

Practical Database Programming with Visual Basic.NET

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

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

R. Abari
J. Anderson
F. Canavero
T. G. Croda

M. El-Hawary
B. M. Hammerli
M. Lanzerotti
O. Malik

S. Nahavandi
W. Reeve
T. Samad
G. Zobrist

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

Practical Database Programming with Visual Basic.NET

Second Edition

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 © 2012 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/permissions>.

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 Basic.NET / Ying Bai. – 2nd ed.

p. cm.

ISBN 978-1-118-16205-7 (pbk.)

1. Microsoft Visual BASIC.
2. BASIC (Computer program language)
3. Microsoft .NET.
4. Database design. I. Title.
QA76.73.B3B335 2012
005.2'768–dc23

2011039947

Printed in United States of America.

*This book is dedicated to my wife, Yan Wang,
and my daughter, Xue Bai.*

Contents

Preface xxv

Acknowledgments xxvii

Chapter 1 Introduction 1

Outstanding Features about This Book 2
Who This Book Is For 2
What This Book Covers 2
How This Book Is Organized and How to Use This Book 5
How to Use the Source Code and Sample Databases 6
Instructors and Customers Supports 8

Chapter 2 Introduction to Databases 10

2.1 What Are Databases and Database Programs? 11
 2.1.1 File Processing System 11
 2.1.2 Integrated Databases 12
2.2 Develop a Database 13
2.3 Sample Database 14
 2.3.1 Relational Data Model 14
 2.3.2 Entity-Relationship Model 17
2.4 Identifying Keys 17
 2.4.1 Primary Key and Entity Integrity 17
 2.4.2 Candidate Key 18
 2.4.3 Foreign Keys and Referential Integrity 18
2.5 Define Relationships 19
 2.5.1 Connectivity 19
2.6 ER Notation 21
2.7 Data Normalization 22
 2.7.1 First Normal Form (1NF) 22
 2.7.2 Second Normal Form (2NF) 23
 2.7.3 Third Normal Form (3NF) 24

2.8	Database Components in Some Popular Databases	26
2.8.1	Microsoft Access Databases	26
2.8.1.1	Database File	27
2.8.1.2	Tables	27
2.8.1.3	Queries	27
2.8.2	SQL Server Databases	28
2.8.2.1	Data Files	28
2.8.2.2	Tables	29
2.8.2.3	Views	29
2.8.2.4	Stored Procedures	29
2.8.2.5	Keys and Relationships	30
2.8.2.6	Indexes	30
2.8.2.7	Transaction Log Files	30
2.8.3	Oracle Databases	31
2.8.3.1	Data Files	31
2.8.3.2	Tables	31
2.8.3.3	Views	32
2.8.3.4	Stored Procedures	32
2.8.3.5	Indexes	33
2.8.3.6	Initialization Parameter Files	33
2.8.3.7	Control Files	33
2.8.3.8	Redo Log Files	34
2.8.3.9	Password Files	34
2.9	Create Microsoft Access Sample Database	34
2.9.1	Create the LogIn Table	34
2.9.2	Create the Faculty Table	36
2.9.3	Create the Other Tables	38
2.9.4	Create Relationships among Tables	41
2.10	Create Microsoft SQL Server 2008 Sample Database	44
2.10.1	Create the LogIn Table	46
2.10.2	Create the Faculty Table	48
2.10.3	Create Other Tables	49
2.10.4	Create Relationships among Tables	54
2.10.4.1	Create Relationship between the LogIn and the Faculty Tables	54
2.10.4.2	Create Relationship between the LogIn and the Student Tables	57
2.10.4.3	Create Relationship between the Faculty and the Course Tables	58
2.10.4.4	Create Relationship between the Student and the StudentCourse Tables	59
2.10.4.5	Create Relationship between the Course and the StudentCourse Tables	60
2.11	Create Oracle 11g XE Sample Database	61
2.11.1	Create a New Oracle Customer User or User Account	63
2.11.2	Create New Customer Sample Database CSE_DEPT	65
2.11.3	Create the LogIn Data Table	69
2.11.4	Create the Faculty Data Table	71
2.11.5	Create Other Tables	74
2.11.6	Create the Constraints between Tables	78

2.11.6.1	Create the Constraints between the LogIn and Faculty Tables	78
2.11.6.2	Create the Constraints between the LogIn and Student Tables	81
2.11.6.3	Create the Constraints between the Course and Faculty Tables	83
2.11.6.4	Create the Constraints between the StudentCourse and Student Tables	83
2.11.6.5	Create the Constraints between the StudentCourse and Course Tables	85
2.12	Chapter Summary	87
	Homework	88

