @CourseID as the dynamic parameter.

Alternatively, you can put only one deleting statement, the second statement, in this stored procedure since we have set a Cascaded mode for the Delete Rule for our data tables in our sample database in Section 2.10.4.5 in Chapter 2. As a course with a primary key `course_id` in the `Course` (parent) table is deleted, all related records in the `StudentCourse` (child) table will also be deleted because of this Cascaded Delete mode. Go to the `File|Save StoredProcedure2` menu item to save this new stored procedure.

To test this stored procedure, we can run it in the Visual Studio.NET environment. Right-click on our new created stored procedure from the Server Explorer window and select the `Execute` item from the pop-up menu to open the `Run Stored Procedure` dialog box, which is shown in Figure 9.80.

Enter CSE-526 into the `Value` box as the value of the input parameter @CourseID and click on the `OK` button to run this stored procedure. The running result is displayed in the `Output` window, which is shown in Figure 9.81.

The result shows that one row has been affected, which means that the selected row in the `Course` table has been successfully deleted. To confirm this data deleting at this moment, we can open our sample database `CSE_DEPT.mdf` using the Microsoft SQL Server 2005 Management Studio Express. Go to `Start|All Programs|Microsoft SQL Server 2005|SQL Server Management Studio Express` to open this Studio. Then open the `Course` table by right-clicking on our `dbo.Course` table and select the item `Open Table`. As our `Course` table is opened, immediately you can find that the course having our input `course_id` CSE-526 has gone from both our `Course` (parent) table and our `StudentCourse` (child) table. Since this is a new added course and no student has taken this course yet, you cannot find this course from the `StudentCourse` table.

The reason we did not use the Server Explorer window in Visual Studio.NET 2008 to open this `Course` table to confirm this data deletion is that sometimes the data tables

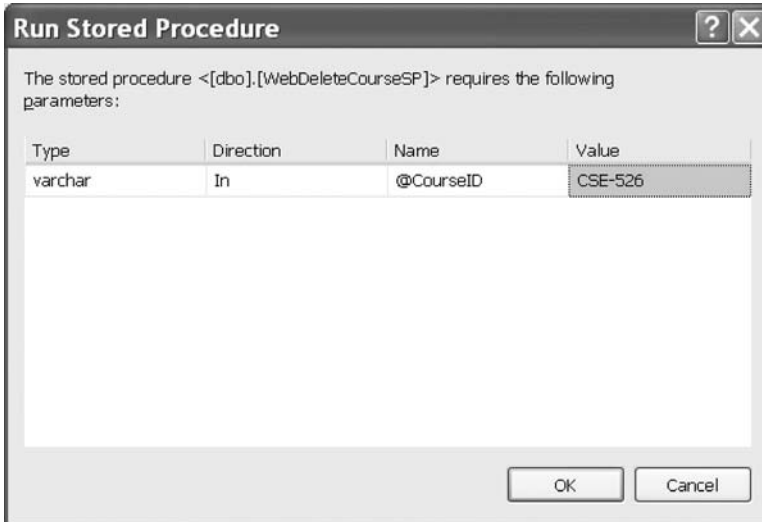


Figure 9.80 Run Stored Procedure dialog box.



Figure 9.81 Running result of the stored procedure WebDeleteCourseSP.

in this Server Explorer window cannot be updated in time after some data actions are performed against our database, and therefore those actions cannot be reflected in time.

It is highly recommended to recover those deleted records from the Course table since we will use the same input parameter later to delete this record again when we test our Web Service project later. Because this is a new added course and no student has taken this course yet, we do not need to recover it for the StudentCourse table. Just perform a recovering job for the Course table by adding the data shown in Table 9.6 into our Course table.

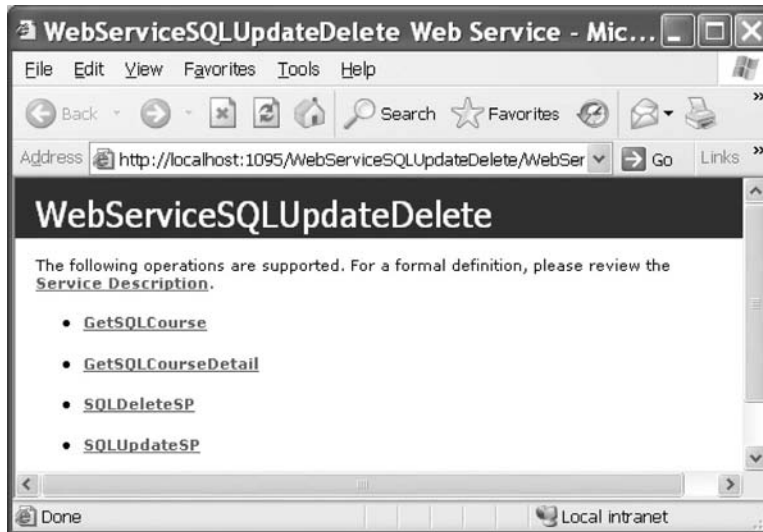
We have finished the development for this Web Service project, and now let's run our Web Service project to test all Web methods.

Click on the Start Debugging button to run our project. The built-in Web interface window is displayed, which is shown in Figure 9.82.

Four Web methods are shown in this built-in interface. First, let's test the Web method SQLUpdateSP. Click on this item to open the parameter-input interface, which is shown in Figure 9.83.

Table 9.6 The Recovered Record for CSE-526 in Course Table

Column Name	Column Value
course_id	CSE-526
course	Embedded Microcontrollers
credit	3
classroom	TC-308
schedule	M-W-F: 9:00-9:55 AM
enrollment	32
faculty_id	B78880

**Figure 9.82** Running status of the Web service project.

Enter the updated parameters shown in Figure 9.83 into the associated box to update a course with the `course_id` of CSE-526. Click on the **Invoke** button to run this method. The running result of this Web method is shown in Figure 9.84.

It can be found from this running result that the member data `SQLOK` is `true`, which means that the running status of this Web method is successful, and a record in the Course table has been updated. Because no data should be returned from the execution of this data updating, all data stored in the returned instance, including the `CourseID[]` array that has 10 elements and 2 integers, `Credit` and `Enrollment`, are either `true` or zero.

To confirm this data updating, now we can call some other Web methods to do this job. First, we want to get back all courses (that is, all `course_id`) taught by the selected faculty. To do that, close the running result interface window shown in Figure 9.84 and click on the **Back** button to return to the home page of this built-in interface. Click on the Web method `GetSQLCourse` to run it to obtain all `course_id`. Enter the faculty Ying Bai into the **Value** box as the input parameter to this Web method, which is shown in Figure 9.85. Then click on the **Invoke** button to run this method.

The running result for the Web method `GetSQLCourse` is shown in Figure 9.86. It can be found that all eight courses or `course_id` taught by the selected faculty Ying Bai are retrieved and displayed in XML tags in this built-in Web interface.



Figure 9.83 Parameter-input interface.

To confirm and check whether the target course CSE-526 has been updated or not, we need to run another Web method `GetSQLCourseDetail()`. Close the running result interface shown in Figure 9.86 and click on the Back button to return to the home page of our Web Service. Click on the Web method `GetSQLCourseDetail` to run it. Enter CSE-526 as the input parameter shown in Figure 9.87 to this method to pick up the detailed information for this course.

Click on the Invoke button to run this method, and the running result is shown in Figure 9.88. It can be found that the course CSE-526 has been updated based on the input parameters we entered for the Web method `SQLUpdateSP()` in Figure 9.83.

Next let's test the Web method `SQLDeleteSP()` to try to delete a course record from the Course table. Close the current running result window shown in Figure 9.88 and click on the Back button to return to the home page of the Web Service. Click on the Web method `SQLDeleteSP` to run it, and then enter CSE-526 as the `course_id` parameter, which is shown in Figure 9.89, into the Value box to this method. Click on the Invoke button to run this method.

The running result is shown in Figure 9.90. It can be found that the returned running status `SQLOK`, which is the only returned data, is `true`, and this means that this data deleting is successful.

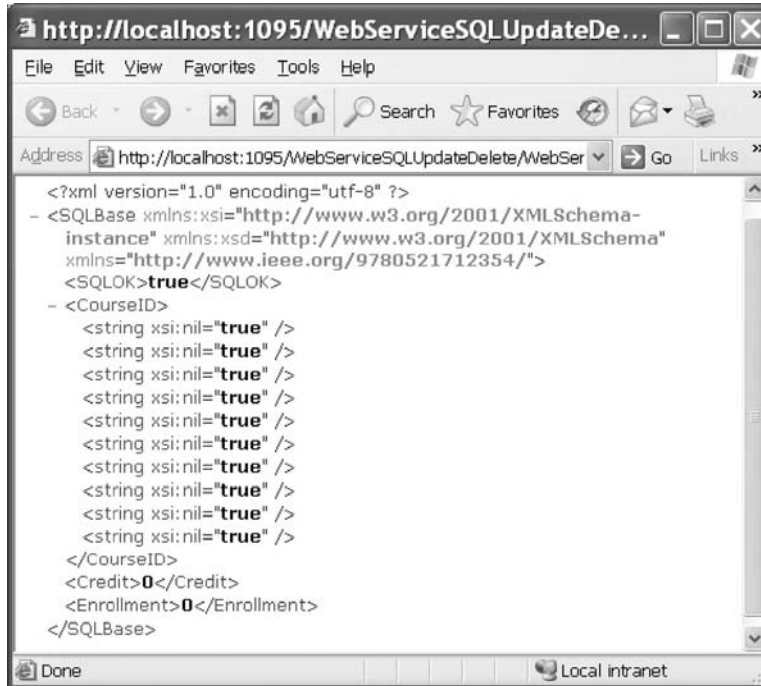


Figure 9.84 Running result for the Web method SQLUpdateSP.

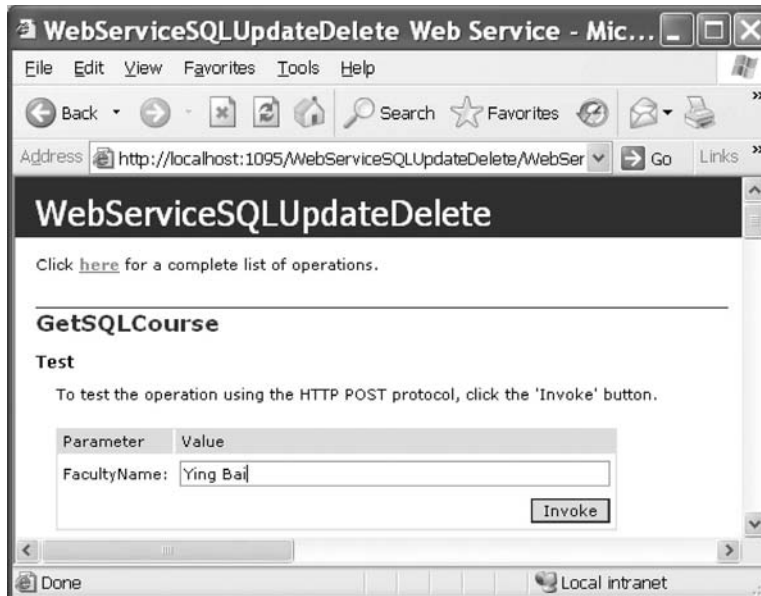


Figure 9.85 Parameter-input built-in Web interface.

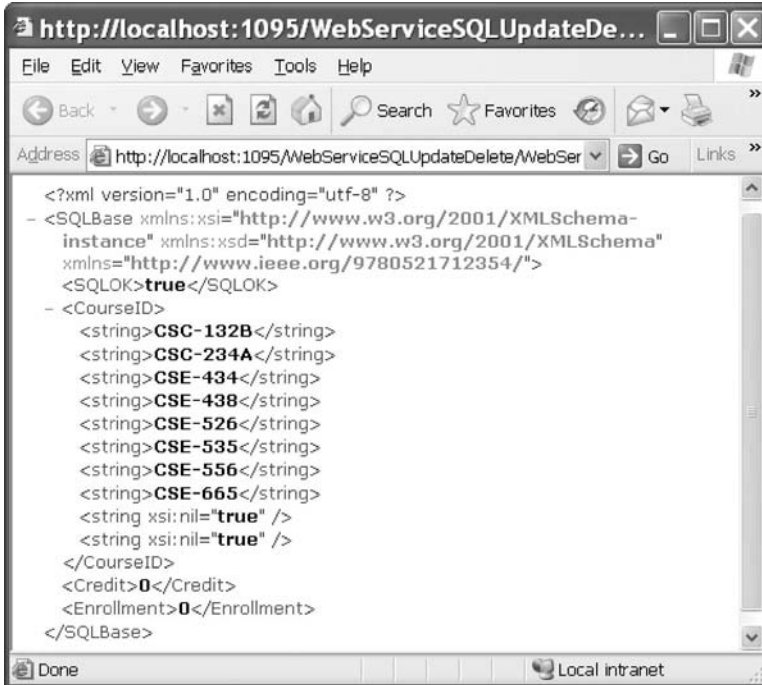


Figure 9.86 Running result for the Web method GetSQLCourse.

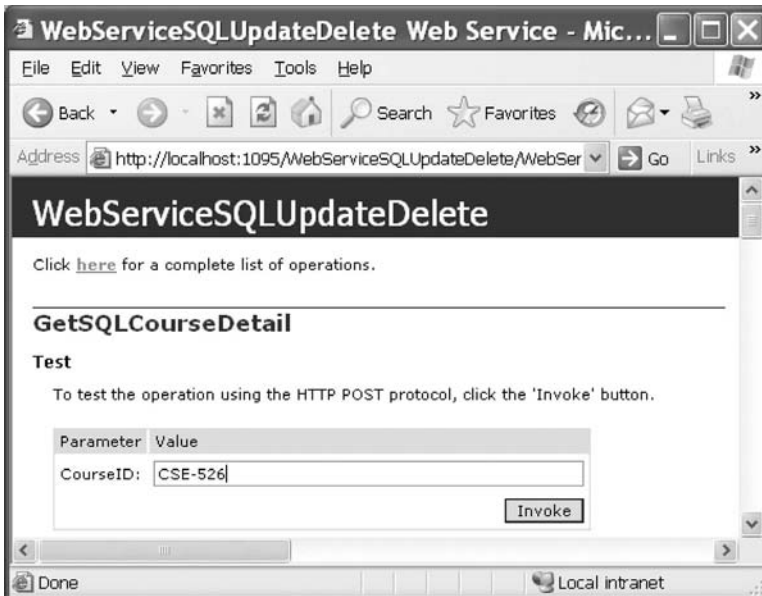


Figure 9.87 Parameter-input Web interface.

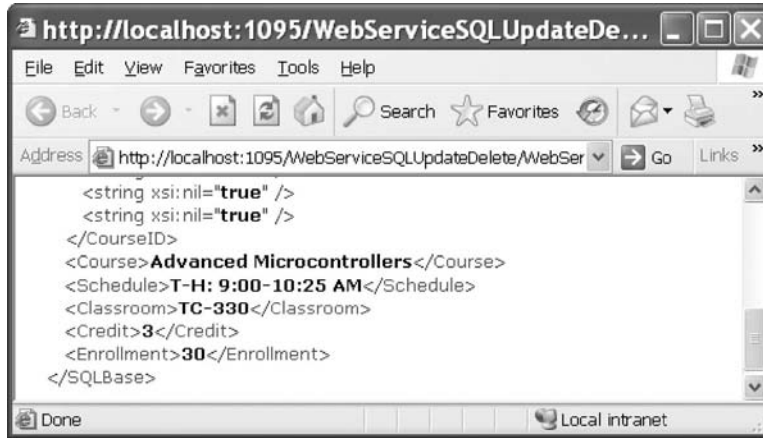


Figure 9.88 Running result of the Web method GetSQLCourseDetail.

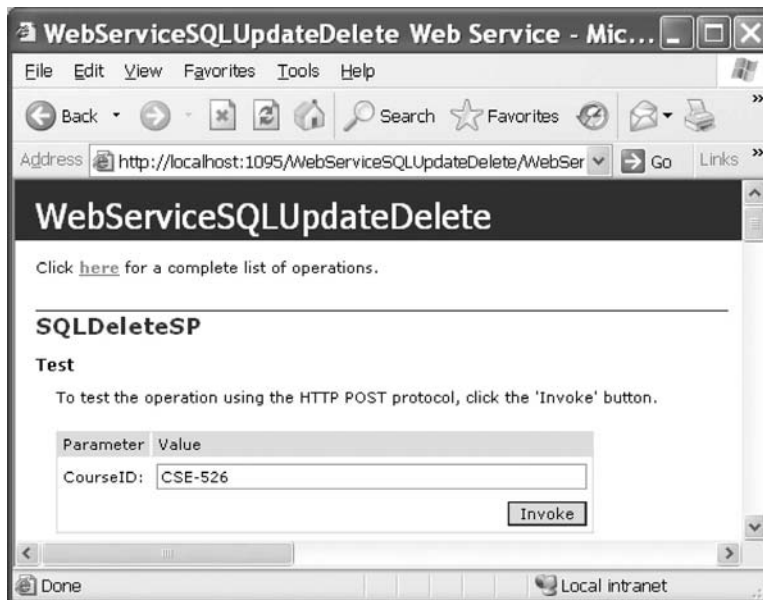


Figure 9.89 Parameter-input interface.

To confirm this data deleting, close the current running result interface shown in Figure 9.90 and click on the Back button to return to the home page of the Web Service project. Click on the Web method GetSQLCourse to run it to pick up all courses taught by the selected faculty Ying Bai. Enter the faculty name Ying Bai into the Value box as the input parameter to this method and click on the Invoke button to run it. The running result is shown in Figure 9.91.

From the running result shown in Figure 9.91, it can be found that the course with the `course_id` of CSE-526 has been deleted from the Course table.

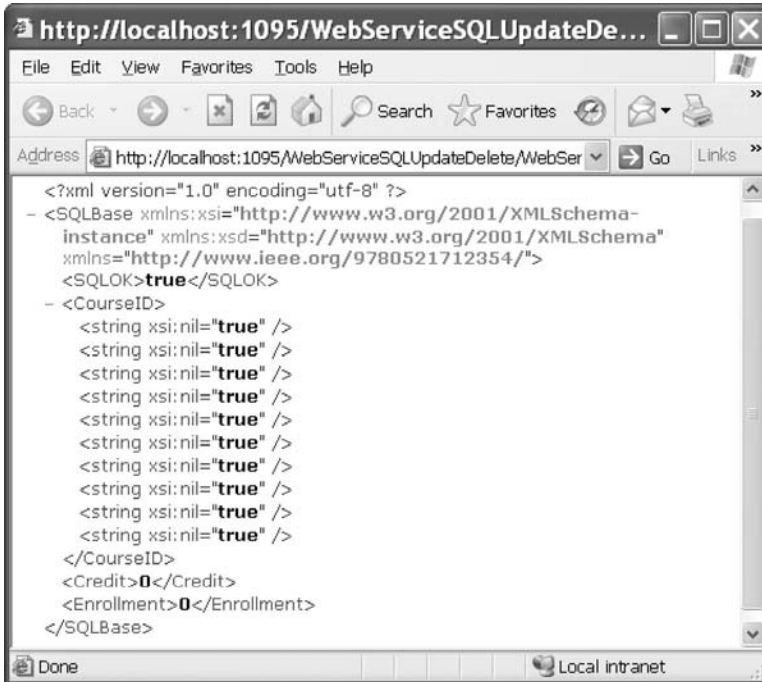


Figure 9.90 Running result of the Web method SQLDeleteSP.

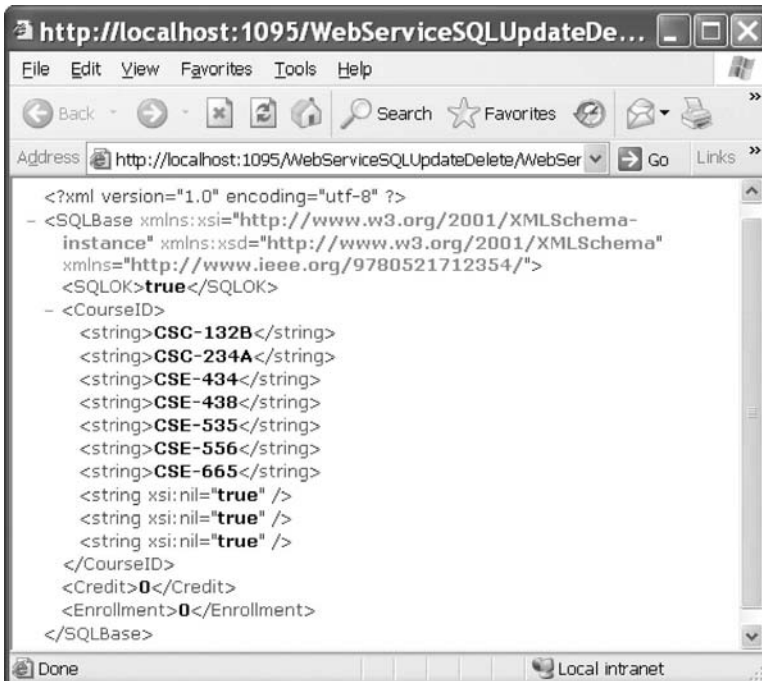


Figure 9.91 Running result of the Web method GetSQLCourse.

To get a clearer picture for this data deleting, let's try to run another Web method `GetSQLCourseDetail()`. Close the current running result interface shown in Figure 9.91 and click on the Back button to return to the home page. Select and click the Web method `GetSQLCourseDetail` to try to run it. Enter CSE-526 as the `course_id` to the Value box as the input parameter to this method and click on the Invoke button to run it.

The running process becomes very slow. The reason for this is because a message box is displayed behind the top page. Try to move the current top page to either side of the screen, and you can find that a message box with a message "No matched course found" has shown up. This means that the queried course has been deleted from the Course table and it cannot be found from that table again.

Click on the OK button to the message box to close it, and the running result is displayed, which is shown in Figure 9.92. The following returned values are displayed for two member data:

- `SQLOK`: false
- `SQLError`: No matched course found

This is identical with the warning message displayed in the message box as this method runs. Close the current page and our Web Service project. Our Web Service project is very successful.

As a reminder, it is highly recommended to recover all deleted data from all tables in our sample database. To do that, open our sample database and the Course table from either the Server Explorer in Visual Studio.NET or Microsoft SQL Server Management

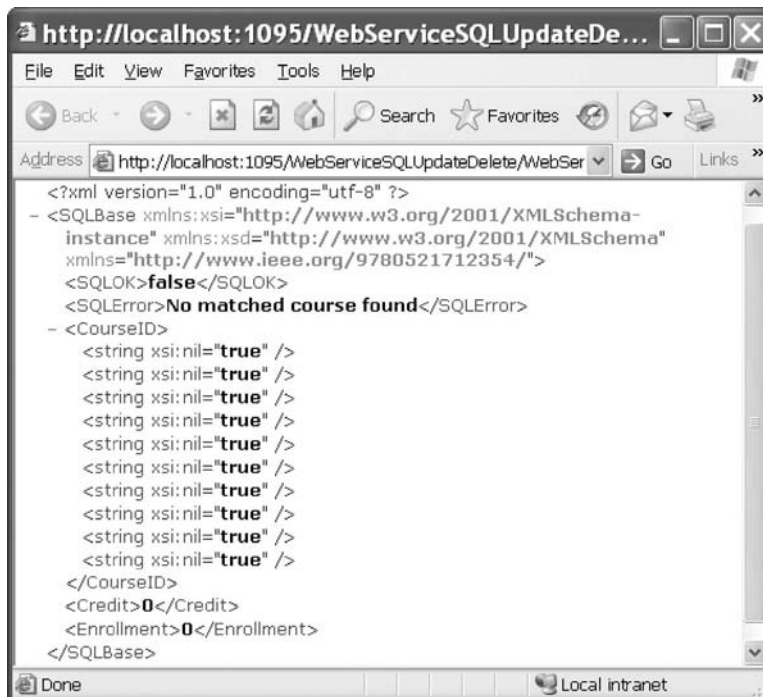


Figure 9.92 Running result of the Web method `GetSQLCourseDetail`.

Studio, and add all pieces of course information shown in Table 9.6 in Section 9.5.3.2 in this chapter for the deleted course CSE-526 into our Course table.

You can remove all message box methods `MessageBox()` from this Web Service project to speed up the execution of this Web Service if you like. A completed Web Service project `WebServiceSQLUpdateDelete` can be found at the folder `DBProjects\Chapter 9` located at the accompanying ftp site (see Chapter 1).

Next let's take care of building some Windows-based and Web-based client projects to use this Web Service.

9.6 BUILD WINDOWS-BASED WEB SERVICE CLIENTS TO USE WEB SERVICES

In order to save space, Section 9.6, which provided a detailed discussion in how to build a Windows-based client project `WinClientSQLUpdateDelete` to consume our Web Service project `WebServiceSQLUpdateDelete`, has been moved to the accompanying ftp site with a file named `WinClientSQLUpdateDelete.pdf` that can be found from the folder `DBProjects\Chapter 9\Doc` that is located at the site `ftp://ftp.wiley.com/public/sci_tech_med/practical_database`. For your convenience, a completed Windows-based client project, `WinClientSQLUpdateDelete`, has also been developed and debugged, which can be found from the folder `DBProjects\Chapter 9` at the same ftp site.

9.7 BUILD WEB-BASED WEB SERVICE CLIENTS TO USE WEB SERVICES

There is no significant difference between building a Windows-based and a Web-based client project to use a Web Service. To save time and space, we try to modify an existing Web-based client project `WebClientSQLInsert` we developed in the previous section to make it our new Web-based client project `WebClientSQLUpdateDelete`.

Actually we can copy and rename that entire project as our new Web-based client project. However, we prefer to create a new ASP.NET website project, and then copy and modify the Course page.

This section can be developed in the following sequences:

1. Create a new ASP.NET website project `WebClientSQLUpdateDelete` and add an existing website page `Course.aspx` from the project `WebClientSQLInsert` into our new project.
2. Add a Web Service reference to our new project and modify the Web form window of the `Course.aspx` to meet our data updating and deleting requirements.
3. Modify the codes in the related methods of the `Course.aspx.cs` file to call the associated Web method to perform our data updating and deleting. The code modifications include the following sections:
 - a. Remove the Insert button's Click method `cmdInsert_Click()` since we do not need any data insertion action in this application.
 - b. Remove the user-defined `FillCourseDataSet()` method since no `DataSet` method will be used in this application.

- c. Remove the TextChanged event method of the Course ID textbox since we do not need this event and its event method in this application.
- d. Modify the codes inside the Page_Load() method.
- e. Develop the codes for the Update button's Click method.
- f. Develop the codes for the Delete button's Click method.
- g. Modify the codes in the Select button's Click method and the related methods such as ProcessObject() and FillCourseListBox().
- h. Modify the codes in the SelectedIndexChanged event method of the course listbox control and the related method FillCourseDetail().

Now let's start these modifications with the first step listed above.

9.7.1 Create New Website Project and Add Existing Web Page

Open Visual Studio.NET and go to the File|New Web Site menu item to create a new website project. Enter C:\Chapter 9\WebClientSQLUpdateDelete into the name box that is next to the Location box, and click on OK to create this new project.

On the opened new project window, right-click on our new project icon WebClientSQLUpdateDelete from the Solution Explorer window, and select the item Add Existing Item from the pop-up menu to open the Add Existing Item dialog box. Browse to our Web project WebClientSQLInsert, select it, and then click on the Add button to open all existing items for this website project. Select both items, Course.aspx and Course.aspx.cs, from the list and click on the Add button to add these two items into our new website project.

9.7.2 Add Web Service Reference and Modify Web Form Window

To add a Web reference of our Web Service to this new Website project, right-click on our new project icon from the Solution Explorer window and select the item Add Web Reference from the pop-up menu. Now open our Web Service project WebServiceSQLUpdateDelete and click the Start Debugging button to run it. As the project runs, copy the URL from the Address box and paste it into the URL box in our Add Web Reference dialog box. Then click on the green Go button to search and add this Web Service as a reference to our client project. You can modify this Web reference name to any name you want. In this application, we prefer to change it to WS_SQLUpdateDelete. Your finished Add Web Reference dialog box should match the one shown in Figure 9.93.

Click on the Add Reference button to finish this adding Web reference process. Immediately you can find that the following three files are created in the Solution Explorer window under the folder App_WebReferences:

- WebServiceSQLUpdateDelete.disco
- WebServiceSQLUpdateDelete.discomap
- WebServiceSQLUpdateDelete.wsdl

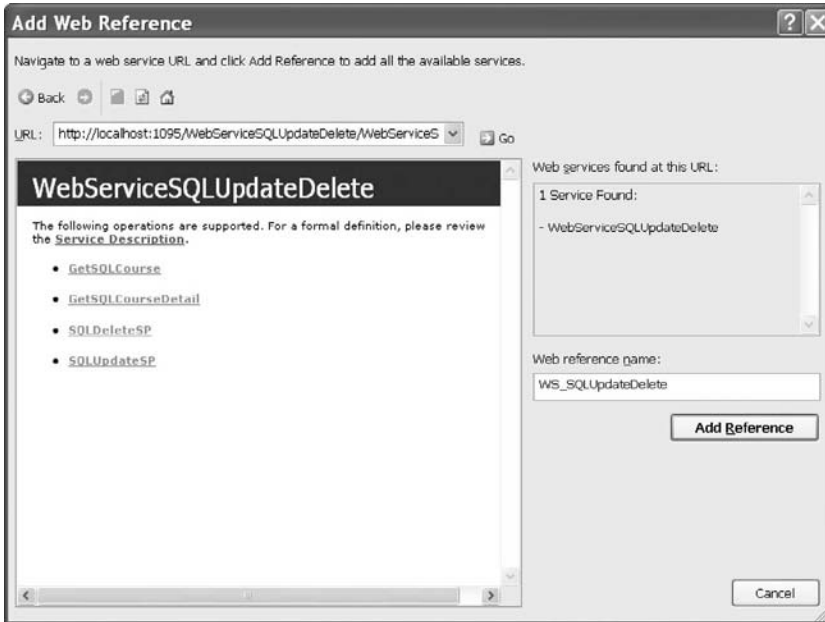


Figure 9.93 Finished Add Web Reference dialog box.