Chapter 3 Introduction to ADO.NET

91

3.1	The ADO and ADO.NET	91
3.2	Overview of ADO.NET	93
3.3	The Architecture of ADO.NET	94
3.4	The Components of ADO.NET	95
3.4.1	The Data Provider	95
3.4.1.1	The ODBC Data Provider	97
3.4.1.2	The OLEDB Data Provider	97
3.4.1.3	The SQL Server Data Provider	98
3.4.1.4	The Oracle Data Provider	98
3.4.2	The Connection Class	99
3.4.2.1	The Open() Method of the Connection Class	101
3.4.2.2	The Close() Method of the Connection Class	102
3.4.2.3	The Dispose() Method of the Connection Class	102
3.4.3	The Command and the Parameter Classes	103
3.4.3.1	The Properties of the Command Class	103
3.4.3.2	The Constructors and Properties of the Parameter Class	104
3.4.3.3	Parameter Mapping	105
3.4.3.4	The Methods of the ParameterCollection Class	107
3.4.3.5	The Constructor of the Command Class	108
3.4.3.6	The Methods of the Command Class	109
3.4.4	The DataAdapter Class	112
3.4.4.1	The Constructor of the DataAdapter Class	112
3.4.4.2	The Properties of the DataAdapter Class	112
3.4.4.3	The Methods of the DataAdapter Class	113
3.4.4.4	The Events of the DataAdapter Class	113
3.4.5	The DataReader Class	115
3.4.6	The DataSet Component	117
3.4.6.1	The DataSet Constructor	119
3.4.6.2	The DataSet Properties	120
3.4.6.3	The DataSet Methods	120
3.4.6.4	The DataSet Events	121
3.4.7	The DataTable Component	123
3.4.7.1	The DataTable Constructor	124
3.4.7.2	The DataTable Properties	125

3.4.7.3	The DataTable Methods	126
3.4.7.4	The DataTable Events	126
3.4.8	ADO.NET Entity Framework 4.1	128
3.4.8.1	Advantages of Using the Entity Framework 4.1	130
3.4.8.2	The ADO.NET 4.1 Entity Data Model	132
3.4.8.3	Using the ADO.NET 4.1 Entity Data Model Wizard	136
3.5	Chapter Summary	145
	Homework	146

Chapter 4 Introduction to Language Integrated Query (LINQ)

149

4.1	Overview of Language Integrated Query	149
4.1.1	Some Special Interfaces Used in LINQ	150
4.1.1.1	The IEnumerable and IEnumerable(Of T) Interfaces	150
4.1.1.2	The IQueryable and IQueryable(Of T) Interfaces	151
4.1.2	Standard Query Operators	152
4.1.3	Deferred Standard Query Operators	154
4.1.3.1	AsEnumerable (Conversion Purpose)	154
4.1.3.2	Cast (Conversion Purpose)	154
4.1.3.3	Join (Join Purpose)	154
4.1.3.4	OfType (Conversion Purpose)	156
4.1.3.5	OrderBy (Ordering Purpose)	156
4.1.3.6	Select (Projection Purpose)	157
4.1.3.7	Where (Restriction Purpose)	158
4.1.4	Nondeferred Standard Query Operators	158
4.1.4.1	ElementAt (Element Purpose)	158
4.1.4.2	First (Element Purpose)	159
4.1.4.3	Last (Element Purpose)	159
4.1.4.4	Single (Element Purpose)	159
4.1.4.5	ToArray (Conversion Purpose)	160
4.1.4.6	ToList (Conversion Purpose)	160
4.2	Introduction to LINQ Query	161
4.3	The Architecture and Components of LINQ	164
4.3.1	Overview of LINQ to Objects	165
4.3.2	Overview of LINQ to DataSet	165
4.3.3	Overview of LINQ to SQL	166
4.3.4	Overview of LINQ to Entities	167
4.3.5	Overview of LINQ to XML	168
4.4	LINQ to Objects	168
4.4.1	LINQ and ArrayList	169
4.4.2	LINQ and Strings	170
4.4.2.1	Query a String to Determine the Number of Numeric Digits	171
4.4.2.2	Sort Lines of Structured Text by any Field in the Line	172
4.4.3	LINQ and File Directories	175
4.4.3.1	Query the Contents of Files in a Folder	175
4.4.4	LINQ and Reflection	177

4.5	LINQ to DataSet	179	
4.5.1	Operations to DataSet Objects	179	
4.5.1.1	Query Expression Syntax	180	
4.5.1.2	Method-Based Query Syntax	182	
4.5.1.3	Query the Single Table	184	
4.5.1.4	Query the Cross Tables	186	
4.5.1.5	Query Typed DataSet	189	
4.5.2	Operations to DataRow Objects Using the Extension Methods	192	
4.5.3	Operations to DataTable Objects	196	
4.6	LINQ to SQL	197	
4.6.1	LINQ to SQL Entity Classes and DataContext Class	198	
4.6.2	LINQ to SQL Database Operations	202	
4.6.2.1	Data Selection Query	203	
4.6.2.2	Data Insertion Query	205	
4.6.2.3	Data Updating Query	206	
4.6.2.4	Data Deletion Query	207	
4.6.3	LINQ to SQL Implementations	210	
4.7	LINQ to Entities	210	
4.7.1	The Object Services Component	211	
4.7.2	TheObjectContext Component	211	
4.7.3	TheObjectQuery Component	211	
4.7.4	LINQ to Entities Flow of Execution	211	
4.7.4.1	Construct an ObjectQuery Instance	212	
4.7.4.2	Compose a LINQ to Entities Query	212	
4.7.4.3	Convert the Query to Command Trees	212	
4.7.4.4	Execute the Query	213	
4.7.4.5	Materialize the Query	214	
4.7.5	Implementation of LINQ to Entities	214	
4.8	LINQ to XML	215	
4.8.1	LINQ to XML Class Hierarchy	215	
4.8.2	Manipulate XML Elements	216	
4.8.2.1	Creating XML from Scratch	216	
4.8.2.2	Insert XML	218	
4.8.2.3	Update XML	219	
4.8.2.4	Delete XML	220	
4.8.3	Manipulate XML Attributes	220	
4.8.3.1	Add XML Attributes	220	
4.8.3.2	Get XML Attributes	221	
4.8.3.3	Delete XML Attributes	222	
4.8.4	Query XML with LINQ to XML	222	
4.8.4.1	Standard Query Operators and XML	223	
4.8.4.2	XML Query Extensions	224	
4.8.4.3	Using Query Expressions with XML	224	
4.8.4.4	Using XPath and XSLT with LINQ to XML	225	
4.8.4.5	Mixing XML and Other Data Models	225	