The only modification to the Web page of the `Course.aspx` is the `DropDownList` box control `ComboMethod`. Because we only use one method to perform the data updating and deleting action in our Web Service project, we do not need the `Method` combobox control in this application. We can remove this control from the graphic user interface. However, it does not matter if we keep it without using it in our Web client project. Our finished graphic user interface is shown in Figure 9.94.

Now let's take care of modifications to the codes in related user-defined methods in the `Course.aspx` page.

9.7.3 Modify Codes for Related Methods

Before we can perform any code modification, first we need to perform the following operations to delete some unused methods in our new project:

1. Remove the Insert button's Click method `cmdInsert_Click()` since we do not need any data insertion action in this application.
2. Remove the user-defined `FillCourseDataSet()` method since no `DataSet` method will be used in this application.
3. Remove the `TextChanged` event method of the Course ID textbox since we do not need this event and its event method in this application.

The first code modification is to change the codes in the `Page_Load()` method and some global variables.

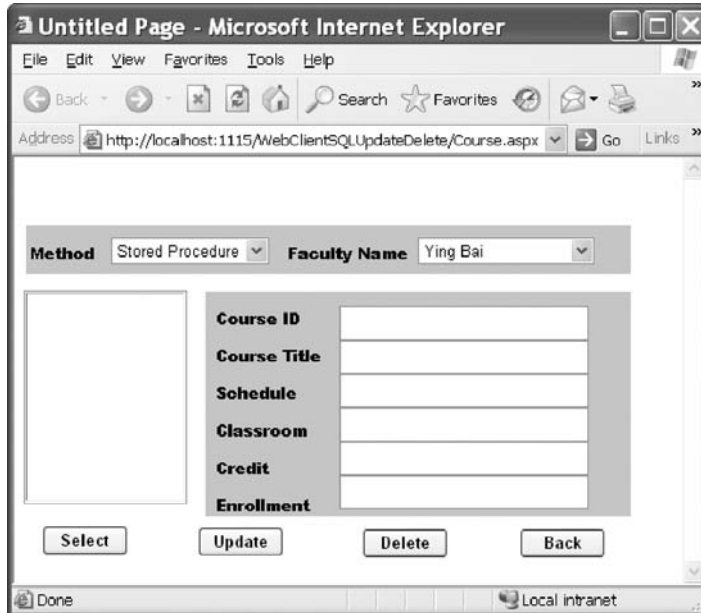


Figure 9.94 Modified graphic user interface.

9.7.3.1 Modify Codes in Page_Load Method

Perform the following changes to complete this modification:

1. Remove the field-level variables `dsFlag` and `wsDataSet`.
2. Change the name of the base class for the field-level instance `wsSQLResult` from `WS_SQLInsert.SQLInsertBase` to `WS_SQLUpdateDelete.SQLBase`.
3. In the `Page_Load()` method, remove the code that is used to add and display the second Web method, `DataSet Method`, from the `ComboMethod` control.

Your modified codes for the `Page_Load()` method should match that shown in Figure 9.95. The modified codes have been highlighted in bold. The next step is to develop the codes for the `Update` button's `Click` method.

9.7.3.2 Develop Codes for Update Button's Click Method

The function of this method is: When a faculty name is selected and all six pieces of updating course information are entered in the six textbox controls, the updating course information will be passed to the Web method `SQLUpdateSP()` we developed in our Web Service project and a stored procedure `WebUpdateCourseSP()` is executed to perform this course updating action as the `Update` button is clicked on by the user. Now let's double-click on the `Update` button to open its `Click` method, and enter the codes shown in Figure 9.96 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. A new instance of our Web proxy class, `wsSQLUpdate`, is created, and this instance is used to access the Web method `SQLUpdateSP()` we developed in our Web Service class

Course	Page_Load()
<pre> public partial class Course : System.Web.UI.Page { WS_SQLUpdateDelete.SQLBase wsSQLResult = new WS_SQLUpdateDelete.SQLBase(); protected void Page_Load(object sender, EventArgs e) { if (!IsPostBack) //these items can only be added into the combo box in one time { ComboName.Items.Add("Ying Bai"); ComboName.Items.Add("Satish Bhalla"); ComboName.Items.Add("Black Anderson"); ComboName.Items.Add("Steve Johnson"); ComboName.Items.Add("Jenney King"); ComboName.Items.Add("Alice Brown"); ComboName.Items.Add("Debby Angles"); ComboName.Items.Add("Jeff Henry"); ComboMethod.Items.Add("Stored Procedure Method"); } } } </pre>	

Figure 9.95 Modified Page_Load method.

Course	cmdUpdate_Click()
<pre> protected void cmdUpdate_Click(object sender, EventArgs e) { A WS_SQLUpdateDelete.WebServiceSQLUpdateDelete wsSQLUpdate = new B WS_SQLUpdateDelete.WebServiceSQLUpdateDelete(); C string errMsg; try { wsSQLResult = wsSQLUpdate.SQLUpdateSP(ComboName.Text, txtCourseID.Text, txtCourseName.Text, txtSchedule.Text, txtClassRoom.Text, Convert.ToInt32(txtCredits.Text), Convert.ToInt32(txtEnroll.Text)); } D catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert('" + errMsg + "')</script>"); } E if (wsSQLResult.SQLOK == false) Response.Write("<script>alert('" + wsSQLResult.SQLError + "')</script>"); } </pre>	

Figure 9.96 Codes for the Update button Click method.

WebServiceSQLUpdateDelete to perform the data updating action against our sample database.

- B.** A local string variable `errMsg` is also created, and it is used to reserve the error source that will be displayed as a part of an error message later.
- C.** A `try...catch` block is used to call the Web method `SQLUpdateSP()` with six pieces of course updating data to execute a stored procedure `WebUpdateCourseSP()` to perform this course updating action.
- D.** An error message will be displayed if any error is encountered during that data updating action. Note the displaying format of this error message. To display a string variable in a message box in the client side, one must use the Java script function

alert() with the input string variable as an argument that is enclosed and represented by "" + input_string + "".

- E. Besides the system error checking, we also need to check the member data SQLOK that is defined in our base class in the Web Service project to make sure that this data updating is application error free. A returned false value of this member data indicates that this data updating encountered some application error, and the error source stored in another member data SQLError is displayed using the Java script function alert().

In a similar way, we can develop the codes for the Delete button's Click method to perform the data deleting actions against our sample database.

9.7.3.3 Develop Codes for Delete Button's Click Method

The function of this method is: When a course_id has been selected either from the listbox control or from the Course ID textbox control in this client page window, the selected course with a primary key that equals to that course_id will be deleted from all tables, including the child and parent tables, from our sample relational database.

Double-click on the Delete button from our client page window to open the Delete Click method, and enter the codes shown in Figure 9.97 into this method.

Let's take a closer look at this piece of code to see how it works.

- A. A new instance of our Web proxy class, wsSQLDelete, is created, and this instance is used to access the Web method SQLDeleteSP() we developed in our Web Service class WebServiceSQLUpdateDelete to perform the data deleting action in our sample database.
- B. A local string variable errMsg is also created, and it is used to reserve the error source that will be displayed as a part of an error message later.
- C. A try...catch block is used to call the Web method SQLDeleteSP() with one piece of course information, course_id, that works as an identifier to run a stored procedure WebDeleteCourseSP() to perform this course deleting action in our sample database.

Course	cmdDelete_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p>	<pre>protected void cmdDelete_Click(object sender, EventArgs e) { WS_SQLUpdateDelete.WebServiceSQLUpdateDelete wsSQLDelete = new WS_SQLUpdateDelete.WebServiceSQLUpdateDelete(); string errMsg; try { wsSQLResult = wsSQLDelete.SQLDeleteSP(txtCourseID.Text); } catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert('\" + errMsg + '\")</script>"); } if (wsSQLResult.SQLOK == false) Response.Write("<script>alert('\" + wsSQLResult.SQLError + '\")</script>"); } }</pre>

Figure 9.97 Codes for the Delete button Click method.

- D. An error message will be displayed if any error is encountered during that data deleting action. Note the displaying format of this error message. To displayed a string variable in a message box in the client side, one must use the Java script function `alert()` with the input string variable as an argument that is enclosed and represented by `" + input_string + "`.
- E. Besides the system error checking, we also need to check the member data `SQLOK` that is defined in our base class in the Web Service project to make sure that this data deleting is application error free. A returned `false` value of this member data indicates that this data deleting encountered some application error and the error source stored in another member data `SQLError` is displayed.

Go to the **File|Save All** menu item to save these modifications and developments. Next let's perform the code modification to the Select button's Click method.

9.7.3.4 Modify Codes in Select Button's Click Method and Related Methods

The function of this method is: Either after a data updating or deleting action is performed, we need to confirm this operation by retrieving the related courses taught by the selected faculty from our sample database. To do that, a desired faculty should be selected from the Faculty Name combobox control, and the Select button should be clicked by the user. Then this method will call the Web method `GetSQLCourse()` we developed in our Web Service project, and an instance that contains all retrieved courses taught by the selected faculty is returned from that Web method. Some user-defined methods are executed to extract those courses from the returned instance and display them in the listbox control in our client page window. Open this method and perform the modifications shown in Figure 9.98 to this method:

Let's take a closer look at this piece of code to see how it works.

- A. Rename the new instance's name to `wsSQLSelect` and change the Web proxy class's name to `WS_SQLUpdateDelete.WebServiceSQLUpdateDelete`.
- B. Add one more local string variable `errMsg` that will be used to store the error source later. Remove the `if...else...end if` block for the method checking process since we have

Course	cmdSelect_Click()
<p>A</p> <p>B</p> <p>C</p> <p>D</p> <p>E</p> <p>F</p>	<pre>protected void cmdSelect_Click(object sender, EventArgs e) { WS_SQLUpdateDelete.WebServiceSQLUpdateDelete wsSQLSelect = new WS_SQLUpdateDelete.WebServiceSQLUpdateDelete(); string errMsg; try { wsSQLResult = wsSQLSelect.GetSQLCourse(ComboName.Text); } catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert('" + errMsg + "')</script>"); } if (wsSQLResult.SQLOK == false) Response.Write("<script>alert('" + wsSQLResult.SQLError + "')</script>"); ProcessObject(ref wsSQLResult); }</pre>

Figure 9.98 Modified codes for the Select button Click method.

only one method, stored procedure method, used in this application. Also remove all codes between the `else` and `end if` half-block since we do not have the `DataSet` method used in this project.

- C. Change the instance name of our Web proxy class from `wsSQLInsert` to `wsSQLSelect` and Web method's name from `GetSQLInsert()` to `GetSQLCourse()`.
- D. Change the name of the member data from `SQLInsertOK` to `SQLOK`.
- E. Change the name of another member data from `SQLInsertError` to `SQLError`.
- F. Remove the last two methods, `FillCourseDataSet()` and `Application["dsFlag"] = false`, since we do not need these two operations in this application.

All modification parts have been highlighted in bold.

Two user-defined methods are related to this Select button's Click method, and they are `ProcessObject()` and `FillCourseListBox()`. The modifications to these two methods include the following steps:

- A. Change the data type of passed argument `wsResult` from `WS_SQLInsert.SQLInsertBase` to `WS_SQLUpdateDelete.SQLBase`.
- B. Change the `if` block condition variable from `SQLInsertOK` to `SQLOK`.
- C. Change the error message member data from `SQLInsertError` to `SQLError`.
- D. Change the data type of passed argument `wsResult` from `WS_SQLInsert.SQLInsertBase` to `WS_SQLUpdateDelete.SQLBase`.

Two modified methods are shown in Figure 9.99 and the modified parts have been highlighted in bold.

Go to the `File/Save All` menu item to save these modifications. Next we will perform the modifications to the last method `SelectedIndexChanged`, which is an event method of the course listbox control `CourseList` in our page window.

```

A private void ProcessObject(ref WS_SQLUpdateDelete.SQLBase wsResult)
    {
        string errMsg;
        B if (wsResult.SQLOK == true)
            FillCourseListBox(ref wsResult);
        else
        {
            C errMsg = "Course information cannot be retrieved: " + wsResult.SQLError;
            Response.Write("<script>alert('" + errMsg + "')D private void FillCourseListBox(ref WS_SQLUpdateDelete.SQLBase sqlResult)
    {
        int index = 0;

        CourseList.Items.Clear(); //clean up the course listbox
        for (index = 0; index <= sqlResult.CourseID.Length - 1; index++)
        {
            CourseList.Items.Add(sqlResult.CourseID[index]);
        }
    }
  
```

Figure 9.99 Modified methods `ProcessObject` and `FillCourseListBox`.

9.7.3.5 Modify Codes in SelectedIndexChanged Method

Open the SelectedIndexChanged method of the listbox control CourseList and perform the modifications shown in Figure 9.100 to this method. Let's take a closer look at these modifications.

- A. Rename the new instance's name to `wsSQLSelect` and change the Web proxy class's name to `WS_SQLUpdateDelete.WebServiceSQLUpdateDelete`.
- B. Change the instance name of our Web proxy class from `wsSQLInsert` to `wsSQLSelect` and Web method's name from `GetSQLInsert` to `GetSQLCourseDetail`.
- C. Change the name of the member data from `SQLInsertOK` to `SQLOK`.
- D. Change the name of the member data from `SQLInsertError` to `SQLError`.

The modification to the related user-defined method `FillCourseDetail()` is to change the data type of the passed argument from `WS_SQLInsert.SQLInsertBase` to `WS_SQLUpdateDelete.SQLBase`.

At this point, we have finished all modifications to this Web-based client project, and now it is time for us to run this project to access our Web Service to perform the data updating and deleting actions. However, before we can run this project, make sure that our Web Service project `WebServiceSQLUpdateDelete` is in the open status. This can be identified by a small white icon located in the status bar at the bottom of the screen. If you cannot find this icon, open our Web Service project `WebServiceSQLUpdateDelete` and click on the Start Debugging button to run it. As long as our Web Service runs once, it can be closed by clicking on the Close button to terminate it, but the small white icon should be in there, which means that our Web Service is open and ready to be accessed.

Now click on the Build/Build Web Site menu item to build our project. You may encounter some compiling errors: One is the `txtCourseID_TextChanged` definition and the other one is the `cmdInsert_Click` definition. Both errors are located at the `Course.aspx` page since we have deleted those two methods. Open that page and remove those definitions from the `Course.aspx` page. Rebuild project and click on the Start Debugging button from our client project to run it. First, let's test the data updating function by

Course	CourseList_SelectedIndexChanged()
<p>A</p> <p>B</p> <p>C</p> <p>D</p>	<pre>protected void CourseList_SelectedIndexChanged(object sender, EventArgs e) { WS_SQLUpdateDelete.WebServiceSQLUpdateDelete wsSQLSelect = new WS_SQLUpdateDelete.WebServiceSQLUpdateDelete(); string errMsg; try { wsSQLResult = wsSQLSelect.GetSQLCourseDetail(CourseList.Text); } catch (Exception err) { errMsg = "Web service is wrong: " + err.Message; Response.Write("<script>alert('" + errMsg + "')</script>"); } if (wsSQLResult.SQLOK == false) Response.Write("<script>alert('" + wsSQLResult.SQLError + "')</script>"); FillCourseDetail(ref wsSQLResult); }</pre>

Figure 9.100 Modified codes for the SelectedIndexChanged method.

updating a course record CSE-665. However, before we can do that, we prefer to retrieve the current information for the course CSE-665. Click on the Select button to get all courses currently taught by the selected faculty member Ying Bai, and all `course_id` will be retrieved and displayed in the listbox control. Click on the course CSE-665 from the listbox control to get the detailed information for this course. Immediately the detailed information related to course CSE-665 is displayed in the associated textbox control, which is shown in Figure 9.101.

Now enter the following updating information for the course CSE-665 into the associated textbox, which is shown in Figure 9.102.

Now click on the Update button to call the Web method `SQLUpdateSP()` in our Web service project to update this course record. To check whether the course CSE-665 has been updated or not, first select another course from the listbox, such as CSC-234A, and then click on the course CSE-665 from the listbox control. Immediately the detailed information about this updated course is displayed in the associated textbox, as shown in Figure 9.103. It can be found that this course has been updated successfully with our updating information.

To test the deleting function, keep the course CSE-665 selected from the listbox, click on the Delete button to delete this record from the Course table. To confirm this course deleting action, click on the Select button to try to retrieve all courses taught by the selected faculty. Immediately all courses are returned and displayed in the listbox control. It can be found that the course CSE-665 has been removed from the Course table and you cannot find it from the listbox now.

Click on the Back button to terminate our client project. Our client project is very successful.

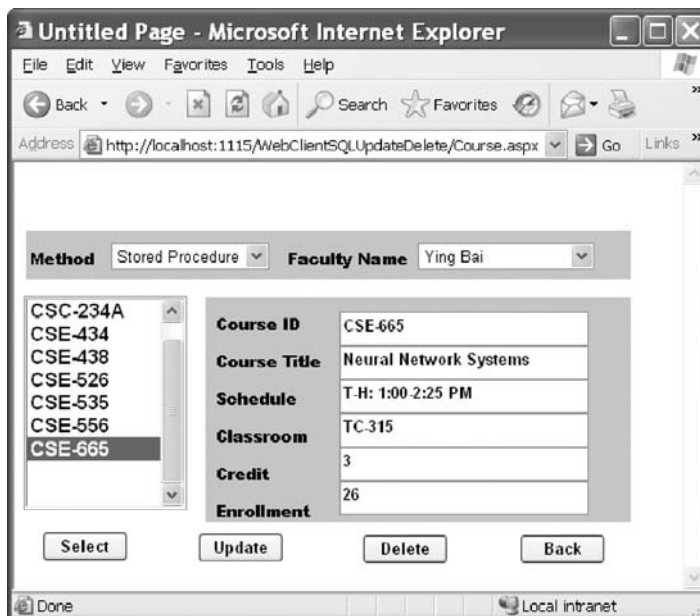


Figure 9.101 Detailed information of the course CSE-665.

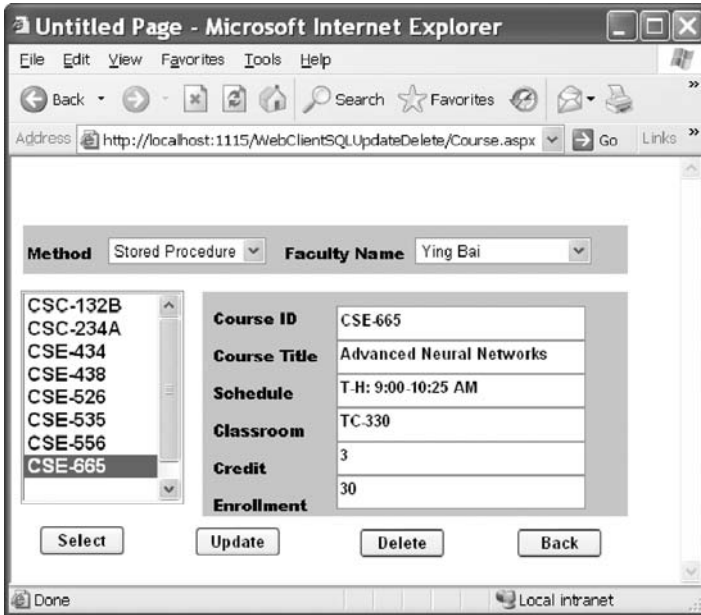


Figure 9.102 Updating information for the course CSE-665.

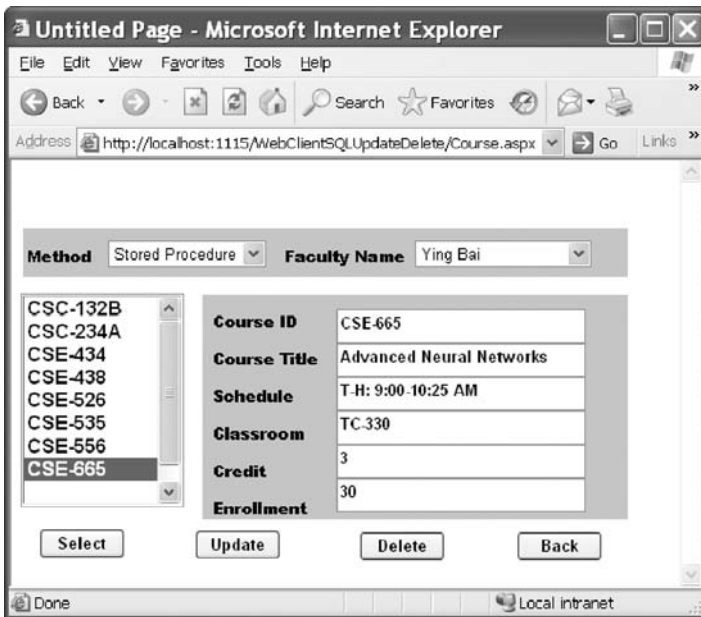


Figure 9.103 Updated course CSE-665.

Table 9.7 The Recovered Record for the Course CSE-665

Column Name	Column Value
course_id	CSE-665
course	Neural Network Systems
credit	3
classroom	TC-315
schedule	T-H: 1:00-2:25 PM
enrollment	26
faculty_id	B78880

It is highly recommended to recover the deleted course CSE-665 for our Course table since we want to keep our database neat and complete. You can recover this data by using one of the following five methods:

1. Use the Server Explorer window in Visual Studio.NET to open our sample database CSE_DEPT.mdf and our Course data table.
2. Use the Microsoft SQL Server Management Studio or Studio Express to open our sample database CSE_DEPT.mdf and our Course data table.
3. Use our Web Service project WebServiceSQLInsert to insert a new course to perform this course recovering.
4. Use our Windows-based Web Service client project WinClientSQLInsert to perform this course recovering.
5. Use our Web-based Web Service client project WebClientSQLInsert to insert a new course to recover this course record.

Relatively speaking, using the last three methods to recover this course information is professional since normally no one wants to access and change the content of a database directly by opening the database to do modifications.

Refer to Table 9.7 to recover the deleted course CSE-665.

A complete Web-based Web Service client project WebClientSQLUpdateDelete can be found at the folder DBProjects\Chapter 9 at the accompanying ftp site (see Chapter 1).

At this point, we have finished the discussion about how to access and manipulate data in the SQL Server database via ASP.NET Web Services. In the next section, we will discuss how to access and manipulate data in the Oracle database via ASP.NET Web Services.

9.8 BUILD ASP.NET WEB SERVICE PROJECT TO ACCESS ORACLE DATABASE

Basically, the procedure to build an ASP.NET Web Service to access the Oracle database is very similar to the procedure to build an ASP.NET Web Service to access the SQL Server database. The main differences are:

1. The connection string defined in the Web configuration file web.config
2. The namespace directories listed at the top of each Web Service page
3. The stored procedures used by each Web Service page

4. The protocol of the data query string used by each Web Service page
5. The nominal names of dynamic parameters for the Parameters collection object

These five distinct points exist between the procedures to build a Web Service to access two kinds of database. Let's give a little more detailed discussion for these issues.

First, when connecting to a different database, the connection string is obviously different, which includes the protocol and security issues in that connection string. Refer to Section 5.19.1 in Chapter 5 to get a clear picture for the difference that exists in the connection strings between these two kinds of databases.

Second, as we know, ADO.NET provides different Data Providers to support users to access the different databases, and these Data Providers are database dependent, which means that different Data Providers are needed to access the different databases. For the Oracle database, ADO.NET provides the namespace `System.Data.OracleClient` that contains all necessary data components to access and manipulate data stored in an Oracle database. In other words, to use matched data components provided by the ADO.NET to access an Oracle database, one must use the associated namespace to access those data components.

Third, the prototype and structure of a stored procedure are different for the different databases. To call a stored procedure to perform a data action against an SQL Server database is totally different from calling a stored procedure to perform the similar data action against an Oracle database.

For differences 4 and 5 listed above, it is clear that the format of a query string is different when calling the different databases. Also the nominal name of the dynamic parameter is distinguished when it is used for the different databases.

Based on the above discussion and analysis as well as the similarity between the SQL Server and Oracle databases, we try to develop our Web Service projects to access the Oracle database by modifying some existing Web Service projects in the following sections. In this section, we concentrate on the modifications to the Web Service project `WebServiceSQLSelect` and make it our new Web Service project `WebServiceOracleSelect`.

9.8.1 Build Web Service Project `WebServiceOracleSelect`

In this section, we try to modify an existing Web Service project `WebServiceSQLSelect` to make it our new Web Service project `WebServiceOracleSelect`, and allow it to access the Oracle database to perform the data selection queries.

Open Windows Explorer and create a new folder `Chapter 9` under the root directory if you have not done that, and then open Internet Explorer and browse to our desired source Web Service project `WebServiceSQLSelect` located at the folder `DBProjects\Chapter 9` at the accompanying ftp site (see Chapter 1). Copy and paste this project into our new folder `Chapter 9`. Rename it `WebServiceOracleSelect`. Perform the following modifications to this project:

1. Change the main Web Service page from `WebServiceSQLSelect.asmx` to `WebServiceOracleSelect.asmx`.
2. Open the `App_Code` folder and change the name of our base class file from `SQLSelectBase.cs` to `OracleSelectBase.cs`.

3. Open the App_Code folder and change the name of our derived class file from SQLSelectResult.cs to OracleSelectResult.cs.
4. Open the App_Code folder and change the name of our code-behind page from WebServiceSQLSelect.cs to WebServiceOracleSelect.cs.

Now open Visual Studio.NET 2008 and our new Web Service project WebServiceOracleSelect to perform the associated modifications to the contents of the files we renamed above. First, let's perform the modifications to our main Web Service page WebServiceOracleSelect.aspx. Open this page by double-clicking on it from the Solution Explorer window and perform the following modifications:

- Change `CodeBehind="~/App_Code/WebServiceSQLSelect.cs"` to `CodeBehind="~/App_Code/WebServiceOracleSelect.cs"`
- Change `Class="WebServiceSQLSelect"` to `Class="WebServiceOracleSelect"`

Second, open the base class file OracleSelectBase.cs and perform the following modifications:

- Change the class name and the constructor's name from SQLSelectBase to OracleSelectBase.
- Change the name of the first member data from SQLRequestOK to OracleRequestOK.
- Change the name of the second member data from SQLRequestError to OracleRequestError.

Next open the derived class file OracleSelectResult.cs by double-clicking on it from the Solution Explorer window, and perform the following modifications:

- Change the class name and the constructor's name from SQLSelectResult to OracleSelectResult.
- Change the base class name (after the resolution operator :) from SQLSelectBase to OracleSelectBase.

9.8.2 Modify Connection String

Double-click on our Web configuration file web.config from the Solution Explorer window to open it. Change the content of the connection string that is under the tag <connectionStrings> to:

```
<add name="ora_conn" connectionString="Server=XE;User
ID=CSE_DEPT;Password=reback;"/>
```

The Oracle database server XE is used for the server name, the User ID is our sample database CSE_DEPT and the Password is determined by the user when adding a new account to create a new user database.

9.8.3 Modify Namespace Directories

First, we need to add an Oracle Data Provider reference to our Web Service project. To do that, right-click on our new project icon WebServiceOracleSelect from the Solution Explorer window, and then select the item Add Reference from the pop-up menu to

open the **Add Reference** dialog box. Browse down along the list until you find the item **System.Data.OracleClient**, click to select it and then click on the **OK** button to add it into our project.

Now double-click on our code-behind page **WebServiceOracleSelect.cs** to open it. On the opened page, add one more namespace line to the top of this page:

```
using System.Data.OracleClient;
```

Also change the name of our Web Service class, which is located after the accessing mode **public class**, and the constructor of this class from **WebServiceSQLSelect** to **WebServiceOracleSelect**.

Next we will perform the necessary modifications to three Web methods combined with those five differences listed above.

9.8.4 Modify Web Method GetSQLSelect and Related Methods

The following issues are related to this modification:

1. The name of this Web method and the name of the returned data type class
2. The query string used in this Web method
3. The names of the data components used in this Web method
4. The user-defined **SQLConn()** and **ReportError()** methods
5. The name of the dynamic parameter

Let's perform those modifications step by step according to this sequence.

Open the Web method **GetSQLSelect()** and perform the modifications shown in Figure 9.104 to this method.