4.9	Visual Basic.NET Language Enhancement for LINQ	227
4.9.1	Lambda Expressions	227
4.9.2	Extension Methods	229
4.9.3	Implicitly Typed Local Variables	232
4.9.4	Query Expressions	234
4.10	Chapter Summary	236
	Homework	237

Chapter 5 Data Selection Query with Visual Basic.NET

241

Part I Data Query with Visual Studio.NET Design Tools and Wizards 242

5.1	A Completed Sample Database Application Example	242
5.2	Visual Studio.NET Design Tools and Wizards	245
5.2.1	Data Components in the Toolbox Window	245
5.2.1.1	The DataSet	246
5.2.1.2	DataGridView	247
5.2.1.3	BindingSource	248
5.2.1.4	BindingNavigator	248
5.2.1.5	TableAdapter	249
5.2.2	Data Source Window	249
5.2.2.1	Add New Data Sources	250
5.2.2.2	Data Source Configuration Wizard	251
5.2.2.3	DataSet Designer	255
5.3	Query Data from SQL Server Database Using Design Tools and Wizards	257
5.3.1	Application User Interface	257
5.3.1.1	The LogIn Form	258
5.3.1.2	The Selection Form	259
5.3.1.3	The Faculty Form	260
5.3.1.4	The Course Form	260
5.3.1.5	The Student Form	263
5.4	Add and Utilize Visual Studio Wizards and Design Tools	265
5.4.1	Add and Configure a New Data Source	265
5.5	Query and Display Data using the DataGridView Control	268
5.5.1	View the Entire Table	268
5.5.2	View Each Record or the Specified Columns	270
5.6	Use DataSet Designer to Edit the Structure of the DataSet	272
5.7	Bind Data to the Associated Controls in LogIn Form	274
5.8	Develop Codes to Query Data Using the Fill() Method	278
5.9	Use Return a Single Value to Query Data for LogIn Form	281
5.10	Develop the Codes for the Selection Form	284
5.11	Query Data from the Faculty Table for the Faculty Form	286
5.12	Develop Codes to Query Data from the Faculty Table	289
5.12.1	Develop Codes to Query Data Using the TableAdapter Method	289
5.12.2	Develop Codes to Query Data Using the LINQ to DataSet Method	291
5.13	Display a Picture for the Selected Faculty	292
5.13.1	Modify the Codes for the Select Button Event Procedure	292
5.13.2	Create a Function to Select the Matched Faculty Image	293

5.14	Query Data from the Course Table for the Course Form	295
5.14.1	Build the Course Queries Using the Query Builder	296
5.14.2	Bind Data Columns to the Associated Controls in the Course Form	298
5.15	Develop Codes to Query Data for the Course Form	300
5.15.1	Query Data from the Course Table Using the TableAdapter Method	300
5.15.2	Query Data from the Course Table Using the LINQ to DataSet Method	302
5.16	Query Data from Oracle Database Using Design Tools and Wizards	304
5.16.1	Introduction to dotConnect for Oracle 6.30 Express	305
5.16.2	Create a New Visual Basic.NET Project: SelectWizardOracle	305
5.16.3	Select and Add Oracle Database 11g XE as the Data Source	307
5.16.4	Modify the Codes to Access the Oracle Database	310

Part II Data Query with Runtime Objects 311

5.17	Introduction to Runtime Objects	312
5.17.1	Procedure of Building a Data-Driven Application Using Runtime Object	314
5.18	Query Data from Microsoft Access Database Using Runtime Object	315
5.18.1	Query Data Using Runtime Objects for the LogIn Form	315
5.18.1.1	Declare Global Variables and Runtime Objects	316
5.18.1.2	Connect to the Data Source with the Runtime Object	317
5.18.1.3	Coding for Method 1: Using DataSet-TableAdapter to Query Data	318
5.18.1.4	Coding for Method 2: Using the DataReader to Query Data	320
5.18.1.5	Clean up the Objects and Terminate the Project	321
5.18.2	Coding for the Selection Form	322
5.18.3	Query Data Using Runtime Objects for the Faculty Form	323
5.18.4	Query Data Using Runtime Objects for the Course Form	331
5.18.5	Query Data Using Runtime Objects for the Student Form	339
5.18.5.1	Coding for the Student Form_Load Event Procedure	341
5.18.5.2	Coding for the Select Button Click Event Procedure	342
5.19	Query Data from SQL Server Database Using Runtime Object	349
5.19.1	Migrating from Access to SQL Server and Oracle Databases	350
5.19.2	Query Data Using Runtime Objects for the LogIn Form	353
5.19.2.1	Declare the Runtime Objects	354
5.19.2.2	Connect to the Data Source with the Runtime Object	354
5.19.2.3	Coding for Method 1: Using the TableAdapter to Query Data	356
5.19.2.4	Coding for Method 2: Using the DataReader to Query Data	357
5.19.3	The Coding for the Selection Form	359
5.19.4	Query Data Using Runtime Objects For the Faculty Form	359
5.19.5	Query Data Using Runtime Objects for the Course Form	362
5.19.6	Retrieve Data from Multiple Tables Using Tables JOINS	363
5.19.7	Query Data Using Runtime Objects for the Student Form	367