- A. Rename this Web method to **GetOracleSelect** and the name of the returned class to **OracleSelectResult**.
- B. Change the prefix of all data classes from **Sql** to **Oracle** and the prefix of all data objects from **sql** to **ora**, respectively.
- C. Modify the query string by replacing the **LIKE @** symbol before the dynamic parameter **facultyName** with the symbol **=:**, which is an assignment operator used in the Oracle database.
- D. Change the returned instance name from **SQLResult** to **OracleResult** and the member data from **SQLRequestOK** to **OracleRequestOK**.
- E. Change the name of the user-defined method from **SQLConn()** to **OracleConn()**.
- F. Change the prefix from **sql** to **ora** for all data objects.
- G. Modify the nominal name of the dynamic parameter by removing the **@** symbol before the nominal name **facultyName**. Also change its data type from **SqlDbType.Text** to **OracleType.VarChar**.
- H. Change the name of the returned instance from **SQLResult** to **OracleResult** and Change the prefix from **sql** to **ora** for all data objects.

Your modified Web method **GetOracleSelect()** should match that shown in Figure 9.104. All modified parts have been highlighted in bold.

	WebServiceOracleSelect	▼	GetOracleSelect()	▼
--	------------------------	---	-------------------	---

```

[WebMethod]
A public OracleSelectResult GetOracleSelect(string FacultyName)
  {
B     OracleConnection oraConnection = new OracleConnection();
     OracleSelectResult OracleResult = new OracleSelectResult();
     OracleCommand oraCommand = new OracleCommand();
     OracleDataReader oraReader;
C     string cmdString = "SELECT faculty_id, office, phone, college, title, email FROM Faculty " +
                        "WHERE faculty_name =: facultyName";
D     OracleResult.OracleRequestOK = true;
E     oraConnection = OracleConn();
F     if (oraConnection == null)
        {
            OracleResult.OracleRequestError = "Database connection is failed";
            ReportError(OracleResult);
            return null;
        }
     oraCommand.Connection = oraConnection;
     oraCommand.CommandType = CommandType.Text;
     oraCommand.CommandText = cmdString;
G     oraCommand.Parameters.Add("facultyName", OracleType.VarChar).Value = FacultyName;
H     oraReader = oraCommand.ExecuteReader();
     if (oraReader.HasRows == true)
         FillFacultyReader(ref OracleResult, oraReader);
     else
        {
            OracleResult.OracleRequestError = "No matched faculty found";
            ReportError(OracleResult);
        }
     oraReader.Close();
     oraConnection.Close();
     oraCommand.Dispose();
     return OracleResult;
  }

```

Figure 9.104 Modified Web method GetOracleSelect.

Next let's modify three user-defined methods: `SQLConn()`, `FillFacultyReader()`, and `ReportError()`, which are related to the `GetOracleSelect()` method. Open these methods and perform the modifications shown in Figure 9.105.

Let's talk a closer look at these modifications to see how they work:

- A.** Change the name of this method from `SQLConn()` to `OracleConn()` and the returning class name from `SqlConnection` to `OracleConnection`. Also change the connection string from `sql_conn` to `ora_conn`.
- B.** Change the data type of the returned connection object to `OracleConnection`.
- C.** Change the data type of the first passed argument from `SQLSelectResult` to `OracleSelectResult`. Also change the data type of the second passed argument from `SqlDataReader` to `OracleDataReader`.
- D.** Change the data type of the passed argument from `SQLSelectResult` to `OracleSelectResult`.
- E.** Change the name of the first member data from `SQLRequestOK` to `OracleRequestOK`.
- F.** Change the name of the second member data from `SQLRequestError` to `OracleRequestError`.

```

A protected OracleConnection OracleConn()
    {
B     string cmdString = ConfigurationManager.ConnectionStrings["ora_conn"].ConnectionString;
        OracleConnection conn = new OracleConnection();
        conn.ConnectionString = cmdString;
        conn.Open();
        if (conn.State != System.Data.ConnectionState.Open)
        {
            MessageBox.Show("Database Open is failed");
            conn = null;
        }
        return conn;
    }

C protected void FillFacultyReader(ref OracleSelectResult sResult, OracleDataReader sReader)
    {
        if (sReader.Read() == true)
        {
            sResult.FacultyID = Convert.ToString(sReader["faculty_id"]);
            sResult.FacultyOffice = Convert.ToString(sReader["office"]);
            sResult.FacultyPhone = Convert.ToString(sReader["phone"]);
            sResult.FacultyCollege = Convert.ToString(sReader["college"]);
            sResult.FacultyTitle = Convert.ToString(sReader["title"]);
            sResult.FacultyEmail = Convert.ToString(sReader["email"]);
        }
    }

D protected void ReportError(OracleSelectResult ErrSource)
    {
E     ErrSource.OracleRequestOK = false;
F     MessageBox.Show(ErrSource.OracleRequestError);
    }

```

Figure 9.105 Modified methods OracleConn, FillFacultyReader, and ReportError.

Your modified methods OracleConn(), FillFacultyReader(), and ReportError() should match that shown in Figure 9.105. All modified parts have been highlighted in bold.

9.8.5 Modify Web Method GetSQLSelectSP and Related Methods

A stored procedure WebSelectFacultySP is called when this Web method is executed to perform the faculty data query against our sample database. The modifications to this Web method include the following two parts:

1. Modifications to the stored procedure since the prototype of a stored procedure in the SQL Server is different from that of a stored procedure in the Oracle database
2. Modifications to the codes in this Web method

Now let's perform the modifications to the stored procedure first.

9.8.5.1 Modifications to Stored Procedure WebSelectFacultySP

Basically, the modifications to this stored procedure is to develop a similar procedure in the Oracle database environment. As you know, to develop a stored procedure that

returns data in the Oracle database, one must build a Package in the Oracle database since a stored procedure developed in Oracle database won't return any data. Refer to Section 5.20.3.7 in Chapter 5 to get more detailed information on building and developing a Package in the Oracle database.

Many different methods can be used to build a Package in Oracle database. In this section we want to use the Object Browser page in Oracle Database 10g Express Edition (XE) to build this Package.

Open the Oracle Database 10g XE home page by going to Start|All Programs |Oracle Database 10g Express Edition|Go To Database Home Page items. Enter the correct username CSE_DEPT and password reback to complete the LogIn process. Then click on the Object Browser and select Create|Package item to open the Create Package window, which is shown in Figure 9.106.

Each package has two parts: the definition or specification part and the body part. First, let's create the specification part by checking the Specification radio button and click on the Next button to open the Name page, which is shown in Figure 9.107.

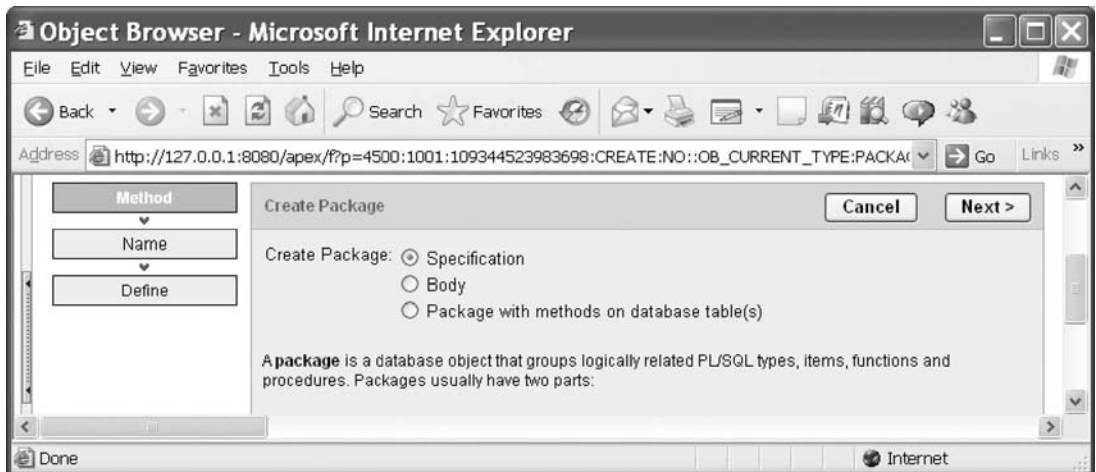


Figure 9.106 Opened Create Package window.

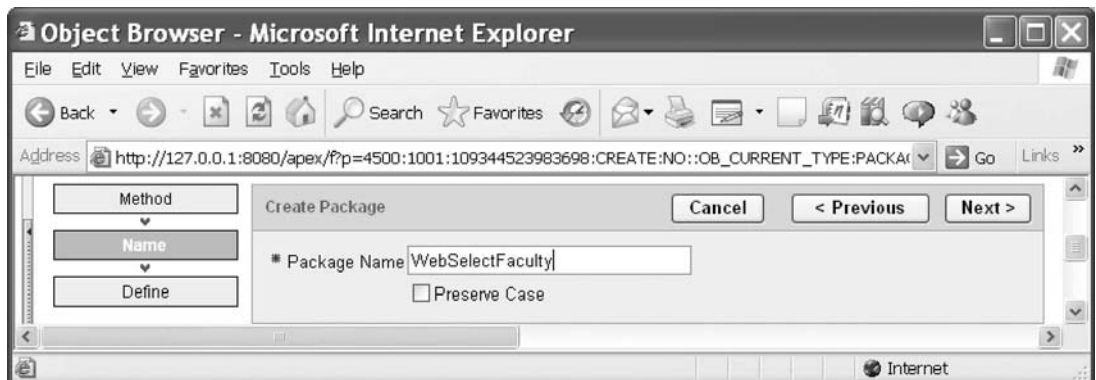


Figure 9.107 Name page of the Package window.

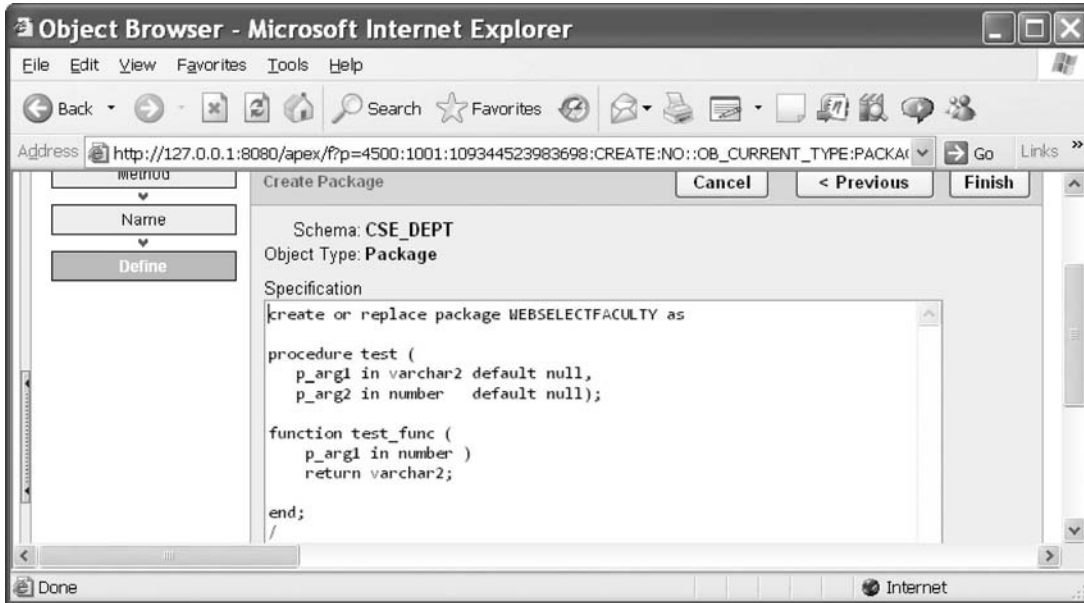


Figure 9.108 Opened Specification page.

Enter the package name `WebSelectFaculty` into the name box and click on the **Next** button to go to the specification page, which is shown in Figure 9.108.

A default package specification prototype, which includes a procedure and a function, is provided on this page, and you need to use your real specifications to replace those default items. Since we don't need any function for our application, remove the default function prototype, and change the default procedure name from `test` to our procedure name `SelectFaculty`. Your finished codes for the specification page should match that shown in Figure 9.109.

The coding language we used in this section is called Procedural Language Extension, for SQL or PL-SQL, which is a popular language and widely used in the Oracle database programming.

In line 2, we defined the returned data type as a `CURSOR_TYPE` by using:

```
TYPE CURSOR_TYPE IS REF CURSOR;
```

since we must use a cursor to return a group of data and the `IS` operator is equivalent to an equal operator.

The prototype of the procedure `SelectFaculty()` is declared in line 3. Two arguments are used for this procedure: input parameter `FacultyName`, which is indicated as an input by using the keyword **in** followed by the data type of `VARCHAR2`. The output parameter is a cursor named `FacultyInfo` followed by a keyword **out**. Each PL-SQL statement must end with a semicolon, and this rule also applies to the **end** statement.

You can click on the **Compile** button to compile this specification block if you like. Next we need to create the body block of this package. Click on the **Finish** button to complete this step. Click on the **Body** tab to open the **Body** page, which is shown in Figure 9.110.

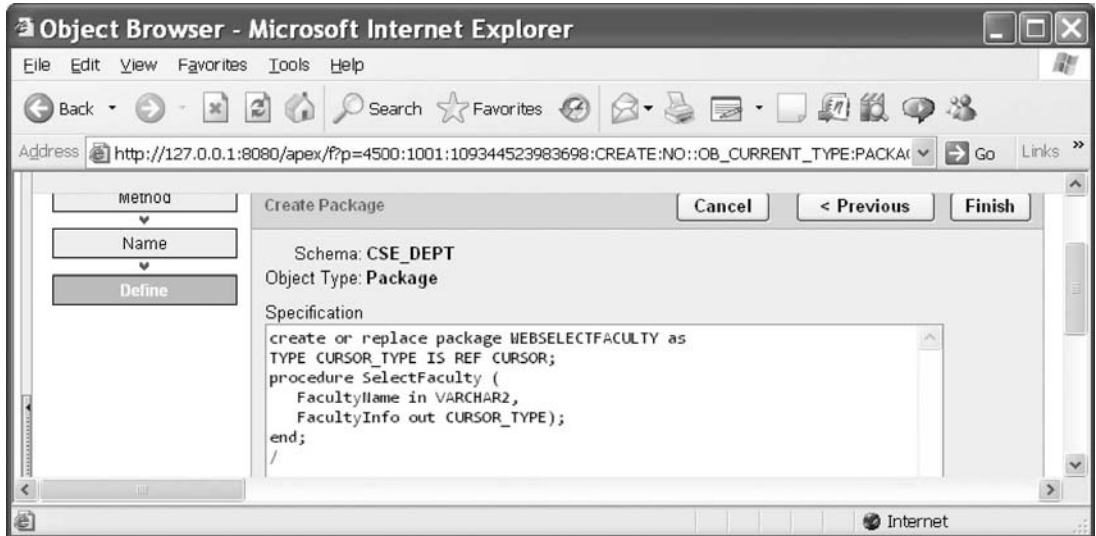


Figure 9.109 Codes for the Specification page.

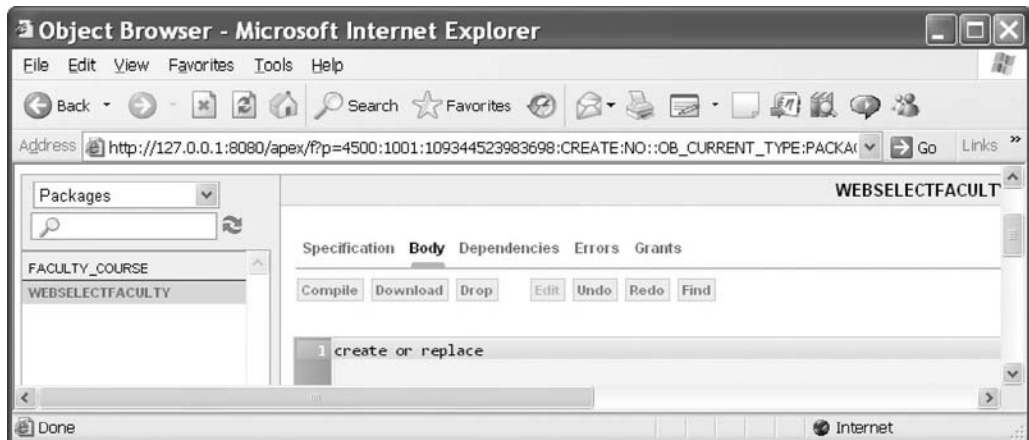


Figure 9.110 Opened Body page of the package.

Click on the Edit button to begin to create our body part. Enter the PL-SQL codes shown in Figure 9.111 into this body.

The procedure prototype is redeclared in line 2. Starting from the **begin**, our real SQL statements are included in lines 6 and 7. The **OPEN FacultyInfo FOR** command is used to assign the returned faculty data columns from the following query to the cursor variable FacultyInfo. Recall that we used a SET command to perform this assignment in the SQL Server stored procedure in Section 5.19.2.7.4 in Chapter 5. There are two **end** commands applied at the end of this Package. The first one is used to end the stored procedure and the second one is for the Package.

Fine. Now let's compile our package by clicking on the Compile button. A successful compiling information.

```

create or replace package body WebSelectFaculty AS
procedure SelectFaculty (FacultyName in VARCHAR2,
                        FacultyInfo out CURSOR_TYPE) AS
begin
OPEN FacultyInfo FOR
SELECT faculty_id, office, phone, college, title, email FROM Faculty
WHERE faculty_name = FacultyName;
end;
end;

```

Figure 9.111 Codes for the Body part of the package.

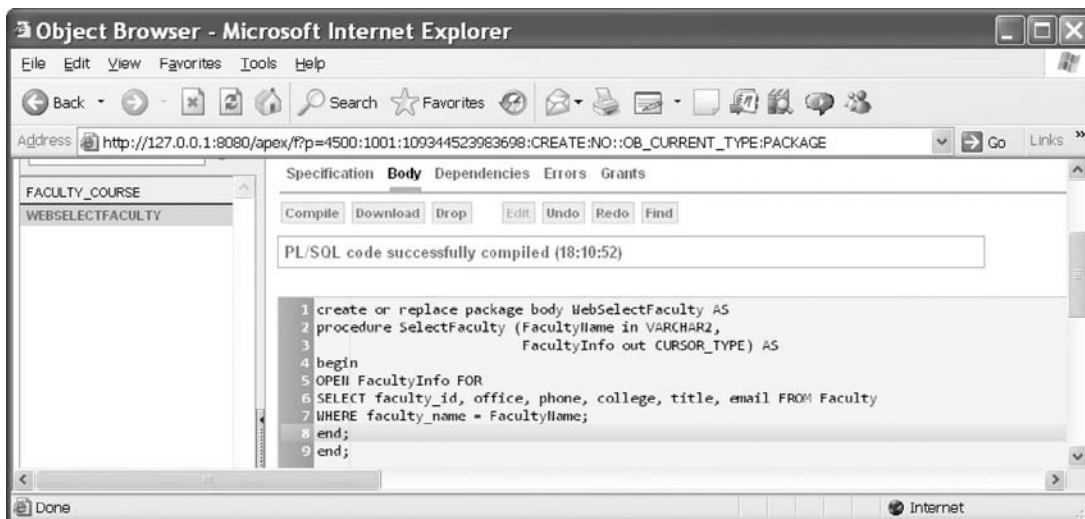


Figure 9.112 Compiled codes for the body part of the package.

PL/SQL code successfully compiled (10:56:11)

is displayed if this package is bug free, which is shown in Figure 9.112.

The development of our Oracle package is completed and now let's go to the Visual Studio.NET to call this package to perform our faculty data query via our Web Service project.

9.8.5.2 Modifications to Codes in Web Method GetSQLSelectSP

The following issues are related to this modification:

1. The name of this Web method and the name of the returned data type class
2. The content of the query string used in this Web method
3. The names of the data components used in this Web method
4. The name of the dynamic parameter
5. The names of the data classes and components used in this Web method

Open this Web method and perform those modifications step by step according to this sequence.

- A. Rename this Web method to `GetOracleSelectSP` and change the name of the returned class to `OracleSelectResult`.
- B. Change the prefix from `Sql` to `Oracle` for all data classes and from `sql` to `ora` for all data objects. Change the returned instance name from `SQLResult` to `OracleResult`; change the derived class name from `SQLSelectResult` to `OracleSelectResult`.
- C. Two Oracle Parameter objects are created here, and they are used to hold the properties of the dynamic parameters `FacultyName` and `FacultyInfo` for the stored procedure. Because the second parameter is an output parameter with a data type of `Cursor`, some special processing steps on this parameter are needed.
- D. Modify the query string by replacing the content of this string with the name of the Package we developed in the Oracle database in the last section. The point is that both the Package's name (`WebSelectFaculty`) and the stored procedure's name (`SelectFaculty`) must be used together with the dot operator to tell the Web Service that an Oracle stored procedure embedded in an Oracle Package will be called to perform this faculty data query as the project runs.
- E. Change the name of the returned instance from `SQLResult` to `OracleResult` and member data from `SQLRequestOK` to `OracleRequestOK`.
- F. Change the name of the method from `SQLConn` to `OracleConn`.
- G. Two Oracle Parameter objects are initialized with the appropriate properties and values. In fact, for the first parameter `FacultyName`, we can initialize and assign properties to it with one command line by using the `Add()` method as we did before. However, the second parameter, as we mentioned in step C, is an output parameter with a data type of `Cursor`. This makes our initialization a little complicated. Two points must be noted for this initialization: First, the data type of this parameter must be assigned to `OracleType.Cursor`, which is identical to the data type we defined in our stored procedure `SelectFaculty()`. Second, the direction of this parameter must be assigned to the **Output**, which is a value of the `ParameterDirection` property.
- H. This direction assignment is very important; otherwise, the stored procedure cannot be executed correctly if this property were not set up correctly.
- I. Change the prefix from `sql` to `ora` for all data objects.
- J. The `Add()` method is executed to add two initialized Parameter objects to the Command object and make the latter ready to be called.
- K. Change the name of the returned instance from `SQLResult` to `OracleResult` and change the prefix from `sql` to `ora` for all data objects.

Your modified Web method `GetOracleSelectSP()` should match that shown in Figure 9.113. All modified parts have been highlighted in bold.

9.8.6 Modify Web Method `GetSQLSelectDataSet`

The function of this Web method is to use a `DataSet` to store the queried faculty information and return that `DataSet` to the calling procedure. The following issues are related to this modification:

WebServiceOracleSelect		GetOracleSelectSP()
	[WebMethod]	
A	public OracleSelectResult GetOracleSelectSP(string FacultyName)	
	{	
B	OracleConnection oraConnection = new OracleConnection();	
	OracleSelectResult OracleResult = new OracleSelectResult();	
	OracleCommand oraCommand = new OracleCommand();	
	OracleDataReader oraReader;	
C	OracleParameter paramFacultyName = new OracleParameter();	
	OracleParameter paramFacultyInfo = new OracleParameter();	
D	string cmdString = "WebSelectFaculty.SelectFaculty";	
E	OracleResult.OracleRequestOK = true;	
F	oraConnection = OracleConn();	
	if (oraConnection == null)	
	{	
	OracleResult.OracleRequestError = "Database connection is failed";	
	ReportError(OracleResult);	
	return null;	
	}	
G	paramFacultyName.ParameterName = "FacultyName";	
	paramFacultyName.OracleType = OracleType.VarChar;	
	paramFacultyName.Value = FacultyName;	
	paramFacultyInfo.ParameterName = "FacultyInfo";	
	paramFacultyInfo.OracleType = OracleType.Cursor;	
H	paramFacultyInfo.Direction = ParameterDirection.Output; //this is very important	
I	oraCommand.Connection = oraConnection;	
	oraCommand.CommandType = CommandType.StoredProcedure;	
	oraCommand.CommandText = cmdString;	
J	oraCommand.Parameters.Add(paramFacultyName);	
	oraCommand.Parameters.Add(paramFacultyInfo);	
	oraReader = oraCommand.ExecuteReader();	
	if (oraReader.HasRows == true)	
	FillFacultyReader(ref OracleResult, oraReader);	
	else	
	{	
K	OracleResult.OracleRequestError = "No matched faculty found";	
	ReportError(OracleResult);	
	}	
	oraReader.Close();	
	oraConnection.Close();	
	oraCommand.Dispose();	
	return OracleResult;	
	}	

Figure 9.113 Modified Web method GetOracleSelectSP.

1. The name of this Web method
2. The content of the query string used in this Web method
3. The names of the data components used in this Web method
4. The name of the user-defined SQLConn() method
5. The name of the dynamic parameter
6. The names of the data classes and components used in this Web method

Open this Web method and perform those modifications step by step according to this sequence. Your modified Web method GetOracleSelectDataSet() should match one shown in Figure 9.114. All modified parts have been highlighted in bold.

```

WebServiceOracleSelect | GetOracleSelectDataSet()
[WebMethod]
A public DataSet GetOracleSelectDataSet(string FacultyName)
  {
B   OracleConnection oraConnection = new OracleConnection();
   OracleSelectResult OracleResult = new OracleSelectResult();
   OracleCommand oraCommand = new OracleCommand();
   OracleDataAdapter FacultyAdapter = new OracleDataAdapter();
   DataSet dsFaculty = new DataSet();
   int intResult = 0;
C   string cmdString = "SELECT faculty_id, office, phone, college, title, email FROM Faculty " +
   "WHERE faculty_name =: facultyName";
D   OracleResult.OracleRequestOK = true;
E   oraConnection = OracleConn();
F   if (oraConnection == null)
   {
   OracleResult.OracleRequestError = "Database connection is failed";
   ReportError(OracleResult);
   return null;
   }
   oraCommand.Connection = oraConnection;
   oraCommand.CommandType = CommandType.Text;
   oraCommand.CommandText = cmdString;
G   oraCommand.Parameters.Add("facultyName", OracleType.VarChar).Value = FacultyName;
   FacultyAdapter.SelectCommand = oraCommand;
   intResult = FacultyAdapter.Fill(dsFaculty, "Faculty");
   if (intResult == 0)
   {
H   OracleResult.OracleRequestError = "No matched faculty found";
   ReportError(OracleResult);
   }
   oraConnection.Close();
   oraCommand.Dispose();
   FacultyAdapter.Dispose();
   return dsFaculty;
  }

```

Figure 9.114 Modified Web method GetOracleSelectDataSet.

Let's take a closer look at this modified Web method to see how it works.

- A.** Rename this Web method GetOracleSelectDataSet.
- B.** Change the prefix from `Sql` to `Oracle` for all data classes and from `sql` to `ora` for all data objects. Also change the returned instance name from `SQLResult` to `OracleResult`; change the derived class name from `SQLSelectResult` to `OracleSelectResult`.
- C.** Modify the query string by replacing the `LIKE @` symbol before the dynamic parameter `facultyName` with the symbol `=:`, which is an assignment operator used in the Oracle database.
- D.** Change the name of the returned instance from `SQLResult` to `OracleResult`, and change the member data from `SQLRequestOK` to `OracleRequestOK`.
- E.** Change the name of the method from `SQLConn` to `OracleConn`.
- F.** Change the prefix from `sql` to `ora` for all data objects.
- G.** Modify the nominal name of the dynamic parameter by removing the `@` symbol before the nominal name `facultyName`. Also change its data type from `SqlDbType.Text` to `OracleType.VarChar`.

- H.** Change the name of the returned instance from `SQLResult` to `OracleResult` and change the prefix from `sql` to `ora` for all data objects.

At this point, we have finished all modifications to our new Web Service project. It is the time for us to run our project to test the data query functions. Click on the Start Debugging button to run our Web Service project. Click on Yes to the message box to allow the project to run in the Debug mode. The running status of the Web Service project is shown in Figure 9.115.

First, let's test the functionality of the Web method `GetOracleSelect()` to pick up the detailed information for the selected faculty member. Click on this method to open the parameter-input page shown in Figure 9.116, and enter the selected faculty name `Ying Bai` into the `Value` box. Then click on the `Invoke` button to execute this method.

The running result of this Web method is shown in Figure 9.117. The detailed information for the selected faculty is displayed in the XML tag format in this built-in Web interface page, as shown in Figure 9.117.

Now let's test the next Web method `GetOracleSelectSP()`. To do that, close the current running result page by clicking on the `Close` button located at the upper-right corner of this page, and click on the `Back` button to return the initial page. Then click on the Web method `GetOracleSelectSP` to open its parameter-input page. Enter the selected faculty name such as `Satish Bhalla` into the `Value` box and click on the `Invoke` button to execute this Web method.

This Web method will call an Oracle Package `WebSelectFaculty` that contains a stored procedure `SelectFaculty()` we developed in the previous section to access our sample Oracle database to retrieve back the detailed information for the selected faculty. The running result is shown in Figure 9.118. The detailed information for the selected faculty `Satish Bhalla` is displayed in this built-in Web interface with XML tags.

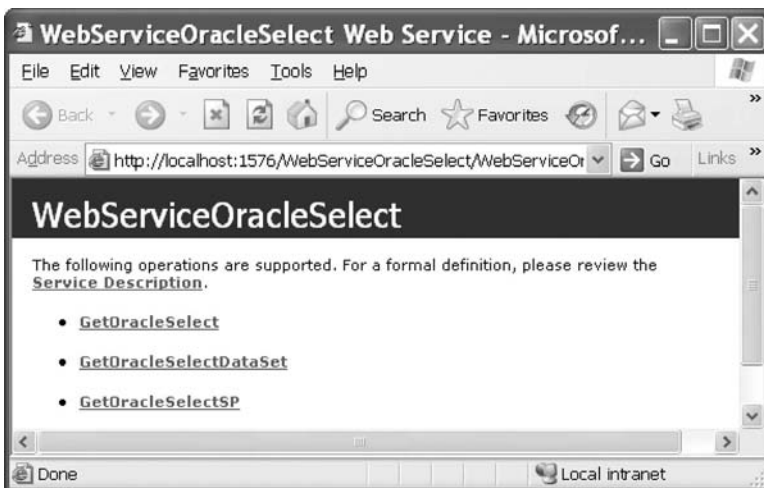


Figure 9.115 Running status of the Web service project.