5.19.8	Query Student Data Using Stored Procedures	369
5.19.8.1	Create the Stored Procedure	370
5.19.8.2	Call the Stored Procedure	371
5.19.8.3	Query Data Using Stored Procedures for Student Form	372
5.19.8.4	Query Data Using More Complicated Stored Procedures	380
5.20	Query Data from Oracle Database Using Runtime Object	384
5.20.1	Install and Configure the Oracle Database 11g Express Edition	384
5.20.2	Configure the Oracle Database Connection String	385
5.20.3	Query Data Using Runtime Objects for the LogIn Form	386
5.20.3.1	Declare the Runtime Objects and Modify the ConnModule	387
5.20.3.2	Connect to the Data Source with the Runtime Object	388
5.20.3.3	Coding for Method 1: Using the TableAdapter to Query Data	389
5.20.3.4	Coding for Method 2: Using the DataReader to Query Data	390
5.20.4	The Coding for the Selection Form	392
5.20.5	Query Data Using Runtime Objects for the Faculty Form	392
5.20.6	Query Data Using Runtime Objects and LINQ to DataSet for the Course Form	396
5.20.7	The Stored Procedures in Oracle Database Environment	397
5.20.7.1	The Syntax of Creating a Stored Procedure in the Oracle	398
5.20.7.2	The Syntax of Creating a Package in the Oracle	398
5.20.8	Create the Faculty_Course Package for the Course Form	400
5.20.9	Query Data Using the Oracle Package For the Course Form	405
5.21	Chapter Summary	411
	Homework	413

Chapter 6 Data Inserting with Visual Basic.NET

417

Part I Data Inserting with Visual Studio.NET Design Tools and Wizards 418

6.1	Insert Data into a Database	418
6.1.1	Insert New Records into a Database Using the TableAdapter.Insert Method	419
6.1.2	Insert New Records into a Database Using the TableAdapter.Update Method	420
6.2	Insert Data into the SQL Server Database Using a Sample Project InsertWizard	420
6.2.1	Create New Project Based on the SelectWizard Project	421
6.2.2	Application User Interfaces	421
6.2.3	Validate Data Before the Data Insertion	421
6.2.3.1	Visual Basic Collection and .NET Framework Collection Classes	422
6.2.3.2	Validate Data Using the Generic Collection	422
6.2.4	Initialization Coding for the Data Insertion	425
6.2.5	Build the Insert Query	426
6.2.5.1	Configure the TableAdapter and Build the Data Inserting Query	426
6.2.6	Develop Codes to Insert Data Using the TableAdapter.Insert Method	427

6.2.7	Develop Codes to Insert Data Using the TableAdapter.Update Method	430
6.2.8	Insert Data into the Database Using the Stored Procedures	435
6.2.8.1	Create the Stored Procedure Using the TableAdapter Query Configuration Wizard	436
6.2.8.2	Modify the Codes to Perform the Data Insertion Using the Stored Procedure	436
6.3	Insert Data into the Oracle Database Using a Sample Project InsertWizardOracle	441

Part II Data Insertion with Runtime Objects 442

6.4	The General Runtime Objects Method	442
6.5	Insert Data into the SQL Server Database Using the Runtime Object Method	444
6.5.1	Insert Data into the Faculty Table for the SQL Server Database	444
6.5.1.1	Develop the Codes to Insert Data into the Faculty Table	444
6.6	Insert Data into the Microsoft Access Database Using the Runtime Objects	453
6.6.1	Modify the Imports Commands and the ConnModule	454
6.6.2	Modify the Database Connection String	454
6.6.3	Modify the LogIn Query Strings	455
6.6.4	Modify the Faculty Query String	456
6.6.5	Modify the Faculty Insert String	458
6.6.6	Modifications to Other Forms	459
6.7	Insert Data into the Oracle Database Using the Runtime Objects	461
6.7.1	Add the Oracle Driver Reference and Modify the Imports Commands	462
6.7.2	Modify the Database Connection String	463
6.7.3	Modify the LogIn Query Strings	464
6.7.4	Modify the Faculty Query String and Query Related Codes	466
6.7.5	Modify the Faculty Insert String and Insertion Related Codes	466
6.7.6	Modifications to Other Forms	468
6.7.6.1	Modify the Codes in the Selection Form	469
6.7.6.2	Modify the Codes in the Course Form	469
6.7.6.3	Modify the Codes in the Student Form	470
6.7.6.4	Modify the Codes in the SP Form	470
6.8	Insert Data into the Database Using Stored Procedures	471
6.8.1	Insert Data into the SQL Server Database Using Stored Procedures	471
6.8.1.1	Develop Stored Procedures of SQL Server Database	471
6.8.1.2	Develop Codes to Call Stored Procedures to Insert Data into the Course Table	474
6.8.2	Insert Data into the Oracle Database Using Stored Procedures	478
6.8.2.1	Develop Stored Procedures in Oracle Database	479
6.8.2.2	Develop Codes to Call Stored Procedures to Insert Data into the Course Table	483

6.9	Insert Data into the Database Using the LINQ to DataSet Method	486
6.9.1	Insert Data Into the SQL Server Database Using the LINQ to SQL Queries	488
6.10	Chapter Summary	488
	Homework	489

Chapter 7 Data Updating and Deleting with Visual Basic.NET

493

Part I Data Updating and Deleting with Visual Studio.NET Design Tools and Wizards 494

7.1	Update or Delete Data Against Databases	495
7.1.1	Updating and Deleting Data from Related Tables in a DataSet	495
7.1.2	Update or Delete Data Against Database Using TableAdapter DBDirect Methods: TableAdapter.Update and TableAdapter.Delete	496
7.1.3	Update or Delete Data Against Database Using TableAdapter.Update Method	497
7.2	Update and Delete Data for Microsoft SQL Server Database	498
7.2.1	Create a New Project Based on the InsertWizard Project	498
7.2.2	Application User Interfaces	499
7.2.3	Validate Data before the Data Updating and Deleting	499
7.2.4	Build the Update and Delete Queries	499
7.2.4.1	Configure the TableAdapter and Build the Data Updating Query	499
7.2.4.2	Build the Data Deleting Query	500
7.2.5	Develop Codes to Update Data Using the TableAdapter DBDirect Method	502
7.2.5.1	Modifications of the Codes	502
7.2.5.2	Creations of the Codes	502
7.2.6	Develop Codes to Update Data Using the TableAdapter.Update Method	503
7.2.7	Develop Codes to Delete Data Using the TableAdapter DBDirect Method	505
7.2.8	Develop Codes to Delete Data Using the TableAdapter.Update Method	507
7.2.9	Validate the Data after the Data Updating and Deleting	508
7.3	Update and Delete Data for Oracle Database	511
7.4	Update and Delete Data for Microsoft Access Database	512

Part II Data Updating and Deleting with Runtime Objects 512

7.5	The Runtime Objects Method	513
7.6	Update and Delete Data for SQL Server Database Using the Runtime Objects	514
7.6.1	Update Data Against the Faculty Table for the SQL Server Database	515
7.6.1.1	Develop Codes to Update the Faculty Data	515
7.6.1.2	Validate the Data Updating	516

7.6.2	Delete Data from the Faculty Table for the SQL Server Database	517
7.6.2.1	Develop Codes to Delete Data	517
7.6.2.2	Validate the Data Deleting	518
7.7	Update and Delete Data for Oracle Database Using the Runtime Objects	520
7.7.1	Add the Oracle Namespace Reference and Modify the Imports Command	521
7.7.2	Modify the Connection String and Query String for the LogIn Form	522
7.7.2.1	Modify the Connection String in the Form Load Event Procedure	522
7.7.2.2	Modify the SELECT Query String in the TabLogIn Button Event Procedure	522
7.7.2.3	Modify the SELECT Query String in the ReadLogIn Button Event Procedure	523
7.7.3	Modify the Query Strings for the Faculty Form	523
7.7.3.1	Modify the SELECT Query String for the Select Button Event Procedure	523
7.7.3.2	Modify the INSERT Query String for the Insert Button Event Procedure	523
7.7.3.3	Modify the UPDATE Query String for the Update Button Event Procedure	524
7.7.3.4	Modify the DELETE Query String for the Delete Button Event Procedure	524
7.7.4	Modify the Query Strings for the Course Form	524
7.7.4.1	Modify the SELECT Query String for the Select Button Event Procedure	525
7.7.4.2	Modify the SELECT Query String for the CourseList Event Procedure	525
7.7.5	Modify the Query Strings for the Student Form	525
7.7.6	Other Modifications	525
7.8	Update and Delete Data Against Database Using Stored Procedures	528
7.8.1	Update and Delete Data Against SQL Server Database Using Stored Procedures	528
7.8.1.1	Modify the Existing Project to Create Our New Project	529
7.8.1.2	Modify the Codes to Update and Delete Data from the Faculty Table	529
7.8.1.3	Develop Two Stored Procedures in the SQL Server Database	531
7.8.1.4	Call the Stored Procedures to Perform the Data Updating and Deleting	537
7.8.2	Update and Delete Data Against Oracle Database Using Stored Procedures	538
7.8.2.1	Modify the Existing Project to Create Our New Project	538
7.8.2.2	Modify the Codes to Update and Delete Data from the Faculty Table	539
7.8.2.3	Develop Stored Procedures in the Oracle Database	542
7.8.2.4	Call the Stored Procedure to Perform the Data Updating and Deleting	546

7.8.3	Update and Delete Data Against Databases Using the LINQ to SQL Query	548
7.8.3.1	Create a New Object of the DataContext Class	549
7.8.3.2	Develop the Codes for the Select Button Click Event Procedure	550
7.8.3.3	Develop the Codes for the Update Button Click Event Procedure	551
7.8.3.4	Develop the Codes for the Delete Button Click Event Procedure	552
7.9	Chapter Summary	554
	Homework	555