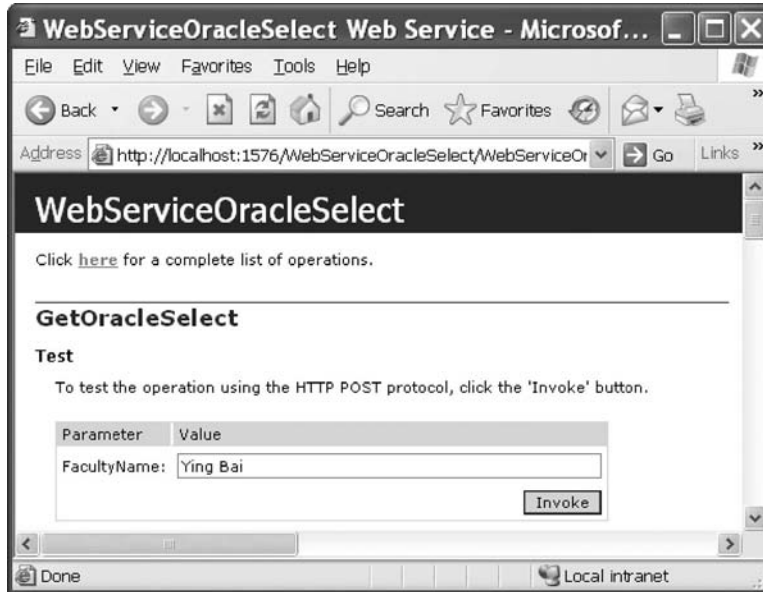


Figure 9.116 Parameter-input page.

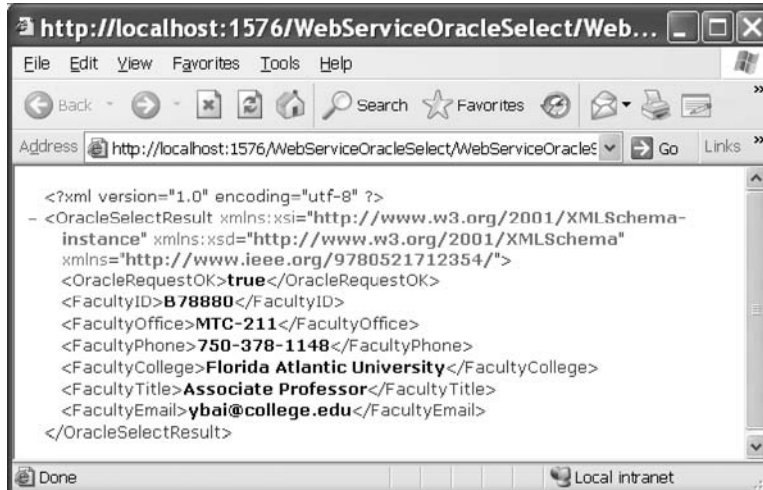


Figure 9.117 Running result of the Web method GetOracleSelect.

Finally let's test the Web method `GetOracleSelectDataSet()`. Close the current running result page and click on the Back button to return to the initial page. Click on the Web method `GetOracleSelectDataSet` to open its parameter-input built-in Web interface. Enter the selected faculty name Ying Bai and click on the Invoke button to run this method. The running result of this Web method is shown in Figure 9.119.

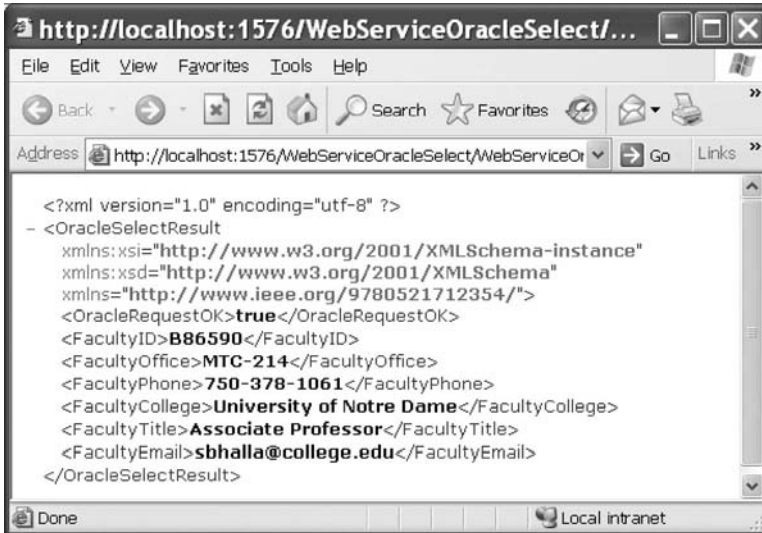


Figure 9.118 Running result of the Web method GetOracleSelectSP.

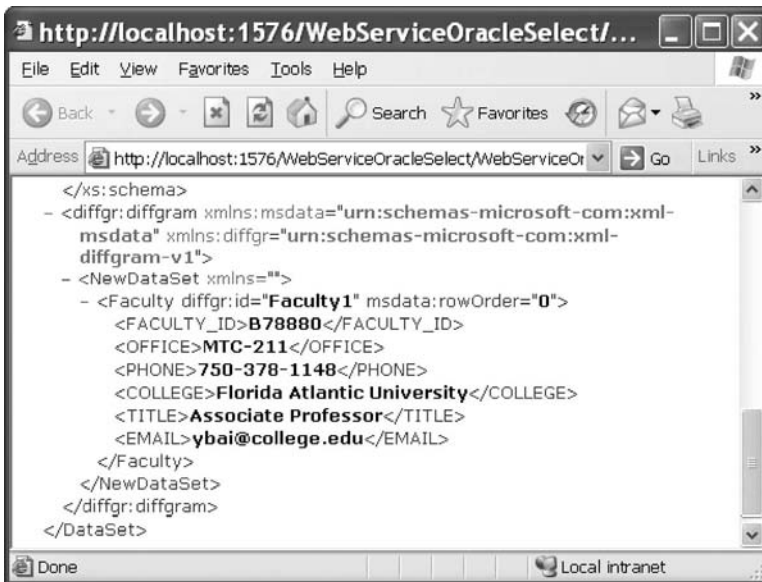


Figure 9.119 Running result of the Web method GetOracleSelectDataSet.

As shown in Figure 9.119, the detailed information contained in a DataSet for the selected faculty Ying Bai is retrieved and displayed in the XML tag format in this built-in Web interface. Our Web Service project is very successful. Click on the Close button for both built-in Web interfaces to terminate our Web Service project.

A complete Web Service project WebServiceOracleSelect can be found at the folder DBProjects/Chapter 9 at the accompanying ftp site (see Chapter 1).

9.9 BUILD WEB SERVICE CLIENTS TO USE THE WEB SERVICE WEBSERVICEORACLESELECT

To use this Web Service project, one can develop either a Windows-based or a Web-based Web Service client project. In fact, there is no significant difference between building a client project to use a Web Service to access an SQL Server database or to access an Oracle database. For example, you can use any client projects, such as either WinClientSQLSelect or WebClientSQLSelect, which we developed in the previous sections, to consume this Web Service project WebServiceOracleSelect with small modifications. The main modification is to replace the Web reference with a new Web reference class, which is our new developed Web Service WebServiceOracleSelect.

Follow the modification steps below to complete these changes.

1. Remove the old Web reference from the Windows-based or Web-based client project. You need to first delete the Web reference object and then you can delete the Web_Reference folder from the current project.
2. Add a new Web reference using the Add Web Reference dialog box, run the desired Web Service project, copy the URL from that running Web Service project, and paste it to the URL box in the Add Web Reference dialog box in the client project.
3. Change the Web reference name for all data components used in the client project.
4. Change the names of the base class and derived class located in the Web reference.
5. Change the names of all Web methods located in the Web reference.

Two completed client projects, WinClientOracleSelect, which is Windows based, and WebClientOracleSelect, which is Web based, can be found at the folder DBProjects\Chapter 9 located at the accompanying ftp site (see Chapter 1).

Next let's develop a Web Service project to perform the data insertion to the Oracle database.

9.10 BUILD ASP.NET WEB SERVICE PROJECT TO INSERT DATA INTO ORACLE DATABASE

Basically, the procedure to build an ASP.NET Web Service to insert data into the SQL Server database is very similar to the procedure to build an ASP.NET Web Service to insert data into the Oracle database. The main differences are listed below:

1. The connection string defined in the Web configuration file web.config
2. The namespace directories listed at the top of each Web Service page
3. The stored procedures used by each Web Service page
4. The protocol of the data query string used by each Web Service page
5. The nominal names of dynamic parameters for the Parameters collection object

These five distinct points exist between the procedures to build a Web Service to insert data into two kinds of databases.

Based on the discussion and analysis we made in Section 9.8 as well as the similarity between the SQL Server and the Oracle databases, we can develop our Web Service projects to insert data into the Oracle database by modifying some existing Web Service

projects in the following sections. In this section, we concentrate on the modifications to the Web Service project `WebServiceSQLInsert` and make it our new Web Service project `WebServiceOracleInsert`.

9.10.1 Build Web Service Project `WebServiceOracleInsert`

In this section, we try to modify an existing Web Service project `WebServiceSQLInsert` to make it our new Web Service project `WebServiceOracleInsert` and enable it to insert data into the Oracle database.

Open Windows Explorer and create a new folder `Chapter 9` under the root directory if you have not done that, and then open Internet Explorer and browse to our desired source Web Service project `WebServiceSQLInsert` at the folder `DBProjects\Chapter 9` at the accompanying ftp site (see Chapter 1). Copy and paste it into our new folder `Chapter 9`. Rename it `WebServiceOracleInsert` and perform the following modifications to this project in the Windows Explorer window:

1. Change the main Web Service page from `WebServiceSQLInsert.asmx` to `WebServiceOracleInsert.asmx`.
2. Open the `App_Code` folder and change the name of our base class file from `SQLInsertBase.cs` to `OracleInsertBase.cs`.
3. Open the `App_Code` folder and change the name of our code-behind page from `WebServiceSQLInsert.cs` to `WebServiceOracleInsert.cs`.

Now open the Visual Studio.NET 2008 and our new Web Service project `WebServiceOracleInsert` to perform the associated modifications to the contents of the files we renamed above. First, let's perform the modifications to our main Web Service page `WebServiceOracleInsert.asmx`. Open this page by double-clicking on it from the Solution Explorer window and perform the following modifications:

- Change `CodeBehind="~/App_Code/WebServiceSQLInsert.cs"` to `CodeBehind="~/App_Code/WebServiceOracleInsert.cs"`.
- Change `Class="WebServiceSQLInsert"` to `Class="WebServiceOracleInsert"`.

Second, open the base class file `OracleInsertBase.cs` and perform the following modifications:

- Change the class name and the constructor's name from `SQLInsertBase` to `OracleInsertBase`.
- Change the name of the first member data from `SQLInsertOK` to `OracleInsertOK`.
- Change the name of the second member data from `SQLInsertError` to `OracleInsertError`.

Go to the `File|Save All` menu item to save these modifications.

9.10.2 Modify Connection String

Double-click on our Web configuration file `web.config` from the Solution Explorer window to open it. Change the content of the connection string that is under the tag `<connectionStrings>` to:


```
<add name="ora_conn" connectionString="Server=XE;User
ID=CSE_DEPT;Password=reback;"/>
```

The Oracle database server XE is used for the server name, the user ID is our sample database CSE_DEPT, and the password is determined by the user when adding a new account to create a new user database.

9.10.3 Modify Namespace Directories

First, we need to add an Oracle Data Provider Reference to our Web Service project. To do that, right-click on our new project icon WebServiceOracleInsert from the Solution Explorer window, and then select the item Add Reference from the pop-up menu to open the Add Reference dialog box. Browse down along the list until you find the item System.Data.OracleClient, click to select it, and then click on the OK button to add it into our project.

Now double-click on our code-behind page WebServiceOracleInsert.cs to open it. On the opened page, add the following namespace line to the namespace area on this page:

```
using System.Data.OracleClient;
```

Change the name of our Web Service class, which is located after the accessing mode public class, and the constructor from WebServiceSQLInsert to WebServiceOracleInsert. Next we will perform some necessary modifications to four Web methods developed in this Web Service project combined with those five differences listed above.

9.10.4 Modify Web Method SetSQLInsertSP and Related Methods

The following issues are related to this modification:

1. The name of this Web method and the name of the returned data type class
2. The content of the query string used in this Web method
3. The names of the data components used in this Web method
4. The user-defined SQLConn() and ReportError() methods
5. The names of the dynamic parameters

Let's perform those modifications step by step according to this sequence. Your modified Web method SetOracleInsertSP is shown in Figure 9.120. All modified parts have been highlighted in bold.

Let's take a closer look at this piece of codes to see how it works.

- A. Rename our Web Service class and the constructor to WebServiceOracleInsert.
- B. Rename this Web method to SetOracleInsertSP and change the name of returned class to OracleInsertBase.
- C. Modify the content of the query string by changing the name of the stored procedure from dbo.InsertFacultyCourse to InsertFacultyCourse. The former is an SQL Server stored procedure and the latter is an Oracle stored procedure.

WebServiceOracleInsert		▼	SetOracleInsertSP()	▼
A			public class WebServiceOracleInsert : System.Web.Services.WebService	
			{	
			public WebServiceOracleInsert()	
			{	
			//Uncomment the following line if using designed components	
			//InitializeComponent();	
			}	
			[WebMethod]	
B			public OracleInsertBase SetOracleInsertSP(string FacultyName, string CourseID, string Course,	
			string Schedule, string Classroom, int Credit, int Enroll)	
			{	
C			string cmdString = "InsertFacultyCourse";	
D			OracleConnection oraConnection = new OracleConnection();	
			OracleInsertBase SetOracleResult = new OracleInsertBase();	
			OracleCommand oraCommand = new OracleCommand();	
			int intInsert = 0;	
E			SetOracleResult.OracleInsertOK = true;	
F			oraConnection = OracleConn();	
			if (oraConnection == null)	
			{	
			SetOracleResult.OracleInsertError = "Database connection is failed";	
			ReportError(SetOracleResult);	
			return null;	
			}	
G			oraCommand.Connection = oraConnection;	
			oraCommand.CommandType = CommandType.StoredProcedure;	
			oraCommand.CommandText = cmdString;	
H			oraCommand.Parameters.Add("FacultyName", OracleType.VarChar).Value = FacultyName;	
			oraCommand.Parameters.Add("CourseID", OracleType.VarChar).Value = CourseID;	
			oraCommand.Parameters.Add("Course", OracleType.VarChar).Value = Course;	
			oraCommand.Parameters.Add("Schedule", OracleType.VarChar).Value = Schedule;	
			oraCommand.Parameters.Add("Classroom", OracleType.VarChar).Value = Classroom;	
			oraCommand.Parameters.Add("Credit", OracleType.Int32).Value = Credit;	
			oraCommand.Parameters.Add("Enroll", OracleType.Int32).Value = Enroll;	
			intInsert = oraCommand.ExecuteNonQuery();	
			oraConnection.Close();	
			oraCommand.Dispose();	
			if (intInsert == 0)	
			{	
I			SetOracleResult.OracleInsertError = "Data insertion is failed";	
			ReportError(SetOracleResult);	
			}	
			return SetOracleResult;	
			}	

Figure 9.120 Modified Web method SetOracleInsertSP.

- D.** Change the prefix from `Sql` to `Oracle` for all data classes and from `sql` to `ora` for all data objects. Also change the returned instance name from `SetSQLResult` to `SetOracleResult`.
- E.** Change the name of the returned instance from `SetSQLResult` to `SetOracleResult` and member data from `SQLInsertOK` to `OracleInsertOK`.
- F.** Change the name of the method from `SQLConn()` to `OracleConn()`.
- G.** Change the prefix from `sql` to `ora` for all data objects.
- H.** Modify nominal names for all seven input parameters to the stored procedure by removing the `@` symbol before each nominal name. Also change the data type of the top five input

```

WebServiceOracleInsert
OracleConn()
A protected OracleConnection OracleConn()
{
B   string cmdString = ConfigurationManager.ConnectionStrings["ora_conn"].ConnectionString;
   OracleConnection conn = new OracleConnection();
   conn.ConnectionString = cmdString;
   conn.Open();
   if (conn.State != System.Data.ConnectionState.Open)
   {
       MessageBox.Show("Database Open is failed");
       conn = null;
   }
   return conn;
}
C protected void ReportError(OracleInsertBase ErrSource)
{
D   ErrSource.OracleInsertOK = false;
E   MessageBox.Show(ErrSource.OracleInsertError);
}

```

Figure 9.121 Modified methods OracleConn and ReportError.

parameters from `SqlDbType.Text` to `OracleType.VarChar`. Change the data type for the last two input parameters from `SqlDbType.Int` to `OracleType.Int32`.

- I. Change the name of the returned instance from `SetSQLResult` to `SetOracleResult` and change the prefix from `sql` to `ora` for all data objects.

Now let's perform modifications to two related methods `SQLConn()` and `ReportError()`. Perform the following modifications to the `SQLConn()` method:

- A. Change the name of this method from `SQLConn` to `OracleConn`, and change the return class name from `SqlConnection` to `OracleConnection`. Also change the connection string from `sql_conn` to `ora_conn`.
- B. Change the data type of the returned connection object to `OracleConnection`.

Perform the following modifications to the `ReportError()` method:

- C. Change the data type of the passed argument from `SQLInsertBase` to `OracleInsertBase`.
- D. Change the name of the first member data from `SQLInsertOK` to `OracleInsertOK`.
- E. Change the name of the second member data from `SQLInsertError` to `OracleInsertError`.

Your modified methods `OracleConn()` and `ReportError()` should match that shown in Figure 9.121. All modified parts have been highlighted in bold.

For the stored procedure `InsertFacultyCourse`, we do not need to perform any modification to this one since we successfully developed this Oracle stored procedure in Section 6.10.2.1 in Chapter 6. Therefore we can directly use this procedure without any problem. Refer to Section 6.10.2.1 in Chapter 6 to get more detailed information and discussion about the development issues for this stored procedure.

9.10.5 Modify Web Method `GetSQLInsert` and Related Methods

The function of this Web method is to retrieve all `course_id`, which includes the original and the new inserted `course_id`, from the `Course` table based on the input faculty

name. This Web method will be called or used by a client project later to get back and display all `course_id` in a listbox control in the client project.

Recall that in Section 5.19.2.5 in Chapter 5, we developed a joined-table query to perform the data query from the Course table to get all `course_id` based on the faculty name. The reason for that is there is no faculty name column available in the Course table, and each course or `course_id` is only related to a `faculty_id` in the Course table. In order to get the `faculty_id` associated with the selected faculty name, one must first go to the Faculty table to perform a query to obtain it. In this situation, a join query is a desired method to complete this functionality.

Open this Web method and perform the modifications shown in Figure 9.122. All modified parts have been highlighted in bold.

Let's take a closer look at these modifications to see how they work.

- A. Change the name of this Web method from `GetSQLInsert` to `GetOracleInsert`. Also change the name of the returned instance from `SQLInsertBase` to `OracleInsertBase`.
- B. Modify the query string to match it to the ANSI 89 standard. Recall that we developed a join-table query string for SQL Server database using the ANSI 92 standard in Section 5.19.2.5 in Chapter 5. Since the ANSI 89 standard is still being used in the Oracle database, we need to modify this joined-table query string by using that standard.

```

WebServiceOracleInsert
GetOracleInsert()

[WebMethod]
public OracleInsertBase GetOracleInsert(string FacultyName)
{
    string cmdString = "SELECT Course.course_id FROM Course, Faculty " +
        "WHERE (Course.faculty_id = Faculty.faculty_id) AND (Faculty.faculty_name =: fname)";
    OracleConnection oraConnection = new OracleConnection();
    OracleInsertBase GetOracleResult = new OracleInsertBase();
    OracleCommand oraCommand = new OracleCommand();
    OracleDataReader oraReader;

    GetOracleResult.OracleInsertOK = true;
    oraConnection = OracleConn();
    if (oraConnection == null)
    {
        GetOracleResult.OracleInsertError = "Database connection is failed";
        ReportError(GetOracleResult);
        return null;
    }
    oraCommand.Connection = oraConnection;
    oraCommand.CommandType = CommandType.Text;
    oraCommand.CommandText = cmdString;
    oraCommand.Parameters.Add("fname", OracleType.VarChar).Value = FacultyName;
    oraReader = oraCommand.ExecuteReader();
    if (oraReader.HasRows == true)
    {
        FillCourseReader(ref GetOracleResult, oraReader);
    }
    else
    {
        GetOracleResult.OracleInsertError = "No matched course found";
        ReportError(GetOracleResult);
    }
    oraReader.Close();
    oraConnection.Close();
    oraCommand.Dispose();
    return GetOracleResult;
}

```

Figure 9.122 Modified Web method `GetOracleInsert`.

- C. Change the prefix from `Sql` to `Oracle` for all data classes and from `sql` to `ora` for all data objects used in this method. Also change the name of the returned instance from `GetSQLResult` to `GetOracleResult`. Change the first member data from `SQLInsertOK` to `OracleInsertOK`.
- D. Change the name of the method from `SQLConn` to `OracleConn`. Change the second member data from `SQLInsertError` to `OracleInsertError`.
- E. Change the prefix from `sql` to `ora` for all data objects.
- F. Modify the nominal name for the input parameter to the stored procedure by removing the `@` symbol before the nominal name `fname`. Also change the data type of this input parameter from `SqlDbType.Text` to `OracleType.VarChar`.
- G. Change the names of two passed arguments to the method `FillCourseReader()` from `GetSQLResult` to `GetOracleResult` and from `sqlReader` to `oraReader`.
- H. Change the name of the returned instance from `GetSQLResult` to `GetOracleResult` and change the prefix from `sql` to `ora` for all data objects.

The modifications to the related method `FillCourseReader()` are relatively simple. Perform the modifications shown in Figure 9.123 to this method.

Let's take a closer look at these modified codes on this method to see how they work as the project runs.

- A. Modify the data types of two passed arguments by changing the data type of the first argument from `SQLInsertBase` to `OracleInsertBase` and changing the data type of the second argument from `SqlDataReader` to `OracleDataReader`.
- B. Change the method from `GetSQLString(0)` to `GetOracleString(0)`.

Your modified method `FillCourseReader()` should match that shown in Figure 9.123.

9.10.6 Modify Web Method `SQLInsertDataSet`

The function of this Web method is to use an insert query to perform the course insertion and then retrieve the new inserted data and store it in a `DataSet` that will be returned to the calling procedure. Perform the modifications shown in Figure 9.124 to this Web method:.

```

WebServiceOracleInsert
FillCourseReader()
protected void FillCourseReader(ref OracleInsertBase sResult, OracleDataReader sReader)
{
    int pos = 0;
    while (sReader.Read())
    {
        sResult.CourseID[pos] = Convert.ToString(sReader.GetOracleString(0)); //the 1st column is course_id
        pos++;
    }
}

```

Figure 9.123 Modified subroutine `FillCourseReader`.

WebServiceOracleInsert	OracleInsertDataSet()
<div style="font-family: monospace; font-size: 0.9em;"> <pre> A [WebMethod] public DataSet OracleInsertDataSet(string FacultyName, string CourseID, string Course, string Schedule, string Classroom, int Credit, int Enroll) { B string cmdString = "INSERT INTO Course VALUES (:course_id, :course, :credit, :classroom, " + C ":schedule, :enrollment, :faculty_id)"; OracleConnection oraConnection = new OracleConnection(); OracleInsertBase SetOracleResult = new OracleInsertBase(); OracleCommand oraCommand = new OracleCommand(); OracleDataAdapter CourseAdapter = new OracleDataAdapter(); DataSet dsCourse = new DataSet(); int intResult = 0; string FacultyID; D SetOracleResult.OracleInsertOK = true; oraConnection = OracleConn(); if (oraConnection == null) { E SetOracleResult.OracleInsertError = "Database connection is failed"; ReportError(SetOracleResult); return null; } oraCommand.Connection = oraConnection; oraCommand.CommandType = CommandType.Text; F oraCommand.CommandText = "SELECT faculty_id FROM Faculty WHERE faculty_name =: Name"; G oraCommand.Parameters.Add("Name", OracleType.VarChar).Value = FacultyName; FacultyID = (string)oraCommand.ExecuteScalar(); H oraCommand.Parameters.Clear(); oraCommand.CommandText = cmdString; I oraCommand.Parameters.Add("faculty_id", OracleType.VarChar).Value = FacultyID; oraCommand.Parameters.Add("course_id", OracleType.Char).Value = CourseID; oraCommand.Parameters.Add("course", OracleType.VarChar).Value = Course; oraCommand.Parameters.Add("schedule", OracleType.Char).Value = Schedule; oraCommand.Parameters.Add("classroom", OracleType.VarChar).Value = Classroom; oraCommand.Parameters.Add("credit", OracleType.Int32).Value = Credit; oraCommand.Parameters.Add("enrollment", OracleType.Int32).Value = Enroll; CourseAdapter.InsertCommand = oraCommand; intResult = CourseAdapter.InsertCommand.ExecuteNonQuery(); if (intResult == 0) { J SetOracleResult.OracleInsertError = "No matched course found"; ReportError(SetOracleResult); } K oraCommand.Parameters.Clear(); L oraCommand.CommandText = "SELECT * FROM Course WHERE faculty_id =: FacultyID"; M oraCommand.Parameters.Add("FacultyID", OracleType.VarChar).Value = FacultyID; N CourseAdapter.SelectCommand = oraCommand; CourseAdapter.Fill(dsCourse, "Course"); CourseAdapter.Dispose(); oraConnection.Close(); oraCommand.Dispose(); return dsCourse; } </pre> </div>	

Figure 9.124 Modified Web method OracleInsertDataSet.

- A.** Change the name of this method from SQLInsertDataSet to OracleInsertDataSet.
- B.** Change the query string by replacing the @ symbol before each input parameter with the colon operator :, and this is required by the Oracle database operation.
- C.** Change the prefix from Sql to Oracle for all data classes and from sql to ora for all data objects used in this method. Also change the name of the returned instance from

SetSQLResult to SetOracleResult. Change the first member data from SQLInsertOK to OracleInsertOK.

- D. Change the name of the method from SQLConn to OracleConn. Change the second member data from SQLInsertError to OracleInsertError.
- E. Change the prefix from sql to ora for all data objects.
- F. Modify the dynamic name for the input parameter by replacing the SQL assignment operator LIKE@ before the nominal name Name with the Oracle assignment operator =:.
- G. Modify the nominal name of the input parameter by removing the @ symbol before the nominal name Name. Also change the data type of this input parameter from SqlDbType.Text to OracleType.VarChar.
- H. Since the next query needs to use the different parameters, the Parameters collection must be cleaned up using the Clear() method to remove the previous parameter objects. This is very important. An Oracle database exception will be encountered if this cleaning job is not performed.
- I. Modify the nominal names for all seven input parameters by removing the @ symbol before each nominal name. Also change the data type of the top five input parameters from SqlDbType.Text to OracleType.VarChar, and change the data type of the last two input parameters from SqlDbType.Int to OracleType.Int32.
- J. Change the prefix from sql to ora for all following data objects. Also change the name of the returned instance from SetSQLResult to SetOracleResult. Change the second member data from SQLInsertError to OracleInsertError.
- K. This is the same function we discussed in step H. However, this cleaning job is unnecessary in the SQL Server database operations.
- L. Modify the dynamic name for the input parameter by replacing the SQL assignment operator LIKE@ before the nominal name FacultyID with the Oracle assignment operator =:.
- M. Modify the nominal name of the input parameter by removing the @ symbol before the nominal name FacultyID. Also change the data type of this input parameter from SqlDbType.Text to OracleType.VarChar.
- N. Change the prefix from sql to ora for all data objects used in this method.

9.10.7 Modify Web Method GetSQLInsertCourse and Related Methods

The function of this Web method is to retrieve the detailed information for a selected course_id that works as an input parameter to this method. An Oracle package is executed to perform this data action and store the retrieved information to an instance that will be returned to the calling procedure.

The following three modifications must be performed for this Web method:

1. Modify the codes of this Web method.
2. Modify the related user-defined FillCourseDetail() method.
3. Modify the content of the query string and create a new Package that contains a stored procedure.

Open this Web method and perform the modifications shown in Figure 9.125 to this Web method.

Let's take a closer look at these modifications to see how they work.

- A. Change the name of this Web method from `GetSQLInsertCourse` to `GetOracleInsertCourse`. Also change the name of the returned base class from `SQLInsertBase` to `OracleInsertBase`.
- B. Change the name of the Package that we will develop in the next section from `dbo.WebSelectCourseSP` to `WebSelectCourseSP.SelectCourse`. The prefix `WebSelectCourseSP` is a Package and the `SelectCourse` is a stored procedure.
- C. Change the prefix from `Sql` to `Oracle` for all data classes and from `sql` to `ora` for all data objects used in this method. Also change the name of the returned instance from `GetSQLResult` to `GetOracleResult`. Change the first member data from `SQLInsertOK` to `OracleInsertOK`.

```

WebServiceOracleInsert
GetOracleInsertCourse()

[WebMethod]
A public OracleInsertBase GetOracleInsertCourse(string CourseID)
  {
B   string cmdString = "WebSelectCourseSP.SelectCourse";
C   OracleConnection oraConnection = new OracleConnection();
   OracleInsertBase GetOracleResult = new OracleInsertBase();
D   OracleDataReader oraReader;
   OracleParameter paramCourseID = new OracleParameter();
   OracleParameter paramCourseInfo = new OracleParameter();
E   GetOracleResult.OracleInsertOK = true;
   oraConnection = OracleConn();
   if (oraConnection == null)
   {
F       GetOracleResult.OracleInsertError = "Database connection is failed";
       ReportError(GetOracleResult);
       return null;
   }
   paramCourseID.ParameterName = "CourseID";
   paramCourseID.OracleType = OracleType.VarChar;
   paramCourseID.Value = CourseID;
   paramCourseInfo.ParameterName = "CourseInfo";
   paramCourseInfo.OracleType = OracleType.Cursor;
   paramCourseInfo.Direction = ParameterDirection.Output; //this is very important
G   OracleCommand oraCommand = new OracleCommand(cmdString, oraConnection);
   oraCommand.CommandType = CommandType.StoredProcedure;
   oraCommand.Parameters.Add(paramCourseID);
   oraCommand.Parameters.Add(paramCourseInfo);
   oraReader = oraCommand.ExecuteReader();
H   if (oraReader.HasRows == true)
       FillCourseDetail(ref GetOracleResult, oraReader);
   else
I   {
       GetOracleResult.OracleInsertError = "No matched course found";
       ReportError(GetOracleResult);
   }
   oraReader.Close();
   oraConnection.Close();
   oraCommand.Dispose();
   return GetOracleResult;
  }

```

Figure 9.125 Modified Web method GetOracleInsertCourse.

- D. Add two OracleParameter objects `paramCourseID` and `paramCourseInfo`. Because of some differences between the SQL Server and Oracle databases, we need to use the different ways to assign parameters to the Command object later.
- E. Change the name of the method from `SQLConn` to `OracleConn`. Change the second member data from `SQLInsertError` to `OracleInsertError`.
- F. Initialize two OracleParameter objects by assigning them with the appropriate values. Note the second parameter `paramCourseInfo`. The data type of this parameter is `Cursor` and the Direction is `Output`. Both values are very important to this parameter and must be assigned exactly as we did here. Otherwise a running exception will be encountered when the project runs.
- G. Add two statements to add two OracleParameter objects to the Command object.
- H. Change the names of two passed arguments to the method `FillCourseDetail()` from `GetSQLResult` to `GetOracleResult` and from `sqlReader` to `oraReader`.
- I. Change the name of the returned instance from `GetSQLResult` to `GetOracleResult` and change the second member data from `SQLInsertError` to `OracleInsertError`.

Your modified Web method `GetOracleInsertCourse()` should match that shown in Figure 9.125. All modified parts have been highlighted in bold.

The modifications to the related method `FillCourseDetail()` are simple, and the only modifications are to change the data type of two passed arguments `sResult` and `sReader`. Change the data type of the first argument from `SQLInsertBase` to `OracleInsertBase`, and change the data type for the second argument from `SqlDataReader` to `OracleDataReader`.

Now let's create an Oracle Package `WebSelectCourseSP` that contains a stored procedure `SelectCourse` to perform this course detailed information query. In this section we use the Object Browser page in Oracle Database 10g XE to build this Package.

Open the Oracle Database 10g XE home page by going to `Start|All Programs|Oracle Database 10g Express Edition|Go To Database Home Page` items. Enter correct username and password such as `CSE_DEPT` and reback to complete the Login process. Then click on the Object Browser and select `Create|Package` item to open the `Create Package` window.

Each package has two parts: the definition or specification part and the body part. First, let's create the specification part by checking the `Specification` radio button and click on the `Next` button to open the `Name` page, which is shown in Figure 9.126. Enter the package name `WebSelectCourseSP` into the `Package Name` box and click on the `Next` button to go to the specification page.

A default package specification prototype, which includes a procedure and a function, is provided in this page, and you need to use your real specifications to replace those default items. Since we don't need any function in our application, remove the default function prototype, and change the default procedure name from the `test` to our procedure name `SelectCourse`. Your finished codes for the specification page should match that shown in Figure 9.127.

In line 2, we defined the returned data type as a `CURSOR_TYPE` by using:

```
TYPE CURSOR_TYPE IS REF CURSOR;
```

since we must use a cursor to return a group of data and the **IS** operator is equivalent to an equal operator.

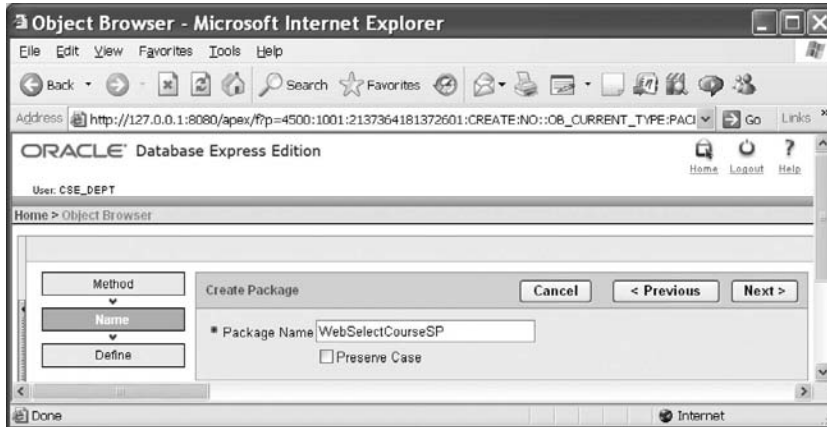


Figure 9.126 Name page of the Package window.

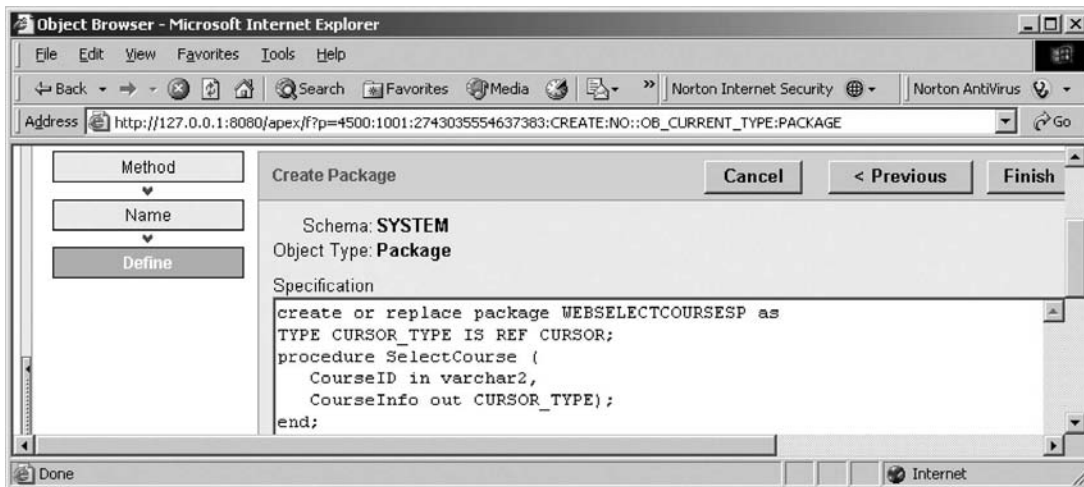


Figure 9.127 Coding for the Specification page.

The prototype of the procedure `SelectCourse()` is declared in line 3. Two arguments are used for this procedure: the input parameter `CourseID`, which is indicated as an input by using the keyword **in** followed by the data type of `VARCHAR2`. The output parameter is a cursor named `CourseInfo` followed by a keyword **out**. Each PL-SQL statement must end in a semicolon, and this rule is also applied to the **end** statement.

Next we need to create the body block of this package. Click on the **Finish** button to complete the specification setup step. Click on the **Body** tab to open the **Body** page, which is shown in Figure 9.128.

Click on the **Edit** button to begin to create our body part. Enter the PL-SQL codes into this body, which are shown in Figure 9.129.

The procedure prototype is redeclared in line 2. Starting from **begin**, our real SQL statements are included in lines 6 and 7. The **OPEN CourseInfo FOR** command is used

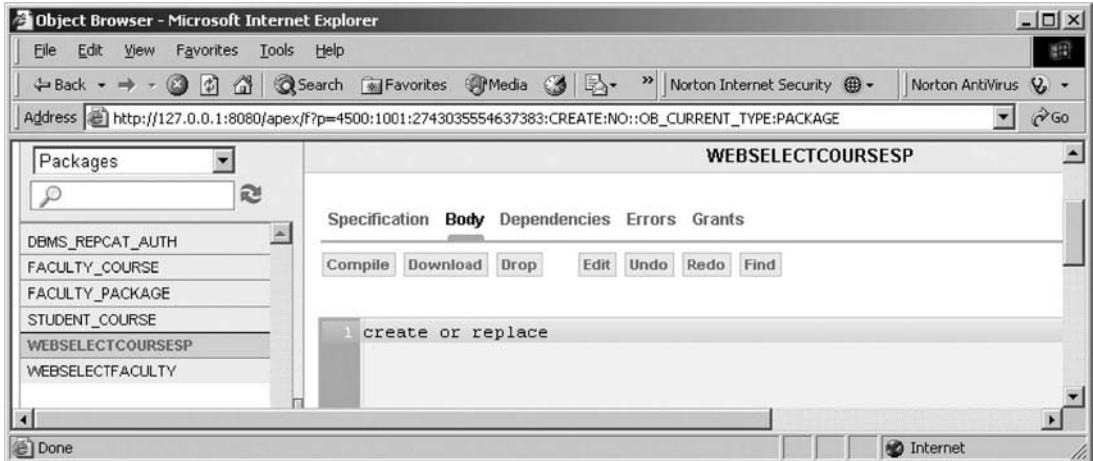


Figure 9.128 Opened Body page of the package.

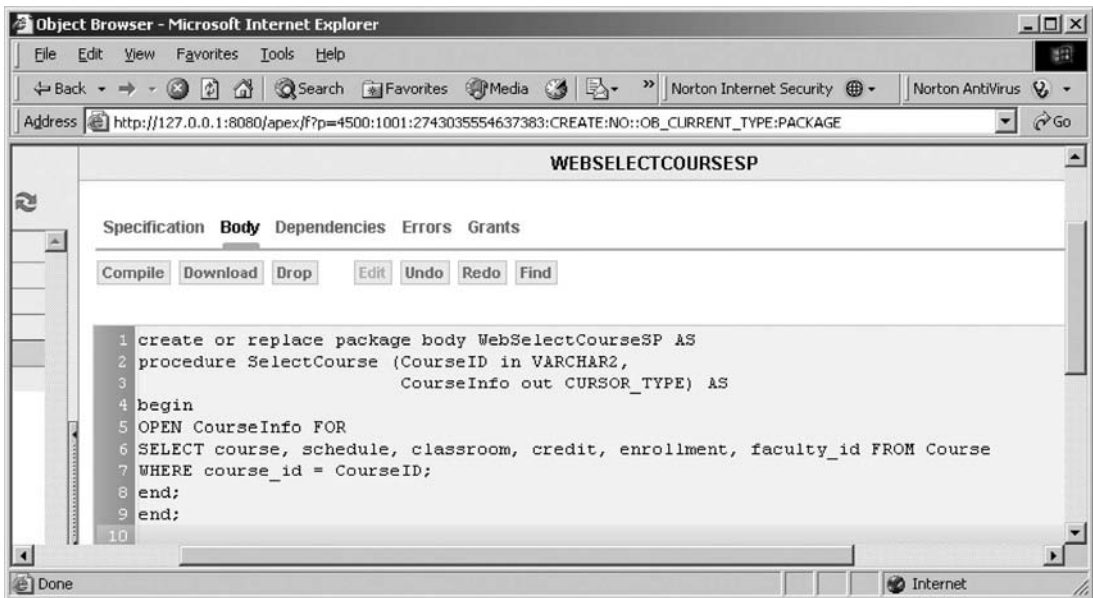


Figure 9.129 Codes for the Body part of the package.

to assign the returned course data columns from the following query to the cursor variable `CourseInfo`. Recall that we used a `SET` command to perform this assignment in the SQL Server stored procedure in Section 5.19.2.7.4 in Chapter 5. There are two **end** commands applied at the end of this Package. The first one is used to end the stored procedure and the second one is for the Package.

Now let's compile our package by clicking on the `Compile` button. A successful compiling information **PL/SQL code successfully compiled (11:27:33)** is displayed if this package is bug free, which is shown in Figure 9.130.

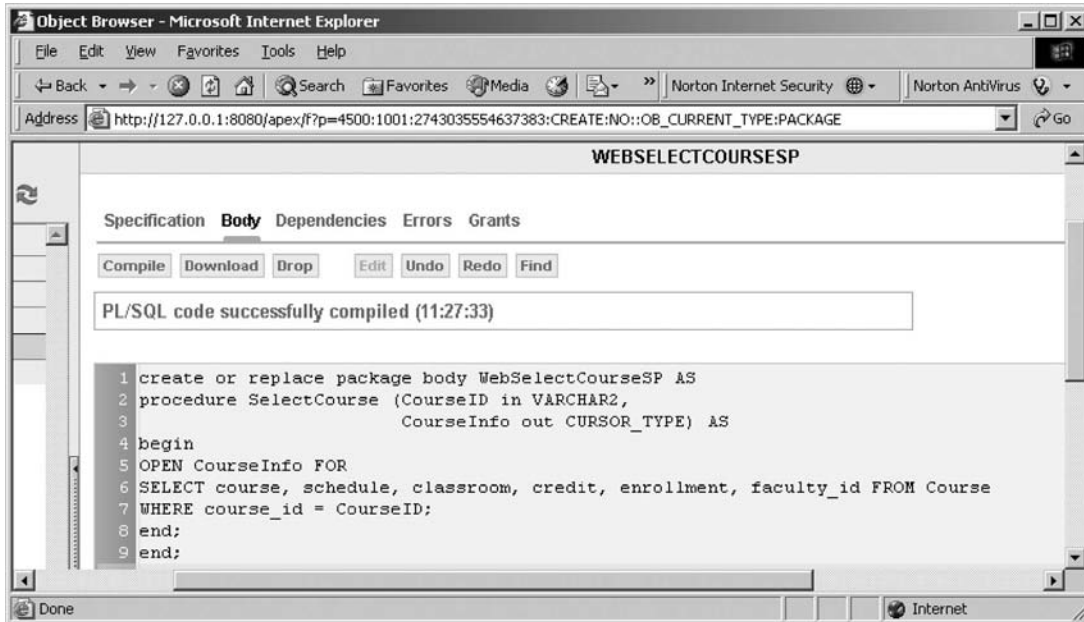


Figure 9.130 Compiled coding for the body part of the package.

At this point, we finished the development of our Oracle package and all modifications to our new Web Service project. Close the Oracle Database 10g XE by clicking on the Close button located at the upper-right corner of the window.

Now let's run our Web Service project to test the data insertion function. Click the Start Debugging button to run our Web Service.

First, let's test the Web method SetOracleInsertSP() by clicking on this method. Enter the input parameters shown in Figure 9.131 into the associated Value boxes in the opened parameter-input page, and click on the Invoke button to run this method. The running result of this Web method is shown in Figure 9.132.

It can be found that the running result of this Web method is fine, and this can be confirmed by the returned status variable <OracleInsertOK> whose value is true. Another way to confirm this data insertion is to open our sample database CSE_DEPT from the Oracle Database 10g XE and open the Course table using the Browser Object to check this new inserted course. An example of our opened Course table is shown in Figure 9.133, and you can find that the new inserted course CSE-575 is located in the last row in this Course table.

Now let's test the second Web method GetOracleInsert(). Close the current running result page by clicking on the Close button located at the upper-right corner of that page, and click on the Back button from the SetOracleInsertSP page to return to the initial default Web interface. Click on the Web method GetOracleInsert to open its parameter-input page. On the opened parameter-input page, enter the faculty name Ying Bai into the Value box and click on the Invoke button to run this method. The running result is shown in Figure 9.134.

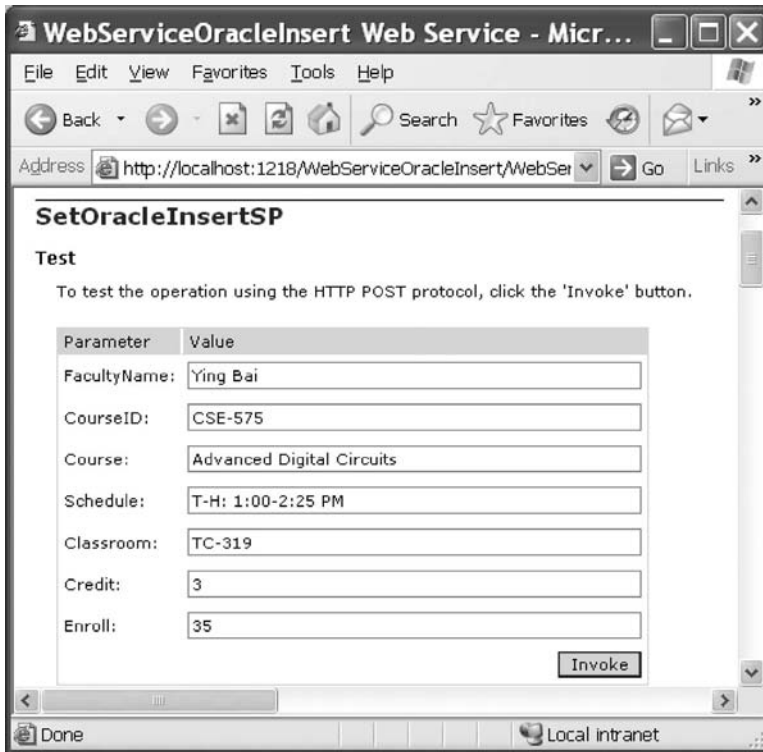


Figure 9.131 Parameter-input built-in Web interface.

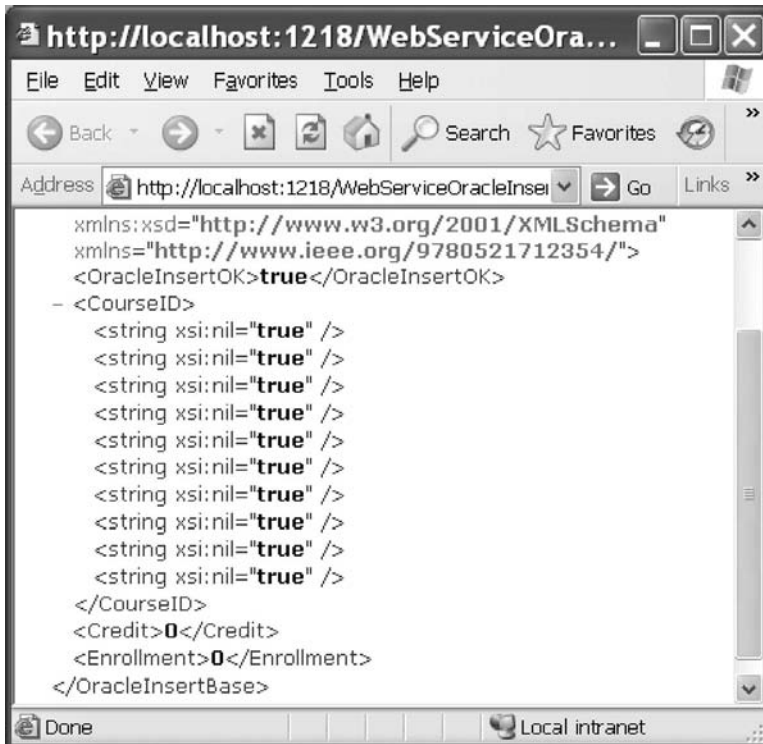
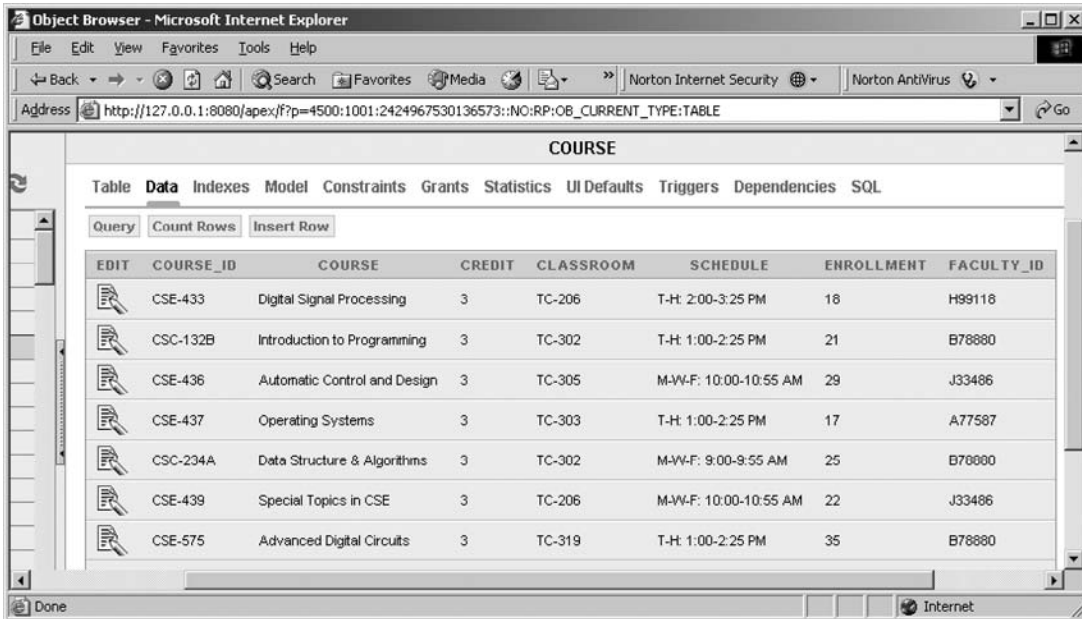


Figure 9.132 Running result of the Web method SetOracleInsertSP.



EDIT	COURSE_ID	COURSE	CREDIT	CLASSROOM	SCHEDULE	ENROLLMENT	FACULTY_ID
	CSE-433	Digital Signal Processing	3	TC-206	T-H: 2:00-3:25 PM	18	H99118
	CSC-132B	Introduction to Programming	3	TC-302	T-H: 1:00-2:25 PM	21	B78880
	CSE-436	Automatic Control and Design	3	TC-305	M-W-F: 10:00-10:55 AM	29	J33486
	CSE-437	Operating Systems	3	TC-303	T-H: 1:00-2:25 PM	17	A77587
	CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B78880
	CSE-439	Special Topics in CSE	3	TC-206	M-W-F: 10:00-10:55 AM	22	J33486
	CSE-575	Advanced Digital Circuits	3	TC-319	T-H: 1:00-2:25 PM	35	B78880

Figure 9.133 Opened Course table.



```

<?xml version="1.0" encoding="utf-8" ?>
- <OracleInsertBase
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.ieee.org/9780521712354/">
  <OracleInsertOK>true</OracleInsertOK>
  - <CourseID>
    <string>CSE-434</string>
    <string>CSE-438</string>
    <string>CSE-575</string>
    <string>CSC-132B</string>
    <string>CSC-234A</string>
    <string xsi:nil="true" />
    <string xsi:nil="true" />
    <string xsi:nil="true" />
    <string xsi:nil="true" />
    <string xsi:nil="true" />
  </CourseID>
  <Credit>0</Credit>
  <Enrollment>0</Enrollment>
</OracleInsertBase>

```

Figure 9.134 Running result of the Web method GetOracleInsert.

It can be found that the new inserted course CSE-575 is indeed added into the Course table, and this data insertion is successful.

Now let's test the third Web method `OracleInsertDataSet()`. Close the current running result page and click on the **Back** button to return to our Web start page. Click on the Web method `OracleInsertDataSet` to test this method. Enter the parameters shown in Figure 9.135 as the new course information into the associated **Value** box in the opened parameter-input page, and click on the **Invoke** button to run this method.

The running result of this Web method is shown in Figure 9.136. It can be found from Figure 9.136 that the new inserted course CSC-532 is indeed added into the Course table in our sample Oracle database.

Finally let's test the Web method `GetOracleInsertCourse()`. Close the current running result page window, and click on the **Back** button to return to our initial Web page. Click on the Web method `GetOracleInsertCourse` to open its parameter-input page.

On the opened page, enter **CSC-532** into the **Value** box as the input parameter, and click on the **Invoke** button to run this method. The running result is shown in Figure 9.137.

It can be found that the detailed information such as the course name, classroom, schedule, credit, and enrollment about the course CSC-532 has been retrieved and displayed in this built-in Web interface in XML tag format.

At this point, we finished testing all Web methods we developed in this Web Service project. A complete Web Service project `WebServiceOracleInsert` can be found at the folder `DBProjects\Chapter 9` located at the accompanying ftp site (see Chapter 1).

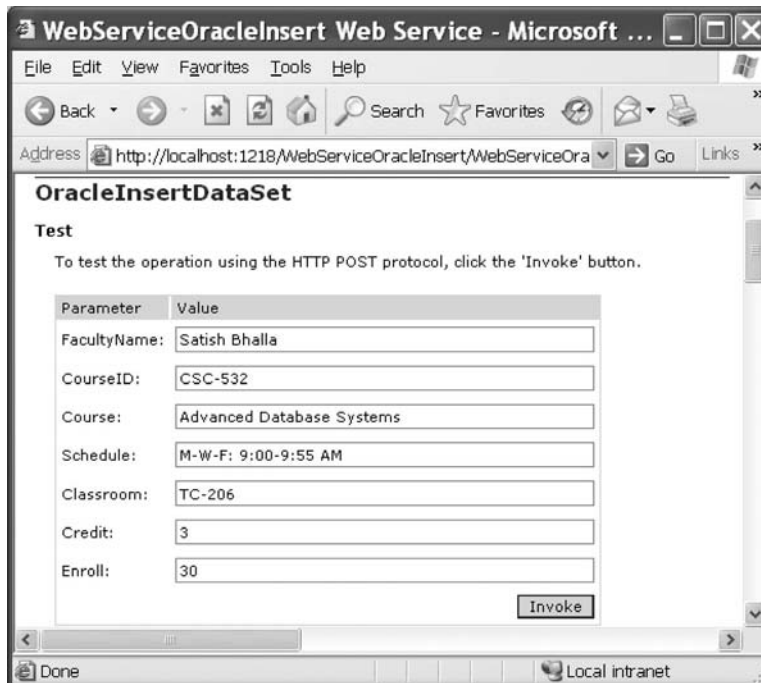


Figure 9.135 Parameter-input page.

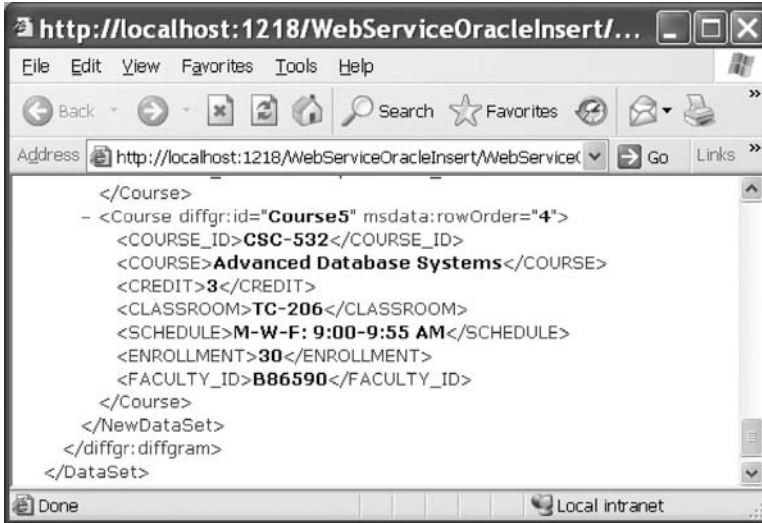


Figure 9.136 Running result of the Web method OracleInsertDataSet.

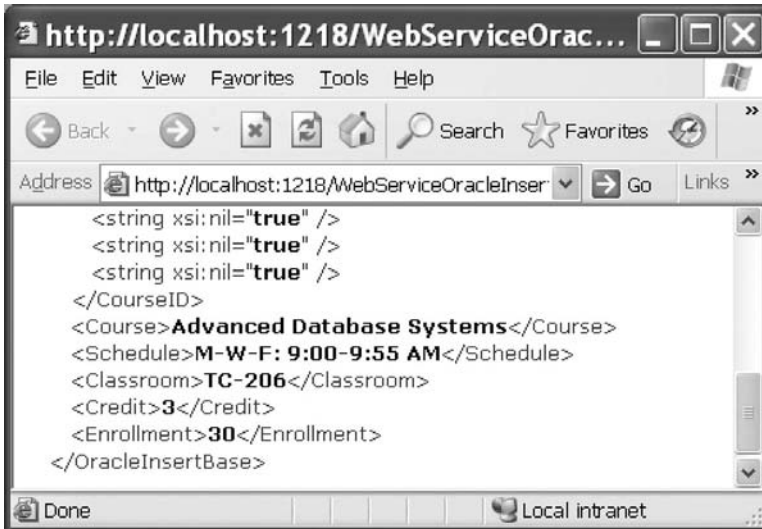


Figure 9.137 Running result of the method GetOracleInsertCourse.