Chapter 8 Accessing Data in ASP.NET	559
--	------------

8.1	What Is the .NET Framework?	560
8.2	What Is ASP.NET?	561
8.2.1	ASP.NET Web Application File Structure	563
8.2.2	ASP.NET Execution Model	563
8.2.3	What Really Happens When a Web Application Is Executed?	564
8.2.4	The Requirements to Test and Run the Web Project	565
8.3	Develop ASP.NET Web Application to Select Data from SQL Server Databases	566
8.3.1	Create the User Interface: LogIn Form	567
8.3.2	Develop the Codes to Access and Select Data from the Database	569
8.3.3	Validate the Data in the Client Side	573
8.3.4	Create the Second User Interface: Selection Page	574
8.3.5	Develop the Codes to Open the Other Page	576
8.3.6	Modify the Codes in the LogIn Page to Transfer to the Selection Page	577
8.3.7	Create the Third User Interface: Faculty Page	578
8.3.8	Develop the Codes to Select the Desired Faculty Information	580
8.3.8.1	Develop the Codes for the Page_Load Event Procedure	581
8.3.8.2	Develop the Codes for the Select Button Event Procedure	582
8.3.8.3	Develop the Codes for Other Procedures	583
8.3.9	Create the Fourth User Interface: Course Page	587
8.3.9.1	The AutoPostBack Property of the List Box Control	589
8.3.10	Develop the Codes to Select the Desired Course Information	590
8.3.10.1	Coding for the Course Page Loading and Ending Event Procedures	591
8.3.10.2	Coding for the Select Button's Click Event Procedure	592
8.3.10.3	Coding for the SelectedIndexChanged Event Procedure of the CourseList Box	594
8.3.10.4	Coding for Other User-Defined Subroutine Procedures	595
8.4	Develop ASP.NET Web Application to Insert Data into SQL Server Databases	598
8.4.1	Develop the Codes to Perform the Data Insertion Function	598
8.4.2	Develop the Codes for the Insert Button Click Event Procedure	599
8.4.3	Modify the Codes in the Subroutine ShowFaculty() for the Data Validation	601
8.4.4	Validate the Data Insertion	602

8.5	Develop Web Applications to Update and Delete Data in SQL Server Databases	604
8.5.1	Modify the Codes for the Faculty Page	605
8.5.2	Develop the Codes for the Update Button Click Event Procedure	606
8.5.3	Develop the Codes for the Delete Button Click Event Procedure	609
8.5.3.1	Relationships between Five Tables in Our Sample Database	609
8.5.3.2	Data Deleting Sequence	610
8.5.3.3	Use the Cascade Deleting Option to Simplify the Data Deleting	611
8.5.3.4	Create the Stored Procedure to Perform the Data Deleting	612
8.5.3.5	Develop the Codes to Call the Stored Procedure to Perform the Data Deleting	616
8.6	Develop ASP.NET Web Applications with LINQ to SQL Query	618
8.6.1	Add an Existing Web Page FacultyLINQ.aspx	620
8.6.2	Create a New Object of the DataContext Class	621
8.6.3	Develop the Codes for the Data Selection Query	622
8.6.4	Develop the Codes for the Data Insertion Query	623
8.6.5	Develop the Codes for the Data Updating and Deleting Queries	625
8.7	Develop ASP.NET Web Application to Select Data from Oracle Databases	628
8.7.1	Add the Oracle Database Reference and Modify Imports Commands	629
8.7.2	Modify the Connection String in the LogIn Page	629
8.7.3	Modify the Query String in the LogIn Page	630
8.7.4	Modify the Query String in the Faculty Page	631
8.7.5	Modify the Query Strings in the Course Page	633
8.7.6	Modify the Global Connection Object in the Selection Page	636
8.8	Develop ASP.NET Web Application to Insert Data into Oracle Databases	636
8.8.1	Create the Codes for the Insert Button Click Event Procedure	637
8.8.2	Create the Codes for the TextChanged Event Procedure of the Faculty ID Textbox	639
8.8.3	Modify the Codes in the Subroutine ShowFaculty() for the Data Validation	639
8.9	Develop ASP.NET Web Application to Update and Delete Data in Oracle Databases	642
8.9.1	Build the Codes for the Project to Perform the Data Updating	642
8.9.1.1	Modifications to the Select Button's Click Event Procedure	642
8.9.1.2	Add the Codes to the Update Button Event and UpdateParameters Procedures	643
8.9.2	Develop Stored Procedures to Perform the Data Deleting	646
8.9.2.1	Delete an Existing Record from the Faculty Table	646
8.9.2.2	Develop the Codes for the Delete Button's Event Procedure	647
8.9.2.3	Validate the Data Deleting Actions	649
8.9.2.4	The Constraint Property: On Delete Cascade in the Data Table	650
8.10	Chapter Summary	653
	Homework	654