9.11 BUILD WEB SERVICE CLIENTS TO USE WEB SERVICE WEBSERVICEORACLEINSERT

To use this Web Service project, one can develop either a Windows-based or a Web-based Web Service client project. Actually there is no significant difference between building a client project to use a Web Service to access an SQL Server database and building a client project to use a Web Service to access an Oracle database. For example, you can

modify any client project we developed in the previous sections, such as either WinClientSQLInsert or WebClientSQLInsert, to use this Web Service project WebServiceOracleInsert. The main modification is to replace the Web Reference with a new Web Reference class, which is our new developed Web Service class WebServiceOracleInsert.

Follow the modification steps below to complete these changes:

1. Remove the old Web reference from the Windows-based or Web-based client project. You need to first delete the Web reference object and then you can delete the Web_Reference folder from the current project.
2. Add a new Web reference using the Add Web Reference dialog box. Run the target Web Service project first, and then copy the URL from that running Web Service project and paste it to the URL box in the Add Web Reference dialog box in the client project.
3. Change the Web reference name for all data components used in the client project.
4. Change the names of the base class and derived class located in the Web reference.
5. Change the names of all Web methods located in the Web reference.

Two completed client projects, WinClientOracleInsert, which is a Windows based, and WebClientOracleInsert, which is Web-based, can be found at the folder DBProjects\Chapter 9 at the accompanying ftp site (see Chapter 1).

9.12 BUILD ASP.NET WEB SERVICE TO UPDATE AND DELETE DATA FOR ORACLE DATABASE

Basically, the procedure to build an ASP.NET Web Service to update and delete data against the SQL Server database is very similar to the procedure to build an ASP.NET Web Service to update and delete data against the Oracle database. The main differences are listed below:

1. The connection string defined in the Web configuration file `web.config`
2. The namespace directories listed at the top of each Web Service page
3. The stored procedures used by each Web Service page
4. The protocol of the data query string used by each Web Service page
5. The nominal names of dynamic parameters for the Parameters collection object

These five distinct points exist between the procedures to build a Web Service to update and delete data in two kinds of databases.

Based on the discussion and analysis we made in Section 9.8 as well as the similarity between the SQL Server and Oracle databases, we can develop our Web Service projects to update and delete data in the Oracle database by modifying some existing Web Service projects we developed in the previous sections. In this section, we concentrate on modifications to a Web Service project WebServiceSQLUpdateDelete and make it our new Web Service project WebServiceOracleUpdateDelete.

9.12.1 Build Web Service Project WebServiceOracleUpdateDelete

In this section, we will modify an existing Web Service project `WebServiceSQLUpdateDelete` to make it our new Web Service project `WebServiceOracleUpdateDelete`, and enable it to update and delete data in our sample Oracle database.

Open Windows Explorer and create a new folder `Chapter 9` under the root directory if you have not done that. Open Internet Explorer and browse to our desired Web Service project `WebServiceSQLUpdateDelete` at the folder `DBProjects\Chapter 9` located at the accompanying ftp site (see Chapter 1). Copy and paste this project into our new folder `Chapter 9`. Rename it `WebServiceOracleUpdateDelete`. Perform the following modifications to this project in the Windows Explorer window:

1. Change the main Web Service page from `WebServiceSQLUpdateDelete.asmx` to `WebServiceOracleUpdateDelete.asmx`.
2. Open the `App_Code` folder and change the name of our base class file from `SQLBase.cs` to `OracleBase.cs`.
3. Open the `App_Code` folder and change the name of our code-behind page from `WebServiceSQLUpdateDelete.cs` to `WebServiceOracleUpdateDelete.cs`.

Now open Visual Studio.NET 2008 and our new Web Service project `WebServiceOracleUpdateDelete` to perform the associated modifications to the contents of the files we renamed above. First, let's perform modifications to our main Web Service page `WebServiceOracleUpdateDelete.asmx`. Open this page by double-clicking on it from the Solution Explorer window and perform the following modifications:

- Change `CodeBehind="~/App_Code/WebServiceSQLUpdateDelete.cs"` to `CodeBehind="~/App_Code/WebServiceOracleUpdateDelete.cs"`.
- Change `Class="WebServiceSQLUpdateDelete"` to `Class="WebServiceOracleUpdateDelete"`.

Second, open the base class file `OracleBase.cs` and perform the following modifications:

- Change the class name and the constructor's name from `SQLBase` to `OracleBase`.
- Change the name of the first member data from `SQLOK` to `OracleOK`.
- Change the name of the second member data from `SQLException` to `OracleError`.

Go to the `File|Save All` menu item to save these modifications.

9.12.2 Modify Connection String

Double-click on our Web configuration file `web.config` from the Solution Explorer window to open it. Change the content of the connection string that is under the tag `<connectionStrings>` to:

```
<add name="ora_conn" connectionString="Server=XE;
User ID=CSE_DEPT;Password=reback;" />
```

The Oracle database server XE is used for the server name, the user ID is the name of our sample database CSE_DEPT, and the password is determined by the user when adding a new account to create a new user database.

9.12.3 Modify Namespace Directories

First, we need to add an Oracle Data Provider Reference to our Web Service project. To do that, right-click on our new project icon `WebServiceOracleUpdateDelete` from the Solution Explorer window, and then select the item **Add Reference** from the pop-up menu to open the **Add Reference** dialog box. Browse down along the list until you find the item `System.Data.OracleClient`, click to select it and then click on the **OK** button to add it into our project.

Now double-click on our code-behind page `WebServiceOracleUpdateDelete.cs` to open it. On the opened page, add the Oracle namespace to the namespace area on this page.

```
using System.Data.OracleClient;
```

Also change the name of our Web Service class, which is located after the accessing mode `public class`, from `WebServiceSQLUpdateDelete` to `WebServiceOracleUpdateDelete`. Perform the same modification to the constructor's name of this class.

Next we will perform the necessary modifications to four Web methods developed in this Web Service project combined with those five differences listed above.

9.12.4 Modify Web Method `SQLUpdateSP` and Related Methods

The following issues are related to this modification:

1. The name of this Web method and the name of the returned data type class
2. The content of the query string used in this Web method
3. The stored procedure used in this Web method
4. The names of the data components used in this Web method
5. The user-defined `SQLConn()` and `ReportError()` methods
6. The names of the dynamic parameters

Let's perform those modifications step by step according to this sequence. Open the Web method `SQLUpdateSP()` and perform the modifications shown in Figure 9.138 to this method.

Let's take a closer look at these modifications to see how they work.

- A. Rename this Web method `OracleUpdateSP` and change the name of the returned class to `OracleBase`.
- B. Modify the content of the query string by changing the name of the stored procedure from `dbo.WebUpdateCourseSP` to `UpdateCourse_SP`. The former is an SQL Server stored procedure and the latter is an Oracle stored procedure that will be developed in the next section.

WebServiceOracleUpdateDelete	OracleUpdateSP()
	<pre> using System.Data.OracleClient; [WebService(Namespace = "http://www.ieee.org/9780521712354/")] [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)] // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line. // [System.Web.Script.Services.ScriptService] public class WebServiceOracleUpdateDelete : System.Web.Services.WebService { public WebServiceOracleUpdateDelete() { //Uncomment the following line if using designed components //InitializeComponent(); } [WebMethod] A public OracleBase OracleUpdateSP(string FacultyName, string CourseID, string Course, string Schedule, B string Classroom, int Credit, int Enroll) C { string cmdString = "UpdateCourse_SP"; OracleConnection oraConnection = new OracleConnection(); OracleBase OracleResult = new OracleBase(); OracleCommand oraCommand = new OracleCommand(); int intUpdate = 0; D OracleResult.OracleOK = true; E oraConnection = OracleConn(); F if (oraConnection == null) { OracleResult.OracleError = "Database connection is failed"; ReportError(OracleResult); return null; } oraCommand.Connection = oraConnection; oraCommand.CommandType = CommandType.StoredProcedure; G oraCommand.CommandText = cmdString; oraCommand.Parameters.Add("FacultyName", OracleType.VarChar).Value = FacultyName; oraCommand.Parameters.Add("inCourseID", OracleType.Char).Value = CourseID; oraCommand.Parameters.Add("inCourse", OracleType.VarChar).Value = Course; oraCommand.Parameters.Add("inSchedule", OracleType.Char).Value = Schedule; oraCommand.Parameters.Add("inClassroom", OracleType.VarChar).Value = Classroom; oraCommand.Parameters.Add("inCredit", OracleType.Int32).Value = Credit; H oraCommand.Parameters.Add("inEnroll", OracleType.Int32).Value = Enroll; intUpdate = oraCommand.ExecuteNonQuery(); oraConnection.Close(); oraCommand.Dispose(); if (intUpdate == 0) { I OracleResult.OracleError = "Data updating is failed"; ReportError(OracleResult); } return OracleResult; } } </pre>

Figure 9.138 Modified Web method OracleUpdateSP.

- C.** Change the prefix from `Sql` to `Oracle` for all data classes and from `sql` to `ora` for all data objects. Also change the returned instance name from `SQLResult` to `OracleResult`.
- D.** Change the name of the returned instance from `SQLResult` to `OracleResult` and member data from `SQLOK` to `OracleOK`.
- E.** Change the name of the subroutine from `SQLConn` to `OracleConn`.
- F.** Change the prefix from `sql` to `ora` for all data objects.

- G. Modify the nominal names for all seven input parameters to the stored procedure by removing the @ symbol before each nominal name. Also change the data type of the top five input parameters from `SqlDbType.Text` to `OracleType.VarChar`. Change the data type for the last two input parameters from `SqlDbType.Int` to `OracleType.Int32`. The prefix **in** for the last six parameters matches the input parameters used for the stored procedure `UpdateCourse_SP()`.
- H. Change the prefix from `sql` to `ora` for all data objects.
- I. Change the name of the returned instance from `SQLResult` to `OracleResult` and change the second member data from `SQLError` to `OracleError`.

Your modified Web method `OracleUpdateSP()` is shown in Figure 9.138. All modified parts have been highlighted in bold.

Next let's perform modifications to two related user-defined methods `SQLConn()` and `ReportError()`.

Perform the following modifications to the `SQLConn()` method:

- A. Change the name of this method from `SQLConn` to `OracleConn` and return class name from `SqlConnection` to `OracleConnection`. Also change the connection string from `sql_conn` to `ora_conn`.
- B. Change the data type of the returned connection object to `OracleConnection`.

Perform the following modifications to the `ReportError()` method:

- C. Change the data type of the passed argument from `SQLBase` to `OracleBase`.
- D. Change the name of the first member data from `SQLOK` to `OracleOK`.
- E. Change the name of the second member data from `SQLError` to `OracleError`.

Your modified methods `OracleConn()` and `ReportError()` should match that shown in Figure 9.139. All modified parts have been highlighted in bold.

Next let's develop the stored procedure `UpdateCourseSP` to perform the course updating function.

```

A protected OracleConnection OracleConn()
    {
        string cmdString = ConfigurationManager.ConnectionStrings["ora_conn"].ConnectionString;
        B OracleConnection conn = new OracleConnection();
        conn.ConnectionString = cmdString;
        conn.Open();
        if (conn.State != System.Data.ConnectionState.Open)
        {
            MessageBox.Show("Database Open is failed");
            conn = null;
        }
        return conn;
    }

C protected void ReportError(OracleBase ErrSource)
    {
        D ErrSource.OracleOK = false;
        E MessageBox.Show(ErrSource.OracleError);
    }
  
```

Figure 9.139 Modified methods `OracleConn` and `ReportError`.

9.12.4.1 Develop Stored Procedure UpdateCourse_SP

A very detailed discussion of creating and manipulating packages and stored procedures in Oracle database is provided in Section 5.20.3.5 in Chapter 5. Refer to that section to get more detailed information for creating Oracle's stored procedures.

The topic we discuss in this section is to update and delete data in the database. Therefore no returned data is needed for these two data actions. We only need to create stored procedures in Oracle database, not packages, to perform the data updating and deleting function.

As discussed in Section 5.20.3.6 in Chapter 5, different methods can be used to create Oracle's stored procedures. In this section, we will use the Object Browser page provided by Oracle Database 10g XE to create our stored procedures.

Open the Oracle Database 10g XE home page by going to **Start|All Programs|Oracle Database 10g Express Edition|Go To Database Home Page** items. Finish the login process by entering the correct username and password such as **CSE_DEPT** and **reback**. Click on the Object Browser and select **Create|Procedures** item to open the Create Procedure window. Click on the **Create** button and select the Procedure icon from the list to open this window.

Enter **UpdateCourse_SP** into the Procedure Name box and keep the Include Argument checkbox checked, and click on the **Next** button to go to the next page.

The next window is used to allow us to enter all input parameters. For this stored procedure we need to perform two queries. Therefore we have seven input parameters. The first query is to get the `faculty_id` from the Faculty table based on the faculty name that is an input and selected by the user. The second query is to update a course record that contains six pieces of information related to a current course in the Course table based on the `faculty_id` that is obtained from the first query. The seven input parameters are **FacultyName**, **CourseID**, **CourseTitle**, **Credit**, **Classroom**, **Schedule**, and **Enrollment**. The first input parameter **FacultyName** is used by the first query, and the following six input parameters are used by the second query.

Enter those input parameters one by one into the argument box. The point is that the data type of each input parameter must be identical with the data type of each data column used in the Course table. Refer to Section 2.11.2.3 in Chapter 2 to get a detailed list of data types used for those data columns in the Course data table.

For the Input/Output parameter definitions, select **IN** for all seven parameters since no output is needed for this data updating query. Your finished argument list should match that shown in Figure 9.140.

Click on the **Next** button to go to the procedure defining page. Enter the codes shown in Figure 9.141 into this new procedure as the body of the procedure using the language of so-called Procedural Language Extension for SQL or PL-SQL. Then click on the **Next** and on the **Finish** buttons to confirm creating this procedure. Your finished stored procedure should match that shown in Figure 9.142.

Seven input parameters are listed at the beginning of this procedure with the keyword **IN** to indicate that these parameters are inputs to the procedure. The intermediate parameter `faculty_id` is obtained from the first query from the Faculty table. The data type of each parameter is indicated after the keyword **IN**, and it must be identical with the data type of the associated data column in the Course table. An **IS** command is

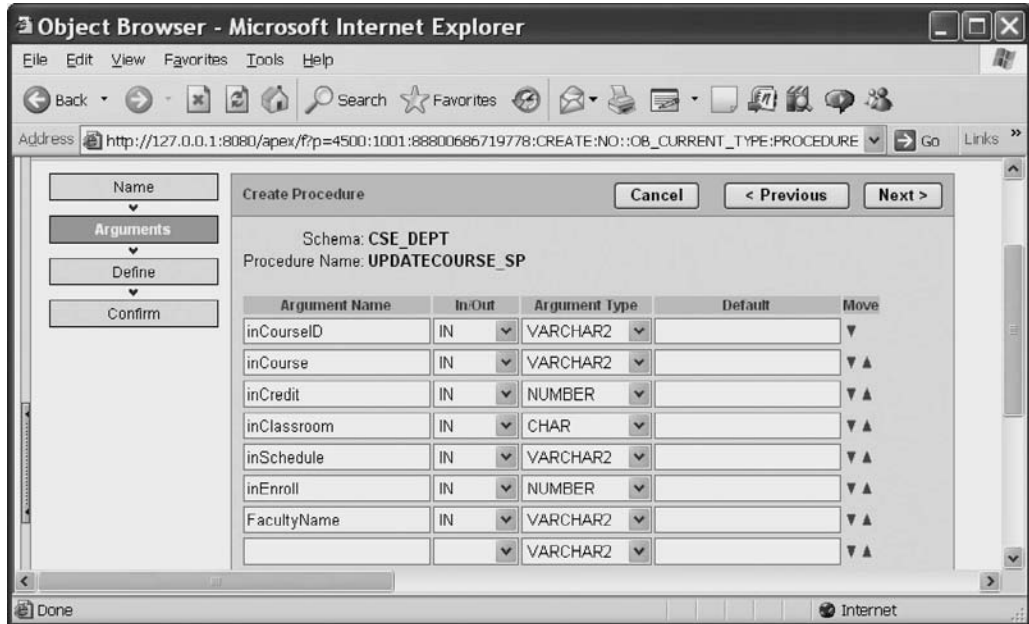


Figure 9.140 Finished argument list.

```

SELECT faculty_id INTO FacultyID FROM Faculty
WHERE faculty_name = FacultyName;
UPDATE Course SET course=inCourse, credit=inCredit, classroom=inClassroom,
schedule=inSchedule, enrollment=inEnroll, faculty_id=FacultyID
WHERE course_id=inCourseID;

```

Figure 9.141 Stored procedure body.

attached after the procedure header to indicate that an intermediate query result, `faculty_id`, will be held by a local variable `facultyID` declared later.

Two queries are included in this procedure. The first query is used to get the `faculty_id` from the `Faculty` table based on the input parameter `FacultyName`, and the second query is to update a course record based on six input parameters in the `Course` table. A semicolon must be attached after each PL-SQL statement.

One important issue is that you need to create one local variable `FacultyID` and attach it after the **IS** command as shown in line 9 in Figure 9.142, and this coding line has been highlighted with shading. Click on the **Edit** button to add this local variable with its data type of `VARCHAR2(10)`. This local variable is used to hold the returned `faculty_id` from the execution of the first query.

Another important issue in arranging the input parameters or arguments in the **UPDATE** command is that the order of those parameters or arguments must be identical with the order of the columns in the associated data table. For example, in the `Course` table, the order of the data columns is `course_id`, `course`, `credit`, `classroom`, `schedule`, `enrollment`, and `faculty_id`. Accordingly, the order of input parameters

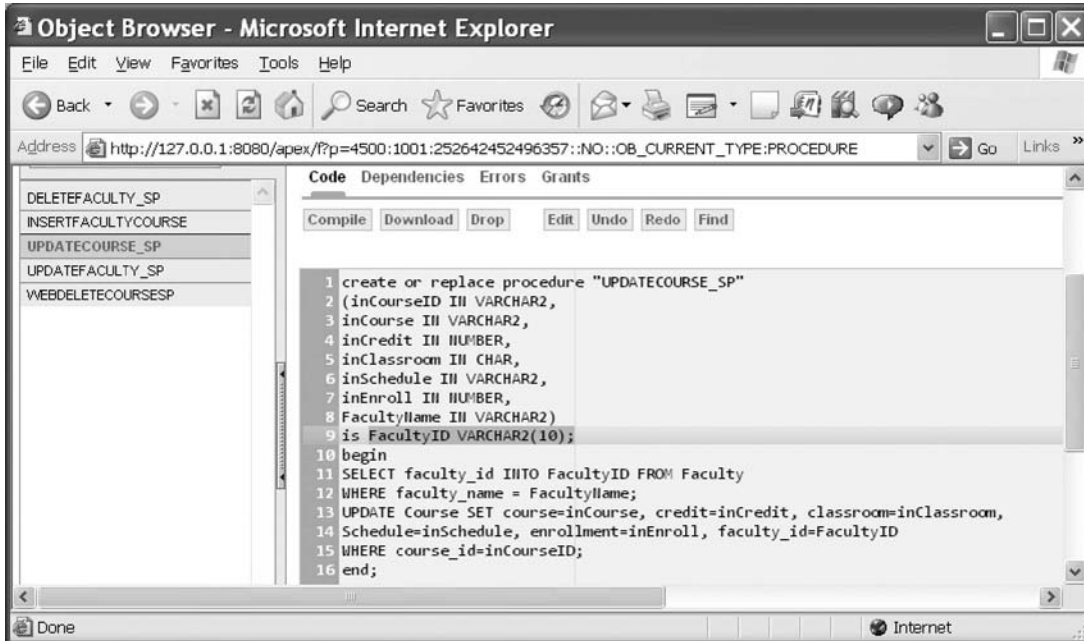


Figure 9.142 Completed stored procedure.

placed in the UPDATE argument list must be identical with the data columns' order displayed above.

To make sure that this procedure works properly, we need to compile it first. Click on the **Compile** button to compile and check our procedure. A successful compilation message should be displayed if our procedure is a bug-free stored procedure. Close the Oracle Database 10g Express Edition by clicking the **Close** button.

9.12.5 Modify Web Method GetSQLCourse and Related Methods

The function of this Web method is to retrieve all `course_id`, which includes the original and the new inserted `course_id`, from the Course table based on the input faculty name. This Web method will be called or consumed by a client project later to get back and display all `course_id` in a listbox control in the client project.

Recall that in Section 5.19.2.5 in Chapter 5, we developed a joined-table query to perform the data query from the Course table to get all `course_id` based on the faculty name. The reason for that is because there is no faculty name column available in the Course table, and each course or `course_id` is related to a `faculty_id` in the Course table. In order to get the `faculty_id` that is associated with the selected faculty name, one must first go to the Faculty table to perform a query to obtain it. In this situation, a join query is a desired method to complete this function.

Open this Web method and perform the following modifications that are shown in Figure 9.143 to this method. All modified parts have been highlighted in bold.


```

[WebMethod]
A public OracleBase GetOracleCourse(string FacultyName)
  {
B     string cmdString = "SELECT Course.course_id FROM Course, Faculty " +
C     "WHERE (Course.faculty_id = Faculty.faculty_id) AND (Faculty.faculty_name =: fname)";
C     OracleConnection oraConnection = new OracleConnection();
C     OracleBase OracleResult = new OracleBase();
C     OracleCommand oraCommand = new OracleCommand();
C     OracleDataReader oraReader;
C     OracleResult.OracleOK = true;
D     oraConnection = OracleConn();
E     if (oraConnection == null)
    {
        OracleResult.OracleError = "Database connection is failed";
        ReportError(OracleResult);
        return null;
    }
    oraCommand.Connection = oraConnection;
    oraCommand.CommandType = CommandType.Text;
    oraCommand.CommandText = cmdString;
F     oraCommand.Parameters.Add("fname", OracleType.VarChar).Value = FacultyName;
    oraReader = oraCommand.ExecuteReader();
    if (oraReader.HasRows == true)
G     FillCourseReader(ref OracleResult, oraReader);
    else
H     {
        OracleResult.OracleError = "No matched course found";
        ReportError(OracleResult);
    }
    oraReader.Close();
    oraConnection.Close();
    oraCommand.Dispose();
    return OracleResult;
  }

```

Figure 9.143 Modified Web method GetOracleCourse.

Let's take a closer look at these modifications to see how they work.

- A.** Change the name of this Web method from GetSQLCourse to GetOracleCourse. Also change the name of the returned instance from SQLBase to OracleBase.
- B.** Modify the query string to match it to the ANSI 89 standard. Recall that we developed a join-table query string for SQL Server database using the ANSI 92 standard in Section 5.19.2.5 in Chapter 5. Since the ANSI 89 standard is still being used in the Oracle database, we need to modify this joined-table query string by using that standard.
- C.** Change the prefix of all data classes from Sql to Oracle and from sql to ora for all data objects used in this method. Also change the name of the returned instance from SQLResult to OracleResult. Change the first member data from SQLOK to OracleOK.
- D.** Change the name of the user-defined method from SQLConn to OracleConn. Change the second member data from SQLError to OracleError.
- E.** Change the prefix of all data objects from sql to ora. Change the second member data from SQLError to OracleError.
- F.** Modify the nominal name for the input parameter to the stored procedure by removing the @ symbol before the nominal name fname. Also change the data type of this input parameter from SqlDbType.Text to OracleType.VarChar.


```

protected void FillCourseReader(ref OracleBase sResult, OracleDataReader sReader)
{
    int pos = 0;
    while (sReader.Read())
    {
        sResult.CourseID[pos] = Convert.ToString(sReader.GetOracleString(0)); //the 1st column is course_id
        pos++;
    }
}

```

Figure 9.144 Modified method FillCourseReader.

- G.** Change the names of two passed arguments to the method FillCourseReader() from SQLResult to OracleResult and from sqlReader to oraReader.
- H.** Change the name of the returned instance from SQLResult to OracleResult, change the second member data from SQLError to OracleError, and change the prefix for all data objects from sql to ora.

The modifications to the related user-defined FillCourseReader() method are relatively simple. Perform the following modifications to this method:

- A.** Modify the data types of two passed arguments by changing the data type of the first argument from SQLBase to OracleBase and changing the data type of the second argument from SqlDataReader to OracleDataReader.
- B.** Change the method from GetSQLString(0) to GetOracleString(0).

Your modified user-defined FillCourseReader() method should match that shown in Figure 9.144. All modified parts have been highlighted in bold. Next let's modify another Web method GetSQLCourseDetail().

9.12.6 Modify Web Method GetSQLCourseDetail and Related Methods

The function of this Web method is to retrieve the detailed information for a selected course_id that works as an input parameter to this method, and store the retrieved information to an instance that will be returned to the calling procedure.

The following three modifications need to be performed for this Web method:

1. Modify the codes of this Web method.
2. Modify the related user-defined FillCourseDetail() method.
3. Modify the content of the query string and make it equal to the name of an Oracle Package WebSelectCourseSP we developed in Section 9.10.7.

Open this Web method and perform the modifications shown in Figure 9.145 to this Web method.

Let's take a closer look at these modifications to see how they work.

- A.** Change the name of this Web method from GetSQLCourseDetail to GetOracleCourseDetail. Also change the name of the returned base class from SQLBase to OracleBase.

WebServiceOracleUpdateDelete		GetOracleCourseDetail()
	[WebMethod]	
A	public OracleBase	GetOracleCourseDetail(string CourseID)
	{	
B	string cmdString = "WebSelectCourseSP.SelectCourse";	
C	OracleConnection oraConnection = new OracleConnection();	
	OracleBase OracleResult = new OracleBase();	
D	OracleDataReader oraReader;	
	OracleParameter paramCourseID = new OracleParameter();	
	OracleParameter paramCourseInfo = new OracleParameter();	
E	OracleResult.OracleOK = true;	
	oraConnection = OracleConn();	
	if (oraConnection == null)	
	{	
	OracleResult.OracleError = "Database connection is failed";	
	ReportError(OracleResult);	
	return null;	
	}	
F	paramCourseID.ParameterName = "CourseID";	
	paramCourseID.OracleType = OracleType.VarChar;	
	paramCourseID.Value = CourseID;	
	paramCourseInfo.ParameterName = "CourseInfo";	
	paramCourseInfo.OracleType = OracleType.Cursor;	
	paramCourseInfo.Direction = ParameterDirection.Output; //this is very important	
G	OracleCommand oraCommand = new OracleCommand(cmdString, oraConnection);	
	oraCommand.CommandType = CommandType.StoredProcedure;	
	oraCommand.Parameters.Add(paramCourseID);	
	oraCommand.Parameters.Add(paramCourseInfo);	
	oraReader = oraCommand.ExecuteReader();	
H	if (oraReader.HasRows == true)	
	FillCourseDetail(ref OracleResult, oraReader);	
	else	
	{	
I	OracleResult.OracleError = "No matched course found";	
	ReportError(OracleResult);	
	}	
	oraReader.Close();	
	oraConnection.Close();	
	oraCommand.Dispose();	
	return OracleResult;	
	}	

Figure 9.145 Modified Web method GetOracleCourseDetail.

- B. Modify the content of the query string and make it equal to the name of the Package we developed in the Section 9.10.7. Change this query string from `dbo.WebSelectCourseSP` to `WebSelectCourseSP.SelectCourse`. The prefix `WebSelectCourseSP` is a Package and the `SelectCourse` is a stored procedure.
- C. Change the prefix of all data classes from `Sql` to `Oracle` and from `sql` to `ora` for all data objects used in this method. Also change the name of the returned instance from `SQLResult` to `OracleResult`. Change the first member data from `SQLOK` to `OracleOK`.
- D. Add two `OracleParameter` objects `paramCourseID` and `paramCourseInfo`. Because some differences exist between the SQL Server and Oracle databases, we need to use different ways to assign parameters to the `Parameters` collection of the `Command` object later.
- E. Change the name of the user-defined method from `SQLConn` to `OracleConn`. Change the second member data from `SQLException` to `OracleError`.

- F. Initialize two `OracleParameter` objects by assigning them with the appropriate values. The point is, for the second parameter `paramCourseInfo`, the data type of this parameter is `Cursor` and the `Direction` is `Output`. Both values are very important to this parameter and must be assigned exactly as we did here. Otherwise a running exception will be encountered when the project runs.
- G. Add two statements to add two `OracleParameter` objects to the `Command` object.
- H. Change the names of two passed arguments to the `FillCourseDetail()` method from `SQLResult` to `OracleResult` and from `sqlReader` to `oraReader`.
- I. Change the name of the returned instance from `SQLResult` to `OracleResult` and change the second member `data` from `SQLException` to `OracleError`.

The modifications to the related user-defined `FillCourseDetail()` method are simple. The only modifications are to change the data type of two passed arguments `sResult` and `sReader`. Change the data type of the first argument from `SQLBase` to `OracleBase`, and change the data type for the second argument from `SqlDataReader` to `OracleDataReader`.

9.12.7 Modify Web Method `SQLDeleteSP`

As we discussed in Section 7.1.1 in Chapter 7, to delete a record from a relational database, one needs to follow the operation steps listed below:

1. Delete records that are related to the parent table using the foreign keys from child tables.
2. Delete records that are defined as primary keys from the parent table.

In other words, to delete one record from the parent table, all records that are related to that record as foreign keys and located at different child tables must be deleted first. In our case, in order to delete a record using the `course_id` as the primary key from the `Course` table (parent table), one must first delete those records using the `course_id` as a foreign key from the `StudentCourse` table (child table). Fortunately we have only one child table related to our parent table in our sample database. Refer to Section 2.5 and Figure 2.5 in Chapter 2 to get a clear relationship description among different data tables in our sample database.

From this discussion, it can be found that to delete a course record from our sample database two deleting queries should be performed: The first query is used to delete the related records from the child table or the `StudentCourse` table, and the second query is used to delete the target record from the parent table or the `Course` table. Of course, we can place these two queries into a stored procedure `WebDeleteCourseSP()`, which we will develop in the following section to perform this deleting action. However, recall that in Section 2.11.3.5 in Chapter 2, we set a one-to-many constraint relationship between the `Course` and the `StudentCourse` tables with an `On Delete Cascade` mode (refer to Figure 2.65). This mode means that all records with a `course_id` that is equal to a `course_id` in the `Course` (parent) table in the `StudentCourse` (child) table will be deleted if that `course_id` is deleted from the `Course` (parent) table. With this `On Delete Cascade` mode, we can use a single statement in this stored procedure to only delete a `course_id` from the `Course` (parent) table, and all related records in the `StudentCourse` (child) table will also be deleted automatically by the database engine.

```

[WebMethod]
A public OracleBase OracleDeleteSP(string CourseID)
{
B     string cmdString = "WebDeleteCourseSP";
C     OracleConnection oraConnection = new OracleConnection();
     OracleBase OracleResult = new OracleBase();
     int intDelete = 0;

D     OracleResult.OracleOK = true;
     oraConnection = OracleConn();
     if (oraConnection == null)
     {
         OracleResult.OracleError = "Database connection is failed";
         ReportError(OracleResult);
         return null;
     }
     OracleCommand oraCommand = new OracleCommand(cmdString, oraConnection);
E     oraCommand.CommandType = CommandType.StoredProcedure;
     oraCommand.Parameters.Add("CourseID", OracleType.VarChar).Value = CourseID;
     intDelete = oraCommand.ExecuteNonQuery();

F     if (intDelete == 0)
     {
         OracleResult.OracleError = "Data deleting is failed";
         ReportError(OracleResult);
     }
     oraConnection.Close();
     oraCommand.Dispose();
     return OracleResult;
}

```

Figure 9.146 Modified Web method OracleDeleteSP.

A single input parameter `course_id` is passed into this stored procedure as the primary key.

Now open this Web method and perform the modifications shown in Figure 9.146 to this Web method. All modified parts have been highlighted in bold.

Let's take a closer look at this piece of modified code to see how it works.

- A. Change the name of this Web method from `SQLDeleteSP` to `OracleDeleteSP` and change the returned data type from `SQLBase` to `OracleBase`.
- B. The content of the query string is equal to the name of the stored procedure `WebDeleteCourseSP` we will develop in the next section. Change the name of the stored procedure from `dbo.WebDeleteCourseSP` to `WebDeleteCourseSP`.
- C. Change the prefix from `Sql` to `Oracle` for all data classes and from `sql` to `ora` for all data objects used in this method. Also change the name of the returned instance from `SQLResult` to `OracleResult`. Change the first member data from `SQLOK` to `OracleOK`.
- D. Change the name of the user-defined method from `SQLConn` to `OracleConn`. Change the second member data from `SQLError` to `OracleError`.
- E. Modify the nominal name for the input parameter to the stored procedure by removing the `@` symbol before the nominal name `CourseID`. Also change the data type of this input parameter from `SqldbType.Text` to `OracleType.VarChar`.
- F. Change the name of the returned instance from `SQLResult` to `OracleResult`. Also change the second member data from `SQLError` to `OracleError` and the prefix from `sql` to `ora` for all data objects.

9.12.7.1 Develop Stored Procedure WebDeleteCourseSP

The topic we are discussing in this section is to delete data against the database. Therefore, no returned data is needed for this kind of data action, and we only need to create stored procedures, not packages, in Oracle database to perform the data deleting function.

As we discussed in Section 5.20.3.6 in Chapter 5, different methods can be used to create Oracle's stored procedures. In this section, we will use the Object Browser page provided by Oracle Database 10g XE to create our stored procedures.

Open the Oracle Database 10g XE home page by going to **Start|All Programs|Oracle Database 10g Express Edition|Go To Database Home Page** items. Finish the login process by entering the correct username and password such as **CSE_DEPT** and **reback**. Click on the **Object Browser** and select the **Create|Procedures** item to open the **Create Procedure** window. Click on the **Create** button and select the **Procedure** icon from the list to open this window.

Enter **WebDeleteCourseSP** into the **Procedure Name** box, keep the **Include Argument** checkbox checked, and click on the **Next** button to go to the next page. The next window allows us to enter input parameters. For this stored procedure we need only one input parameter **CourseID**. Enter this input parameter into the argument box. The point is that the data type of this input parameter must be identical with the data type of the data column **course_id** used in the **Course** table. Refer to Section 2.11.2.3 in Chapter 2 to get a detailed list of data types used for those data columns in the **Course** data table.

For the **Input/Output** parameters definition, select **IN** for this input parameter since no output is needed for this data deleting query. Your finished argument list should match that shown in Figure 9.147.

Click on the **Next** button to go to the procedure defining page. Enter the codes shown in Figure 9.148 into this new procedure as the body of the procedure using the language of so-called **Procedural Language Extension for SQL** or **PL-SQL**. Then click on the **Next** and the **Finish** buttons to confirm creation of this procedure. Your finished stored procedure should match that shown in Figure 9.149.

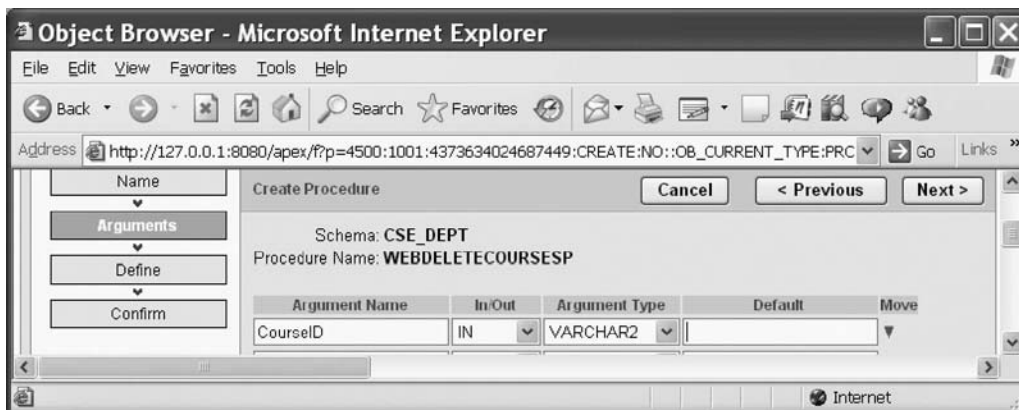


Figure 9.147 Argument list.

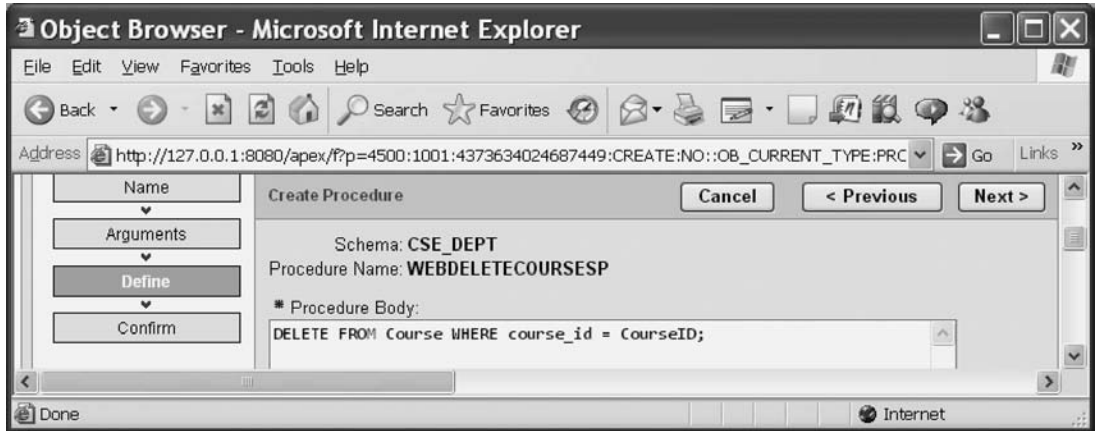


Figure 9.148 Coding body of the stored procedure.

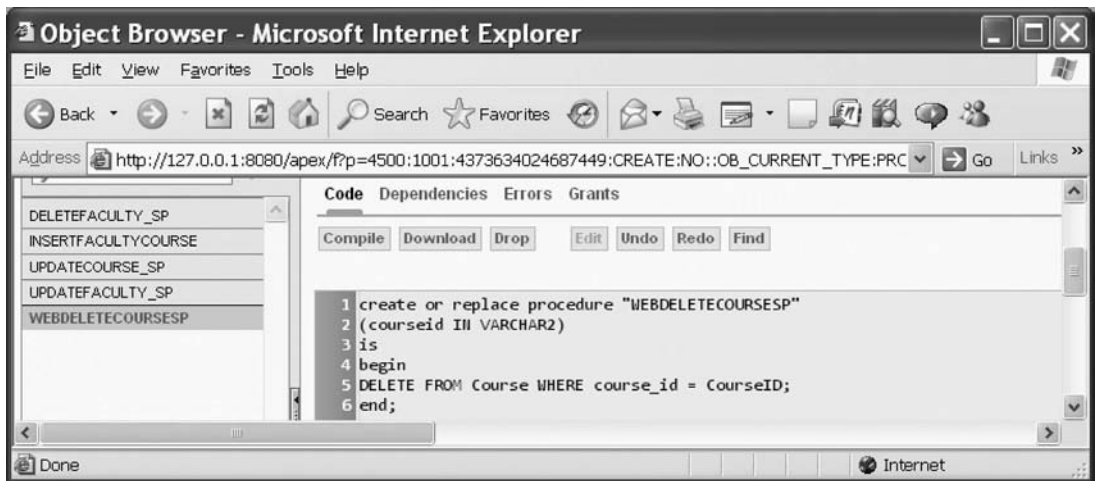


Figure 9.149 Completed coding body of the stored procedure.

A single query is included in this procedure. Because of the On Delete Cascade mode, as this query is executed, the record with a primary key `course_id` that equals to the input parameter to this procedure in the `Course` (parent) table is deleted. All records that have a foreign key `course_id` that is equal to the `course_id` in the `Course` table will also be deleted from the `StudentCourse` (child) table.

To make sure that this procedure works properly, we need to compile it first. Click on the **Compile** button to compile and check our procedure. A successful compilation message should be displayed if our procedure is a bug-free stored procedure. Close the Oracle Database 10g Express Edition by clicking on the **Close** button.

At this point, we have finished all modifications to our new Web Service project `WebServiceOracleUpdateDelete`. Now it is the time for us to run this project to test the data updating and deleting function.

9.12.7.2 Implement and Test Web Service Project

Click on the Start Debugging button to run the project. First, let's test the Web method OracleUpdateSP() to update a course record CSE-665 that is taught by the faculty member Ying Bai against our sample database. To do that, let's first check the original detailed information of this course by running the Web method GetOracleCourseDetail() by clicking on it from the opened built-in Web interface. On the opened parameter-input page, enter CSE-665 into the Value box as the course_id and click on the Invoke button to retrieve the detailed information for this course. The running result of this method is shown in Figure 9.150.

Keep in mind the detailed information for this course and let's now try to update this course by running the Web method OracleUpdateSP(). To do that, close the current running result page and click on The Back button to return to the initial Web page. Click on the Web method OracleUpdateSP to open its parameter-input page. Enter the updating course information shown in Figure 9.151 into the associated Value boxes.

Click on the Invoke button to run this method. From the running result window, it can be found that the member data OracleOK is true, which means that this data updating is successful. Close the current running result window.

To confirm this course updating, click on the Back button to return to the initial page, and click on the Web method GetOracleCourseDetail to try to get back the detailed information for that updated course to validate this data updating. Enter CSE-665 into the CourseID box and click on the Invoke button to run this method. The running result of this method is shown in Figure 9.152. Compare this running result with the one shown in Figure 9.150. It can be found that this course has been updated.

Close the current running result window and click on the Back button to return to the initial page. Next let's test the Web method OracleDeleteSP().

Click on this method and enter a course_id for which you want to delete from the Course table, such as CSE-665, into the CourseID Value box, and click on the Invoke button to perform this data deleting. It can be found from the running result that the member data OracleOK is true, which means that this data deleting is successful. Close the current running result window.

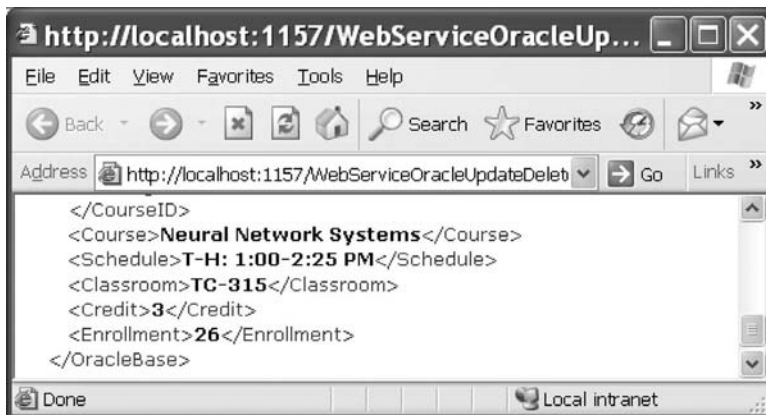


Figure 9.150 Running result of the Web method GetOracleCourseDetail.



Figure 9.151 Parameter-input page.

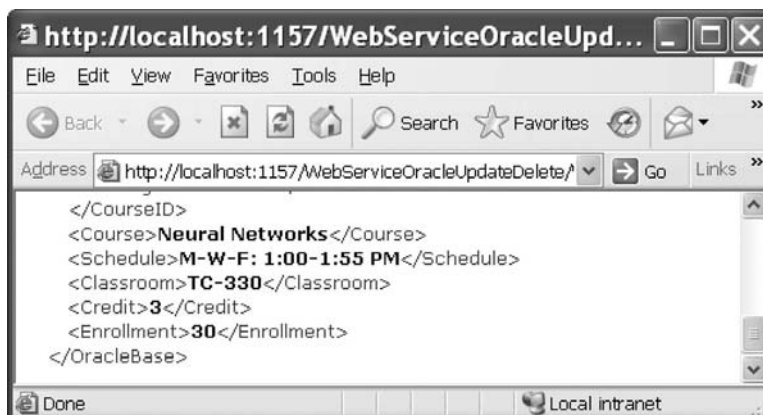


Figure 9.152 Running result of the Web method GetOracleCourseDetail.

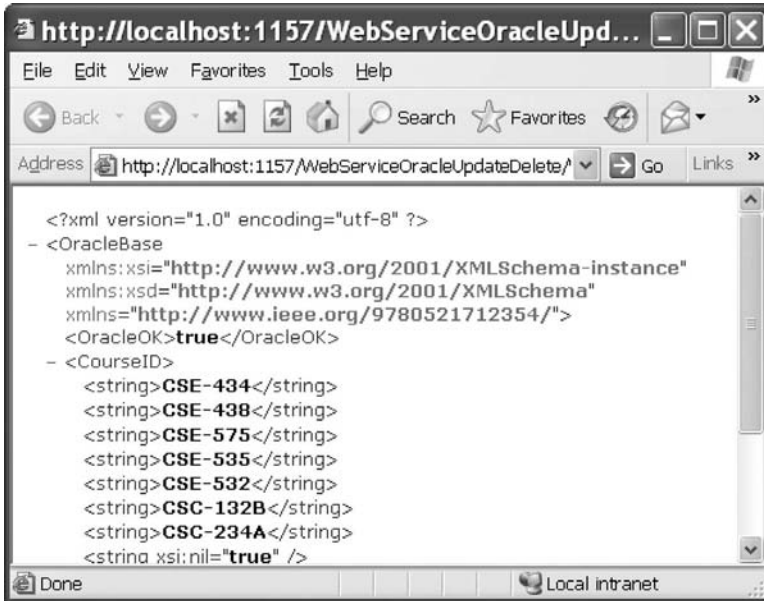


Figure 9.153 Running result of the Web method GetOracleCourse.

To confirm this data deleting, let's run the Web method GetOracleCourse() to retrieve back all courses taught by the selected faculty. Recall that the course CSE-665 was taught by the faculty member Ying Bai. In the initial Web page, click on this method to run it. Enter the faculty name Ying Bai in the FacultyName box and click on the Invoke button to run this Web method. The running result is shown in Figure 9.153.

From this running result, it can be found that the course CSE-665 has been deleted from the Course table successfully. To keep our sample database neat and complete, it is highly recommended to recover this deleted course record. Refer to Table 9.7 in Section 9.7.3.5 in this chapter to recover this course. You can perform this recovery job by opening the Oracle Database 10g XE and using the Insert Row button.

At this point, we finished testing all Web methods we developed in this Web Service project. A complete Web Service project WebServiceOracleUpdateDelete can be found at the folder DBProjects\Chapter 9 at the accompanying ftp site (see Chapter 1).

9.13 BUILD WEB SERVICE CLIENTS TO USE WEB SERVICE

To use the Web Service project WebServiceOracleUpdateDelete, one can develop either a Windows-based or a Web-based Web Service client project. In fact, there is no significant difference between building a client project to consume a Web Service to access the SQL Server database and building a client project to use a Web Service to access the Oracle database. For example, you can use any client project, such as either WinClientSQLUpdateDelete or WebClientSQLUpdateDelete, which we developed in the previous sections, to use this Web Service project with small modifications. The main

modification is to replace the Web Reference with a new Web Reference class that is our new developed Web Service `WebServiceOracleUpdateDelete`.

Follow the modification steps below to complete these changes.

1. Remove the old Web reference from the Windows-based or Web-based client project. You need to first delete the Web reference object and then you can delete the `Web_Reference` folder from the current project.
2. Add a new Web reference using the **Add Web Reference** dialog. Run the desired Web Service project, copy the URL from that running Web Service project, and paste it to the URL box in the **Add Web Reference** dialog in the client project.
3. Change the Web reference name for all data components used in the client project. Change the namespace for all project files.
4. Change the name of the base class located in the Web reference.
5. Change the names of all Web methods located in the Web reference.

Two completed client projects, `WinClientOracleUpdateDelete`, which is Windows based, and `WebClientOracleUpdateDelete`, which is Web based, can be found at the folder `DBProjects\Chapter 9` at the accompanying ftp site (see Chapter 1).

A possible bug that existed in the Windows-based project `WinClientOracleUpdateDelete` is some duplicated setting tags in the configuration file `app.config`. One is the duplicated section tag under the `<sectionGroup>` and the other one is the setting tag under the `<WinClientOracleUpdateDelete.Properties.Settings>`. Remember, only a single section and a setting tag can be allowed in these two tags. Remove the duplicated section and setting tag if you found one.

9.14 CHAPTER SUMMARY

Detailed discussions and analyses about the structure and components of the Web Services are provided in this chapter. Unlike the ASP.NET Web applications in which the user needs to access the Web server through the client browser by sending requests to the server to obtain the desired information, the ASP.NET Web Services provide an automatic way to search, identify, and return the desired information required by the user through a set of methods installed in the Web server, and those methods can be accessed by a computer program, not the user, via the Internet. Another important difference between the ASP.NET Web applications and ASP.NET Web Services is that the latter do not provide any graphic user interfaces (GUIs), and users need to create those GUIs themselves to access the Web Services via the Internet.

Two popular databases, SQL Server and Oracle, are discussed and used for three pairs of example Web Service projects, which include:

- `WebServiceSQLSelect` and `WebServiceOracleSelect`
- `WebServiceSQLInsert` and `WebServiceOracleInsert`
- `WebServiceSQLUpdateDelete` and `WebServiceOracleUpdateDelete`

Each Web Service contains different Web methods that can be used to access different databases and perform the desired data actions such as `Select`, `Insert`, `Update`, and

Delete via the Internet. To use those Web Services, different Web Service client projects are also developed in this chapter. Both Windows-based and Web-based Web Service client projects are discussed and built for each kind of Web Service listed above. In total, 18 projects, including the Web Service projects and the associated Web Service client projects, are developed in this chapter. All projects have been debugged and tested and can be run in any Windows-compatible operating systems such as Windows 95, 98, 2000, XP, Vista, and Window 7.

HOMEWORK

I. True/False Selections

- ___ 1. Web Services can be considered as a set of methods installed in a Web server and can be called by computer programs installed on the clients computers through the Internet.
- ___ 2. Web Services do not require the use of browsers or HTML, and therefore Web Services are sometimes called *application services*.
- ___ 3. XML is a text-based data storage language, and it uses a series of tags to define and store data.
- ___ 4. SOAP is an XML-based communication protocol used for communications between different applications. Therefore, SOAP is a platform-dependent and language-dependent protocol.
- ___ 5. WSDL is an XML-based language for describing Web Services and how to access them. In WSDL terminology, each Web Service is defined as an abstract endpoint or a port and each Web method is defined as an abstract operation.
- ___ 6. UDDI is an XML-based directory for businesses that list themselves on the Internet. The goal of this directory is to enable companies to find one another on the Web and make their systems interoperable for e-commerce.
- ___ 7. The code-behind page is the most important file in a Web Service since all Visual Basic .NET codes related to building a Web Service are located on this page, and our major coding development will be concentrated on this page.
- ___ 8. The names and identifiers used in the SOAP message can be identical. In other words, those names and identifiers can be the same name and identifier used by any other message.
- ___ 9. A single Web Service can contain multiple different Web methods.
- ___ 10. You do not need to deploy a Web Service to the development server if you use that service locally in your computer. However, you must deploy it to a production server if you want other users to access your Web Service via the Internet.

II. Multiple Choices

1. A Web Service is used to effectively _____ the target information required by computer programs.
 - a. Find
 - b. Find, identify, and return
 - c. Identify
 - d. Return

2. Four fundamental components of a Web Service are _____.
 - a. IIS, Internet, Client, and Server
 - b. Endpoint, Port, Operation, and types
 - c. .asmx, web.config, .asmx.vb, and Web_Reference
 - d. XML, SOAP, WSDL, and UDDI
3. The XML is used to _____ the data to be transferred between applications.
 - a. Tag
 - b. Rebuild
 - c. Receive
 - d. Interpreter
4. SOAP is used to _____ the data tagged in the XML format into the messages represented in the SOAP protocol.
 - a. Organize
 - b. Build
 - c. Wrap and pack
 - d. Send
5. WSDL is used to map a concrete network protocol and message format to an abstract endpoint and _____ the Web Services available in an WSDL document format.
 - a. Illustrate
 - b. Describe
 - c. Provide
 - d. Check
6. UDDI is used to _____ all Web Services that are available to users and businesses.
 - a. List
 - b. Display
 - c. Both a and b
 - d. None of above
7. Unlike Web-based applications, a Web Service project does not provide a _____.
 - a. Start page
 - b. Configuration file
 - c. Code-behind page
 - d. Graphic user interface
8. Each Web Service must be located at a unique _____ in order to allow users to access it.
 - a. Computer
 - b. Server
 - c. SOAP file in a server
 - d. Namespace in a server
9. To use a Web Service by either a Windows-based or a Web-based client project, the prerequisite job is to add a _____ into the client project.
 - a. Connection
 - b. Web reference
 - c. Reference
 - d. Proxy class

10. The running result of a Web Service is represented by a(n) _____ format since each Web Service does not provide a graphic user interface (GUI).
- XML
 - HTTP
 - HTML
 - Java scripts