Chapter 9 ASP.NET Web Services**657**

9.1	What Are Web Services and Their Components?	658
9.2	Procedures to Build a Web Service	659
9.2.1	The Structure of a Typical Web Service Project	660
9.2.2	The Real Considerations When Building a Web Service Project	660
9.2.3	Introduction to Windows Communication Foundation (WCF)	661
9.2.3.1	What Is WCF?	662
9.2.3.2	WCF Data Services	662
9.2.3.3	WCF Services	663
9.2.3.4	WCF Clients	663
9.2.3.5	WCF Hosting	664
9.2.3.6	WCF Visual Studio Templates	664
9.2.4	Procedures to Build an ASP.NET Web Service	665
9.3	Build ASP.NET Web Service Project to Access SQL Server Database	666
9.3.1	Files and Items Created in the New Web Service Project	667
9.3.2	A Feeling of the Hello World Web Service Project As it Runs	671
9.3.3	Modify the Default Namespace	674
9.3.4	Create a Base Class to Handle Error Checking for Our Web Service	675
9.3.5	Create the Real Web Service Class	676
9.3.6	Add Web Methods into Our Web Service Class	677
9.3.7	Develop the Codes for Web Methods to Perform the Web Services	678
9.3.7.1	Web Service Connection Strings	678
9.3.7.2	Modify the Existing HelloWorld Web Method	680
9.3.7.3	Develop the Codes to Perform the Database Queries	682
9.3.7.4	Develop the Codes for Subroutines Used in the Web Method	684
9.3.8	Develop the Stored Procedure to Perform the Data Query	687
9.3.8.1	Develop the Stored Procedure WebSelectFacultySP	687
9.3.8.2	Add Another Web Method to Call the Stored Procedure	688
9.3.9	Use DataSet as the Returning Object for the Web Method	689
9.3.10	Build Windows-Based Web Service Clients to Consume the Web Services	692
9.3.10.1	Create a Web Service Proxy Class	693
9.3.10.2	Develop the Graphic User Interface for the Windows-Based Client Project	695
9.3.10.3	Develop the Code to Consume the Web Service	696
9.3.11	Build Web-Based Web Service Clients to Consume the Web Service	703
9.3.11.1	Create a New Web Site Project and Add an Existing Web Page	704
9.3.11.2	Add a Web Service Reference and Modify the Web Form Window	704
9.3.11.3	Modify the Codes for the Related Event Procedures	706
9.3.12	Deploy the Completed Web Service to Production Servers	710
9.3.12.1	Copy Web Service Files to the Virtual Directory	712
9.3.12.2	Publish Precompiled Web Service	713

9.4	Build ASP.NET Web Service Project to Insert Data into SQL Server Database	714
9.4.1	Modify an Existing Web Service Project	714
9.4.2	Develop the Web Service Methods	715
9.4.3	Develop and Modify the Codes for the Code-Behind Page	716
9.4.3.1	Develop and Modify the First Web Method SetSQLInsertSP	716
9.4.3.2	Develop the Second Web Method GetSQLInsert	721
9.4.3.3	Develop the Third Web Method SQLInsertDataSet	725
9.4.3.4	Develop the Fourth Web Method GetSQLInsertCourse	729
9.4.4	Build Windows-Based Web Service Clients to Consume the Web Services	734
9.4.4.1	Create a Windows-Based Consume Project and a Web Service Proxy Class	734
9.4.4.2	Develop the Graphic User Interface for the Client Project	736
9.4.4.3	Develop the Code to Consume the Web Service	738
9.4.5	Build Web-Based Web Service Clients to Consume the Web Services	749
9.4.5.1	Create a New Web Site Project and Add an Existing Web Page	750
9.4.5.2	Add a Web Service Reference and Modify the Web Form Window	750
9.4.5.3	Modify the Codes for the Related Event Procedures	752
9.5	Build ASP.NET Web Service to Update and Delete Data for SQL Server Database	762
9.5.1	Modify an Existing Web Service Project	763
9.5.2	Guideline in Modifying Related Web Methods	764
9.5.2.1	Modify the Web Method from SetSQLInsertSP to SQLUpdateSP	764
9.5.2.2	Modify the Web Method GetSQLInsert to GetSQLCourse	766
9.5.2.3	Modify the Web Method GetSQLInsertCourse to GetSQLCourseDetail	768
9.5.2.4	Add a New Web Method SQLDeleteSP	769
9.5.3	Develop Two Stored Procedures WebUpdateCourseSP and WebDeleteCourseSP	771
9.5.3.1	Develop the Stored Procedure WebUpdateCourseSP	771
9.5.3.2	Develop the Stored Procedure WebDeleteCourseSP	774
9.6	Build Windows-Based Web Service Clients to Consume the Web Services	783
9.6.1	Modifications to the File Folder and Project Files	784
9.6.2	Add a New Web Reference to Our Client Project	784
9.6.3	Modify the Codes for the Different Event Procedures and Subroutines	786
9.6.3.1	Modify the Codes of the Form_Load Event Procedure and Form-Level Variables	786
9.6.3.2	Modify the Codes for the Select Button Click Event Procedure and Related User-defined Subroutine Procedures	787
9.6.3.3	Remove the Insert Button Click Event Procedure	788
9.6.3.4	Modify the Codes for the SelectedIndexChanged Event Procedure	788
9.6.3.5	Develop the Codes for the Update Button Event Procedure	789
9.6.3.6	Develop the Codes for the Delete Button Event Procedure	790