III. Exercises

- Write a paragraph to answer and explain the following questions:
 - What is ASP.NET Web Service?
 - What are main components of the ASP.NET Web Service?
 - How is an ASP.NET Web Service executed?
- Suppose we have a Web Service project and the main service page contains the following statement:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/testWeb.cs"
Class="testWeb" %>
```

Answer the following questions:

- What is the name of this Web Service?
 - What are the name and the location of the code-behind page of this Web Service?
 - Is the content of this page related to the WSDL file of this Web Service?
- Suppose we have developed a Web Service named `WebServiceSQLSelect` with a Web method `GetSQLStudent()` that has an input parameter `student_name` and returns six pieces of student information, such as `student_id`, `gpa`, `credits`, `major`, `schoolYear`, and `email`. Please list steps to develop a Windows-based client project to consume that Web Service.
 - Add the Web method `GetSQLStudent()` in question 3 into our Web Service project `WebServiceSQLSelect` and develop codes to perform the student data query via our sample SQL Server database `CSE_DEPT`. The project file is located at the folder `DBProjects\Chapter 9` at the site ftp://ftp.wiley.com/public/sci_tech_med/practical_database.
 - Develop a Windows-based Web Service client project `WinClientSQLStudent` to consume the Web Service developed in question 4, exactly to consume the new Web method `GetSQLStudent()`.
Hints: Refer to the project `WinClientSQLSelect` located at the folder `DBProjects\Chapter 9` at the site ftp://ftp.wiley.com/public/sci_tech_med/practical_database. Actually you can add some controls to the main form window and some codes to associated methods in that project to complete this job.
 - Develop a Web-based Web Service client project `WebClientSQLStudent` to use the Web Service developed in Exercise 4, that is, to use the new Web method `GetSQLStudent()`.
Hints: Refer to the project `WebClientSQLSelect` located at the folder `DBProjects\Chapter 9` at the site ftp://ftp.wiley.com/public/sci_tech_med/practical_database. Actually you can add some controls to the main page window and some codes to associated methods in that project to complete this job.

Index

.NET Framework 3.5, 117, 153, 170, 174, 202, 459, 654, 745
.NET Framework collection, 14, 474, 477
.NET Framework Data provider for OLE DB, 123, 659
.NET Framework Data provider for SQL Server, 659
.NET-based language, 653–655, 745
.NET-compatible language, 564
.\SQLEXPRESS, 126, 373

A

AcceptButton property, 281–284, 286
AcceptChanges() method, 219
accessing mode Public, 335
Active Server Page.NET, 652
ActiveX Data Object (ADO), 117
Add Column, 94
Add Connection, 214, 215, 223–226, 272–274, 287–289, 327, 328, 396, 643, 711
Add Default Document, 797
Add Existing Item, 375, 570, 790, 820, 855
Add New Data Source, 11, 271, 287, 327
Add New Stored Procedure, 396, 398, 406, 411, 412, 557, 628, 630, 773, 815, 842, 844
Add Project Data Source, 584
Add Query, 295, 304, 310, 318, 319, 480, 504, 512, 586, 596
Add Query method, 304
Add Reference, 222, 425, 547, 612, 643, 711, 720, 761, 780, 790, 821, 855, 867, 868, 883, 901
Add Service Reference, 779
Add Web Reference, 779, 780, 790, 821, 855, 856, 881, 899, 917
Add() method, 131–133, 150, 153, 168, 241, 242, 307, 338, 341, 344, 356, 381, 389, 402, 405, 438, 456–458, 475–479, 489, 490, 501, 527, 530, 542, 550, 552–554, 559, 588, 612–614, 618, 619, 622, 623, 627, 636, 664, 669, 682, 685, 693, 702, 732, 737, 783, 807, 828, 875
AddAfterSelf(), 241, 242
AddBeforeSelf(), 241
AddWithKey, 147

AddWithValue() method, 132, 133
ADO.NET 2.0 architectures, 121, 168, 169
ADO.NET 3.5, 9, 10, 27, 29, 118, 153–159, 161, 166, 168–170, 172, 174, 191, 233, 238, 258
Anonymous types, 234, 237, 253
ANSI 89 standard, 387, 724, 886, 907
ANSI 92 standard, 387, 724, 886, 907
app.config file, 166, 167, 227, 416, 644
App_Code, 751, 754, 756, 760, 762, 799, 800, 834, 866, 867, 882, 900, 920
App_Data, 751, 754
App_WebReferences, 790, 855
application services, 749, 918
Application state, 652, 663, 669, 673, 674, 682, 691, 692, 695–697, 699, 701, 702, 715, 720, 722–724, 726, 730, 731, 736, 739, 746, 791, 822, 825, 827
Application state function, 663, 669, 674, 699, 701, 702, 715, 722, 724, 726, 730, 731, 739, 739, 746, 791, 822, 825, 827
Application.Exit() method, 302, 420
Application.Run() method, 302
array initializer, 246
ArrayList, 10, 179, 181, 193, 194
AS operator, 451
AsEnumerable operator, 179, 258
AsEnumerable() method, 210, 213
ASM file, 751
ASP Code, 393
ASP Request object, 664
ASP Response object, 664
ASP.NET AJAX, 654, 755, 760, 766, 902
ASP.NET application file, 656
ASP.NET parser, 657
ASP.NET Web application, 652, 654–657, 659–660, 663, 665, 675, 738, 744–747, 789, 820
ASP.NET Web Services, 19, 655, 748–751, 753–893, 897–919
AssemblyInfo.cs file, 657
Attributes, 11, 44, 243–244
AutoPostBack property, 652, 673, 680–683, 745, 822, 826
AutoSizeColumnsMode, 291
AutoSizeRowsMode, 291

B

BindingNavigator, 11, 144, 267–269, 330, 463
 BindingSource, 269–270, 296, 298–300, 324, 327, 329–330, 461, 463, 469, 499, 519, 583
 BindingSource object, 263, 269, 299
 Bitmap indexes, 61
 Body page, 451–452, 872–873, 892–893
 BorderStyle property, 286
 B-tree indexes, 61

C

Caching, 119, 656
 Candidate Key, 7, 45
 Cardinality, 44, 61
 Cascade Delete mode, 581, 598
 Cascade Delete Related Records, 71
 Cascade mode, 597, 609, 707, 738, 910, 913
 Cascade Update Related Fields, 71
 Cast() operator, 195
 Cast<TResult> method, 193
 CData value, 240
 child stored procedure, 411–413
 Class libraries, 653
 Clear() method, 145, 147, 370, 493, 534, 606, 889
 ClearBeforeFill property, 313, 505
 Client Server database, 54
 Client-side evaluation, 236
 client-side scripts, 655
 Close() method, 126–128, 140, 210, 213, 306, 309, 340, 343, 420, 522, 561, 664, 669, 795
 code-behind page, 655, 661–662, 751–752, 754–755, 759–760, 765, 767, 774, 776, 800–801, 806, 816, 834, 840, 867–868, 882–883, 900–901, 918–920
 ColumnChanged, 151–152
 Command class, 128–130, 133–135, 171, 367, 402, 462, 517, 519, 525–526, 531, 555, 563, 575, 600, 605, 607, 618–619, 664, 684, 702, 708, 729, 739–740, 803, 817
 Command Console, 654
 CommandText property, 129, 147, 167, 395, 408, 563, 576, 622–623, 650, 812
 CommandType, 129, 134–136, 142, 147, 209–213, 217–219, 339, 345, 354, 367, 379–380, 384, 390, 395, 400, 437, 708, 811, 911
 CommandType property, 129, 147, 395, 408, 462, 576, 622–623, 626–627, 635–636, 650, 774, 817
 common language runtime (CLR), 155, 234
 Community Technology Previews (CTPs), 262
 compiler directive, 760, 800
 Component Object Model (COM), 117, 170
 Composite Primary Key, 93, 99
 Conceptual Design, 40
 Conceptual layer, 154–155, 172
 Conceptual Schema Definition Language (CSDL), 154

configuration file web.config, 672, 751, 753, 865, 867, 881, 882, 899, 900
 Configuration file, 274, 415, 416, 473, 481, 656, 668, 672, 751, 753, 865, 867, 881, 882, 899, 900, 917, 919
 ConfigurationManager class, 765
 Connection Class, 9, 124–128, 221, 336, 414, 540, 548, 720
 Connection property, 129, 147, 403
 connection string, 13, 15, 17, 19, 21, 122, 124–126, 155, 163, 166, 167, 169, 210, 211, 213, 217, 227, 274, 289, 328, 333, 336, 367, 372–375, 377, 389, 415, 416, 423, 433, 434, 436, 516, 538–540, 546, 548, 549, 611, 612, 618, 644, 663, 784, 713, 719–721, 751, 764–766, 768, 865–867, 869, 881, 882, 885, 899, 900, 903
 Connection() Method, 521
 ConnectionState.Open, 127, 336, 342, 378, 381, 382, 419, 426, 440, 521, 539, 540, 549, 663, 665, 670, 673, 682, 691, 695, 721, 723, 725, 729, 747, 765–767, 870, 885, 903
 ConnectionString tag, 415
 ConnectionStrings attribute, 765
 ConnectionStringSettingsCollection object, 765
 Console.WriteLine() method, 188, 194, 195, 200, 208, 216, 218, 220, 228
 Constraint Name, 106, 107, 109, 110, 742
 Constraint Type, 106, 107, 109, 110, 742
 control collection, 142, 474
 control file, 61
 ControlToValidate, 666
 Count Property, 302, 314, 323, 338, 346, 353, 355, 357, 365, 380, 402, 409, 438, 453, 455, 458, 462
 Count() method, 195
 Create Package, 449, 450, 871, 891
 CREATE PROCEDURE, 394, 397, 567, 904, 912
 Create Procedure window, 567, 904, 912
 CURSOR_TYPE, 448, 449, 872, 874, 891
 custom stored procedures, 393, 398

D

Data Access Objects (DOA), 117
 Data Binding Source, 310, 555
 Data Connections node, 223, 225
 Data consistency and data integrity, 39
 Data independence, 39, 113
 Data normalization, 7, 50, 51, 53
 Data Provider, 9, 119–133, 135, 137, 140, 146, 148, 155, 166–172, 331–336, 343, 351, 372–377, 382–386, 389–391, 435, 440, 441, 445, 456, 461–465, 539, 540, 548, 559, 570, 618, 659, 662–664, 668, 672, 673, 681, 683, 691, 720, 722–724, 726, 759, 761, 765, 767, 783, 901
 Data sharing, 39
 Data Sheet view, 94, 97

- Data Source Configuration window, 469, 583
 - Data Source Configuration Wizard, 11, 214, 270–273, 287–289, 327, 328
 - data validation, 14, 15, 30, 474, 475, 478, 482, 493–495, 500, 503, 511, 519–524, 529, 534–536, 545, 555, 556, 559, 561, 564–566, 772, 773, 785, 594, 606, 608, 624, 652, 665–667, 690–692, 697, 698, 731, 734, 736, 810, 813, 833
 - DataAdapter Class, 9, 137–139, 210, 777
 - Database connectivity, 117, 121, 169, 656
 - Database Management System, 39, 272
 - DataBindings property, 298, 299, 310, 321, 500
 - DataColumn objects, 143, 148–150, 462
 - DataColumnCollection, 120, 148, 150, 171
 - DataConnector, 270
 - DataContext, 19, 177, 121, 221–223, 226–228, 259, 260, 372, 414–416, 554, 642–644, 710–713, 715, 717
 - DataGridView, 11, 12, 144, 263, 267–269, 290–295, 460
 - DataPager control, 654
 - DataReader class, 9, 135, 140, 142, 404
 - DataRelation object, 143, 267
 - DataRelations, 276
 - DataRow objects, 10, 143, 146, 148–152, 171, 203, 204, 216
 - DataRowCollection, 120, 150, 171, 462, 580, 698
 - DataRowExtensions, 190, 216, 220
 - DataRows, 469, 581, 582, 592
 - DataRowState, 150
 - DataSet class, 119, 139, 143–146, 267, 275, 365, 793
 - DataSet Designer, 11–12, 143, 214–215, 276–278, 293, 295–296, 304, 310, 318–319, 323, 469, 504, 583
 - DataSet Events, 9, 145
 - DataTable class, 143, 148–151, 171, 209, 220, 786
 - DataTable Events, 9, 151
 - DataTable Methods, 9, 150
 - DataTable Properties, 9, 150
 - DataTableExtensions class, 220
 - DataView, 148
 - DBML file, 222–223
 - DBMS, 38–41, 45, 58
 - DBNull.Value, 217
 - DbType property, 130
 - Debug mode, 878
 - Default SMTP Virtual Server, 797
 - Default.aspx pages, 655
 - deferred loading, 418
 - Delete Rule, 84–86, 88, 231, 707, 746, 840, 845
 - DeleteCommand, 30, 120, 128, 137, 169, 332, 468, 517, 578, 600, 648
 - DeleteOnSubmit() method, 231
 - Design button, 620, 661, 664
 - design tools and wizards, 29–33, 262, 266, 278, 303, 330–331, 466–467, 469, 512, 515–516, 519, 555, 574
 - Design view, 62–63, 65, 76, 661
 - development server, 753, 796, 918
 - DialogResult, 591–593, 607, 623, 628, 636
 - Dictionary() class, 474
 - Dictionary<TKey, TValue>,
 - Direction property, 455
 - Disconnection mode, 276
 - Discovery file, 779
 - DisplayMember property, 321
 - Dispose() method, 128, 138–140, 145, 338, 340
 - Document Object Model (DOM), 192
 - Drop Column, 94
 - DropDownList control, 711, 791, 822
 - dynamic SQL statement, 264, 279, 295, 297
- E**
- Edit DataSet with Designer window, 586
 - ElementAt, 178, 183
 - ElementAtOrDefault<TSource> method, 183
 - Elements() methods, 247
 - EnableVisualStyles method, 269
 - END statement, 450
 - EndEdit() method, 590
 - Enterprise Manager Wizard, 393
 - entity classes, 158, 221–223, 226, 376, 414–415, 417, 426, 554, 642–643, 710–711
 - Entity Data Model (EDM), 154–155, 170, 191
 - Entity Data Model Designer, 156–157, 238, 258
 - Entity Data Model item template, 155
 - Entity Data Model Wizard, 10, 156, 158, 166, 238
 - Entity Framework, 9, 29, 118, 153–154, 159–160, 169–170, 172, 147, 191–192, 233–238, 258
 - Entity Integrity, 7, 45
 - Entity Mapping Details, 156, 158
 - Entity Model Browser, 156–158, 238
 - EntityConnection object, 167, 234
 - EntityContainer, 167, 234
 - EntityObject, 234
 - Entity-Relationship Model (ER), 7, 38
 - EnumerableExtensions class, 251
 - ER notation, 50
 - ErrorMessage, 666
 - ExecuteNonQuery() method, 30, 120, 136–137, 148, 333, 394, 467–468, 495, 517, 526, 563, 574, 578, 600, 605–608, 648, 693, 702, 708–709, 732, 736, 740, 803, 812, 835–836, 841–842
 - ExecuteReader() method, 120, 122, 135–136, 140–141, 168, 171, 333, 339, 346, 355, 357, 381, 390, 394, 402–403, 409, 439, 443, 455, 531, 664, 674, 684, 686, 729, 769, 808, 817, 838–839
 - ExecuteScalar() method, 136, 812
 - extended stored procedures, 393
 - extension method, 220, 247, 252, 253, 260

F

Field() method, 217–219
 FieldCount property, 142
 fields declaration section, 474, 475
 File Processing System, 7, 38, 39
 File Server database, 54
 File System, 198, 199, 201, 753, 754, 756, 780
 Fill() method, 12, 138–140, 146–148, 172, 204, 209–211, 213, 216, 217, 268, 270, 293, 297, 300–302, 314, 319, 320, 338, 339, 346, 353, 355, 357, 359, 365, 380, 389, 390, 394, 395, 402, 409, 438, 443, 455, 458, 461–463, 490, 522, 575, 648, 777, 813
 Fillby() method, 310, 311
 FillSchema() method, 138
 first normal form, 7, 50–52, 113
 First() method, 230, 645, 716
 FirstOrDefault, 183
 foreach loop, 177, 186, 188, 194, 195, 202, 204–206, 208, 210, 213, 216, 228, 236, 255, 259, 319, 324, 346, 347, 355, 356, 359, 368–370, 403, 417, 418, 423, 426, 427, 430, 431, 444, 456, 474, 490, 603, 829
 Foreign Key Relationships, 82–86, 88
 Foreign keys, 7, 32, 38, 40, 45, 57, 68, 69, 82–84, 104, 114, 115, 164, 229, 387, 417, 580, 699, 705, 707, 708, 742, 835, 840, 910
 Form File object, 375, 376, 781
 FrontPage 2002 Server Extensions, 659

G

generic collection, 14, 175, 195, 197, 259, 474, 475
 GET HTTP request, 657
 GetEnumerator method, 180, 182
 GetString() method, 142, 348, 770
 global connection object, 335, 382, 403, 422, 425, 426, 430, 440, 443, 457, 663, 669, 673, 674, 691, 720–724, 726, 791, 822
 Global.asax file, 656, 746
 GroupBy, 206, 245
 GroupJoin method, 180

H

HasRows property, 132, 339, 346, 355, 357, 381, 402, 403, 443, 664, 674, 675, 684, 686, 769, 770, 808, 817
 Hide() method, 302, 314, 380, 522
 HttpApplication base class, 656
 HttpApplicationState class, 663
 HttpContext class, 664
 HttpModules, 656
 Hypertext Transfer Protocol, 654, 749

I

IEnumerator.MoveNext, 205, 255
 IGrouping instances, 237
 IIS virtual directory, 796
 Image.Url property, 675
 ImageAlign property, 671

implicitly typed local variable, 254, 259, 315, 324, 346, 355, 359, 369, 370, 443, 455
 ImportRow(), 150
 Indexes, 8, 58, 60, 61, 199, 200, 449
 Inetpub, 657
 initialization parameter file, 61
 InitializeComponent() method, 755
 inner join, 180, 211, 386, 388
 Insert Row, 94, 101, 641, 642, 916
 InsertCommand, 120, 128, 137, 169, 332, 468, 517, 526, 576, 600, 650, 811, 812, 888
 InsertOnSubmit() method, 229, 554
 InstanceName, 765
 instantiate the proxy class, 782
 Integrated ASP.NET AJAX support, 654
 Integrated Development Environment (IDE), 270
 Internet Information Services, 33, 658, 751, 796–797
 Internet Information Services (IIS), 33, 658, 796
 InvalidOperationException, 337, 541, 549
 IQueryable, 10, 29, 174–178, 181–182, 186, 208–209, 221, 235–237, 253, 256–259, 417, 423, 426–427, 553
 IS operator, 449, 451, 872, 891
 IsClosed property, 140
 IsDigit(), 195
 IsMatch() method, 195
 IsPostBack property, 673

J

Java script function alert(), 860
 Javascript alert() method, 664
 joined table query, 32, 338, 446, 557, 684, 801, 806, 886, 906–907
 Just-In-Time compiler, 653, 745

K

keySelector method, 181
 KeyValuePair, 476–477

L

lambda expressions, 248, 250–251
 Lambda expressions, 250–251, 259
 lambda operator, 250–251
 Language Integrated Query (LINQ), 10, 153, 174–175, 177, 179, 181, 183, 185, 187, 189, 191, 193, 195
 LastOrDefault method, 184
 LINQ Standard Query Operators, 137, 235–236
 LINQ to DataSet, 213–220, 238, 250, 256–260, 323–325, 344, 351–254
 LINQ to Entities, 236–238, 250, 256–258
 LINQ to Objects, 24–28, 175–177, 185–189, 192, 198
 LINQ to SQL, 10, 13, 15, 175–185
 LINQ to XML, 10–11, 22–24, 28–31, 185, 188–192, 238–242
 List(), 228
 List<T>, 184, 200, 202, 220

- ListView control, 654
 - LoadDataRow(), 150
 - Local File System, 753
 - Local IIS, 798
 - localhost, 126, 373, 377–378, 658, 663, 764–765, 780, 799
 - Logical Design, 40
 - Logical layer, 154, 172
- M**
- machine.config file, 656
 - main service page file Service.asmx, 753, 756
 - many-to-many relationship, 47, 49, 110
 - Mapping Schema Language (MSL), 154
 - Materialization, 237
 - MetadataWorkspace object, 234
 - method-based query syntax, 204, 235
 - Microsoft ACE OLEDB 12.0, 127
 - Microsoft Intermediate Language (MSIL), 653, 657, 746
 - Microsoft SQL Server 2005 Express, 33, 82, 162, 287
 - Microsoft SQL Server Database Engine,
 - Microsoft.ACE.OLEDB.12.0 driver, 127
 - Minimizing data redundancy, 113
 - MissingSchemaAction property, 147
 - Modify Column, 94
 - MSDAORA driver, 373
- N**
- Name page, 449, 450, 871, 891, 892
 - Named Parameter Mapping, 131, 132, 171
 - NamespaceResolver, 248
 - nested stored procedure, 411, 413, 414, 461
 - New Foreign Key, 82, 84–86, 88
 - NewRow() method, 148
 - null object, 731, 768, 769
 - nullable data type, 217
- O**
- Object Browser, 91, 93, 106, 446, 449, 453, 461, 462, 464, 567, 570, 576, 637, 638, 640, 642, 650, 740, 871, 891, 904, 912
 - Object Browser page, 446, 449, 462, 464, 467, 570, 576, 637, 642, 650, 871, 891, 904, 912
 - object initializers, 254
 - Object Linking and Embedding (OLE), 121
 - Object Linking and Embedding Database, 121
 - Object Relational designer, 191, 221–224, 226, 258, 260, 414, 415, 428, 429, 554, 643, 711
 - Object Relational Mapping, 421
 - Object Services, 11, 233, 234, 237
 - ObjectContext class, 234
 - ObjectQuery, 11, 234–237
 - ObjectStateManager, 234
 - OdbcType, 130
 - OfType, 175, 177, 178, 180, 181, 185, 193
 - OfType<TResult>, 181, 193
 - OleDbConnection, 122, 125–127, 137, 167, 209, 212, 218, 219, 335, 336, 373, 374, 460, 461, 540, 542, 552
 - OleDbDataAdapter, 123, 137, 209, 212, 218, 219, 337, 345, 354, 358, 364, 460, 461, 541, 543
 - OleDbDataReader, 122, 125, 140, 339, 345, 348, 354, 356, 358, 360, 385, 389, 391, 444, 460, 461, 543, 544, 552, 619
 - OleDbExceptionError, 337, 541
 - OleDbType, 130, 131, 337, 339, 345, 354, 358, 364, 385, 389, 541–544, 576, 618, 619, 622, 623, 650
 - ON clause, 389, 553, 684
 - On Delete Cascade, 19, 106, 107, 109, 110, 742, 743, 910, 913
 - one-to-many relationships, 49
 - Open DataBase Connectivity, 117, 121, 169
 - Open() Method, 9, 126–128, 167, 210, 211, 217, 333, 522, 664, 765
 - Oracle client reference, 547
 - Oracle Database 10g Express Edition, 13, 89, 112, 432, 446, 449, 548, 567, 637, 740, 871, 891, 904, 906, 912, 913
 - Oracle Database 10g Release, 327, 433
 - Oracle package, 13, 32, 453, 455, 464, 465, 575, 649, 874, 875, 878, 889, 891, 894, 908
 - Oracle Parameter objects, 437, 455, 875
 - Oracle stored procedures, 461
 - OracleClient namespace, 128, 548, 549
 - OracleDataAdapter, 137, 437, 442, 454, 458, 461, 550, 551, 877, 888
 - OracleDataReader, 135, 140, 439, 442, 444, 454, 456, 457, 551, 552, 614, 615, 722–726, 729, 869, 870, 876, 886, 887, 890, 891, 907–910
 - OracleExceptionError, 549
 - OracleParameter object, 443
 - OracleParameter objects, 438, 891, 909, 910
 - OrderBy, 178, 181, 198, 205–208, 236, 245, 248, 255, 259
 - Output dialog box, 399
 - Output window, 473, 481, 558, 774, 799, 815, 845
- P**
- Package Name, 891
 - Page_Load() method, 663, 668, 672–673, 682–683, 691, 695, 713, 720, 722–724, 727–729, 747, 789, 791–792, 820, 822–823, 855–857
 - Parameter class, 129–130, 169
 - ParameterCollection class, 132
 - ParameterDirection property, 875
 - ParameterName property, 438
 - Parameters property, 129–130, 133, 171, 605, 613, 618–619, 622–623, 627, 636
 - params object, 240
 - parent stored procedure, 411

- Parse() method, 506, 510
 - passing-by-reference, 563, 770
 - password file, 59, 61–62
 - PasswordChar property, 300
 - PerformClick() method, 506
 - Physical Design, 40–41
 - PL-SQL, 449–451, 464, 568, 577, 636, 650, 872–873, 892, 904–905, 912
 - Position property, 679, 712
 - Positional Parameter Mapping, 131–132
 - Preview Data, 275, 277–278, 289
 - Primary keys, 388, 45, 77, 114
 - Procedural Language Extension, 449, 464, 568, 872, 904, 912
 - Procedure Name, 567, 904, 912
 - production server, 796, 798, 918
 - Project Assembly files (.dll),
 - Project Main Entry file, 537–538, 547–548
 - Property Page, 727
- Q**
- Queries, 8, 15–16, 19–20, 55, 185, 188, 235, 276, 553–554, 585, 716, 768
 - Query Analyzer, 393
 - Query Execution, 186–188
 - query expression syntax, 182–183, 204–205, 235
- R**
- Read() method, 142, 348, 360, 677, 686, 770, 818
 - ReadAllLines(), 197
 - Real Time Coding Method, 393
 - Redirect() method, 669–670, 677, 683, 691, 795
 - redo log files, 59, 61
 - Referential Integrity, 7, 45, 71
 - Remote Data Objects (RDO),
 - Remote Procedure Call (RPC), 749
 - Remove() method, 243–244, 478
 - RemoveNodes() method, 243
 - Rename Column, 94
 - ReplaceNodes() method, 242
 - RequiredFieldValidator control, 666
 - Response object, 664, 669–670, 675, 677, 683, 691, 793, 823
 - RowChanged, 151–152
 - RowDeleted, 151–152
 - Run Stored Procedure, 413, 558, 628–631, 774, 815, 843, 845–846
- S**
- ScriptService, 755, 760, 766, 902
 - second normal form, 50–51, 53
 - Select Page to Start dialog box, 727
 - SELECT which returns a single value, 304, 318
 - SelectedIndexChanged() method, 356, 357, 389, 426, 457
 - SelectMany() method, 128
 - SelectSingleNode, 248
 - Service Reference Settings, 779
 - Set As Start Page, 670, 795, 831
 - SET command, 407, 413, 453, 873, 893
 - SetAttribute(), 245
 - SetElement(), 243, 244
 - SetField, 213, 217–220
 - Show Methods Pane, 428
 - Show Table Data, 559, 630, 844
 - Simple Object Access Protocol, 648, 748, 749
 - SOAP message, 749, 751, 752, 757, 918
 - SOAP Namespaces, 748
 - Source button, 661
 - specification page, 579, 571, 872, 873, 891, 892
 - Split button, 661
 - Split() method, 195, 198
 - SQL Commands, 33, 446, 449, 637, 638
 - SQL Server 2005 Express, 28, 33, 82, 126, 162, 272, 274, 279, 287, 288, 371–374, 410, 460, 461, 496, 675
 - SQL Server Authentication mode, 288
 - SQL Server Enterprise Manager, 393
 - SQL Server Management Studio Express, 72, 115, 232, 233, 274, 420, 422, 446, 536, 627, 697, 707, 845
 - SqlDbType.Int, 802, 811, 812, 836, 885, 889, 903
 - SqlDbType.Text, 769, 775, 776, 802, 807, 811, 816, 836, 837, 839, 841, 868, 877, 885, 887, 889, 903, 907, 911
 - SQLMetal, 191, 221, 222, 258, 414, 415, 554
 - Stack(), 474
 - standard Integrated Security, 378
 - Standard Query Operators, 10–11, 29, 174, 177–179, 220, 234–236
 - Start Options, 694, 702, 727, 733, 737
 - StartPosition property, 158, 281–284, 286
 - Store Schema Definition Language, 154
 - string array, 183, 500–502, 521–523, 559, 805, 809, 818
 - SubmitChanges(), 229–232, 544, 575, 646, 647, 715, 717
 - System stored procedures, 393
 - System.Collections namespace, 193
 - System.Collections.Generic, 167, 175, 189, 194, 198, 201, 203, 207, 212, 216, 227, 335, 352, 416, 419, 421, 425, 430, 435, 441, 445, 474, 475, 525
 - System.Collections.Generic namespace, 175
 - System.Data namespace, 118, 119, 148, 170, 332
 - System.Data.Linq, 189, 222, 226, 227, 378, 414–416, 418, 420, 421, 424, 425, 429, 643, 711
 - System.Data.Objects.DataClasses, 234
 - System.Data.Odbc, 122, 332
 - System.Data.OleDb, 122–123, 209–212, 215–219, 332, 334–335, 343, 351–352, 362, 371, 383, 386, 540, 542, 618

System.Data.OracleClient, 122, 124, 332, 371, 435, 441, 445, 547, 549–550, 552, 570, 612, 720–726, 866, 868, 883, 901–902

System.Data.SqlClient, 332, 383, 441

System.Drawing() method, 676

System.Drawing.Image.FromFile(), 349

System.Expressions namespace, 236

System.IO, 195–198, 200–201

System.Linq, 167, 175, 186, 189, 194, 198, 201, 203, 207, 209, 212, 216, 218–220, 227, 335, 352, 378, 383, 386, 392, 416, 419, 421, 425, 430, 435, 441, 445, 525, 553, 755, 760, 766

System.Reflection, 202–203

System.Web.UI.WebControls.Image, 675

System.XML.Linq, 189

System.Xml.XPath namespace, 248

System.Xml.Xsl namespace, 248

T

TabIndex, 281–285, 327, 471–472, 498–499, 662, 668, 671, 679, 689, 712, 728

TableAdapter Configuration Wizard, 293, 295, 298, 310, 468

TableAdapter.Delete(), 30, 467, 578, 591–592, 648–649

TableAdapter.Insert(), 30, 467–468, 470, 472, 478, 480, 482, 495, 498, 574–575

TableAdapter.Insert() method, 30, 467–468, 470, 482, 498, 574–575

TableAdapter's Update() method, 467–468, 508, 574

TableAdapter's DBDirect methods, 467

Test Connection, 163, 214, 223, 274, 288–289, 327, 396

TextChanged event, 483–484, 503, 508, 510, 527, 733, 820, 822, 826, 855–856

TextChanged method, 483–484, 510, 527, 727, 731–733

third normal form, 50–53

tnsnames.ora file, 373–374, 433–434

ToArray, 178, 184–185, 188, 204

ToList, 178, 184–185, 188, 204

Toolbox window, 262, 266–268, 270, 281, 290, 661, 666, 668, 670–671

ToString() method, 347, 490, 786

Transform() method, 248

typed DataSet, 144, 213–216, 276, 786

U

unified application models, 654

unified type system, 654

Universal Description, Discovery and Update Rule, 84–86, 88, 707

UPDATE Specification, 597, 609, 707

UpdateCommand, 30, 120, 128, 137, 169, 332, 468, 517, 578, 600, 648

Use SQL Statements, 304

using keyword, 335

V

Validate() command, 590

validation controls, 666

Value boxes, 894, 914

VALUES, 136, 447, 525–526, 553, 558, 569, 575, 577, 692, 731–732, 811, 888

VARCHAR, 394, 455, 557, 651

Views, 8, 57, 59, 163, 192

virtual directory, 796–799

W

W3C XML Information Set, 238

Web configuration file, 668, 865, 867, 881–882, 899–900

Web Form, 19–21, 654–655, 657, 661, 667, 670, 678, 689, 711, 745, 790, 821, 855

Web reference, 778–785, 790

Web server, 30, 655–658, 673, 744, 745, 748, 749, 753, 779, 792, 795–798, 819, 922, 925, 917, 918

Web service deployment, 796

Web service proxy class, 20, 772, 778–780, 819

Web Services Description Language, 748, 749

Web Services Interoperability Organization, 755

Web.config configuration file, 656

Web_Reference folder, 881, 899, 917

Web-based application, 652, 663, 675, 676, 702, 746

Web-based client application, 789, 820

WebServiceBinding attributes, 751, 754

Where(), 208, 229, 251

Window Form Designer files, 538, 547

Window Form Object files, 537, 538, 547

window.close, 665, 670, 714, 721, 795, 831

Windows Authentication, 72, 163, 288, 394, 754, 765

Windows Communication Foundation, 262

Windows Components Wizard, 658, 659

Windows Form Designing window, 281

Windows Forms Application, 279, 326

Windows Presentation Foundation, 262

Write() method, 264, 269, 275, 793, 823

X–Z

XAttributes, 238

XCDATA, 238, 239

XComment, 239, 240, 247

XContainer, 239

XDeclaration, 239

XDocuments, 238

XDocumentType, 239

XML names, 239

XML namespace, 239, 248

XML Schema, 143, 146, 149, 151, 167, 168

XML Transformation, 241, 245

902 Index

XML Tree, 238, 239, 241, 243, 246
XML Web service links, 656
XML Web Services, 653–655, 755, 779
xmlns attribute, 758
XML-specific query extensions, 246, 247
XNamespace, 239
XNodes, 238
XObject, 238, 239, 247

XPathEvaluate, 248
XPathSelectNodes, 248
XPathNavigator, 248
XPathSelectElement, 248
XProcessingInstruction, 238–240
XQuery, 186, 205, 238
XSLT, 11, 238, 248
XText, 238–240

About the Author

Dr. YING BAI is an Associate Professor in the Department of Computer Science and Engineering at Johnson C. Smith University, Charlotte, North Carolina. His special interests include computer architecture, software engineering, mix-language programming, fuzzy logic controls, robotic controls, and robots calibrations. His industry experience includes positions as software and senior software engineers at companies such as Motorola MMS, Schlumberger ATE Technology, Immix TeleCom, and Lam Research. During recent years, Dr. Bai has published more than 20 academic research papers in IEEE transactions journals and international conferences. Since 2003, he has also published 6 books; his first book was also translated to Russian in 2005. Most books are about software programming, serial port programming, fuzzy logic controls in industrial applications, and database programming.

www.FreeDownload.ir