9.7	Build Web-Based Web Service Clients to Consume the Web Services	793
9.7.1	Create a New Web Site Project and Add an Existing Web Page	794
9.7.2	Add a Web Service Reference and Modify the Web Form Window	794
9.7.3	Modify the Codes for the Related Event Procedures and Subroutines	796
9.7.3.1	Modify the Codes in the Page_Load Event Procedure	796
9.7.3.2	Modify Codes in the Select Button Event Procedure and Related Subroutines	796
9.7.3.3	Modify the Codes in the SelectedIndexChanged Event Procedure of the Course List Box Control and Related Subroutines	798
9.7.3.4	Remove the Insert Button Click Event Procedure and the TextChanged Event Procedure of the Course ID Textbox	799
9.7.3.5	Develop Codes for the Update Button Click Event Procedure	799
9.7.3.6	Develop Codes for the Delete Button Click Event Procedure	800
9.8	Build ASP.NET Web Service Project to Access Oracle Database	804
9.8.1	Build a Web Service Project WebServiceOracleSelect	805
9.8.2	Modify the Connection String	806
9.8.3	Add Oracle Database References and Modify the Namespace Directories	806
9.8.4	Modify the Web Method GetSQLSelect and Related Subroutines	807
9.8.5	Modify the Web Method GetSQLSelectSP and Related Subroutines	809
9.8.5.1	Modifications to the Stored Procedure WebSelectFacultySP	810
9.8.5.2	Modifications to the Codes in the Web Method GetSQLSelectSP	814
9.8.5.3	Modify the Web Method GetSQLSelectDataSet	816
9.9	Build Web Service Client Projects to Consume the Web Service	819
9.10	Build ASP.NET Web Service Project to Insert Data into Oracle Database	820
9.10.1	Build a Web Service Project WebServiceOracleInsert	820
9.10.2	Modify the Connection String	821
9.10.3	Add Oracle Database Reference and Modify the Namespace Directories	822
9.10.4	Modify the Web Method SetSQLInsertSP and Related Subroutines	822
9.10.5	Modify the Web Method GetSQLInsert and Related Subroutines	825
9.10.6	Modify the Web Method SQLInsertDataSet	826
9.10.7	Modify the Web Method GetSQLInsertCourse and Related Subroutines	828
9.10.8	Build the Oracle Package WebSelectCourseSP	830
9.11	Build Web Service Client Projects to Consume the Web Service	836
9.12	Build ASP.NET Web Service to Update and Delete Data for the Oracle Database	838
9.12.1	Build a Web Service Project WebServiceOracleUpdateDelete	838
9.12.2	Modify the Connection String	839
9.12.3	Add Oracle Database Reference and Modify the Namespace Directories	839
9.12.4	Modify the Web Method SQLUpdateSP and Related Subroutines	840
9.12.4.1	Develop the Stored Procedure UpdateCourse_SP	842

9.12.5	Modify the Web Method GetSQLCourse and Related Subroutines	845
9.12.6	Modify the Web Method GetSQLCourseDetail and Related Subroutines	847
9.12.7	Modify the Web Method SQLDeleteSP	849
9.12.7.1	Develop the Stored Procedure WebDeleteCourseSP	850
9.13	Build Web Service Client Projects to Consume the Web Service	855
9.14	Chapter Summary	856
Homework		857

Index	860
--------------	------------

About the Author	868
-------------------------	------------

Preface

Databases have become an integral part of our modern day 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 albums in selected regions based on the music preference of teenagers. Grocery stores display more popular items at the 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 allow users to access and manipulate their records or data in databases. Visual Basic.NET is an ideal candidate to be selected to provide this GUI functionality. Unlike other programming languages, Visual Basic.NET is a kind of language that has advantages, such as easy-to-learn and easy-to-be-understood with little learning curves. Beginning of Visual Studio.NET 2005, 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 the Internet. ADO.NET and ASP.NET are two important submodels of .NET Framework. The former provides all components, including the 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 the ASP.NET environment to allow 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 Basic.NET and relational databases, such as Microsoft Access, SQL Server 2008, and Oracle Database 11g XE. The book provides a detailed description about the practical considerations and applications in database programming with Visual Basic.NET 2010 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 Basic.NET 2010 environment.

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

1. A novel writing style is adopted to try to attract students' or beginning programmers' interest 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. Updated database programming tools and components are covered in the book, such as .NET Framework 4.0, LINQ, ADO.NET 4.0, and ASP.NET 4.0, to enable readers to easily and quickly learn and master advanced techniques in database programming and develop professional and practical database applications.
3. A real completed sample database CSE_DEPT with three versions, Microsoft Access 2007, SQL Server 2008, and Oracle Database 11g XE, is provided and used for entire book. Step by step, a detailed illustration and description about how to design and build a practical relational database is provided.
4. Covered both fundamental and advanced database programming techniques to convenience both beginning students and experienced programmers.
5. 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 Basic.NET 2010 directly to perform data operations.
6. Good textbook for college students, and good reference book for programmers, software engineers, and academic researchers.

I sincerely hope that this book can provide useful and practical helps and guides to all readers or users who adopted this book, and I will be more than happy to know that you can develop and build professional and practical database applications with the help of this book.

YING BAI

Acknowledgments

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

Special thanks to Dr. Satish Bhalla, who is the chapter contributor for this book. Dr. Bhalla is a specialist in database programming and management, especially in SQL Server, Oracle, and DB2. Dr. Bhalla spent a lot of time to prepare materials for Chapter 2, and he is deserving of thanks.

Many thanks to Senior Editor Taisuke Soda and Associate Editor Mary Hatcher who made this book available to the public. You would not find this book in the market without their deep perspective and hard work. The same thanks are extended to the editorial team of this book. Without their contributions, it is impossible for this book to be published.

These thanks should also be extended to the following book reviewers for their precious opinions to this book:

- Dr. Xintao Wu, Associate Professor, Department of Information and Systems, University of North Carolina at Charlotte
- 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, Associate Professor, Department of Physics and Computer Science, Christopher Newport University

Last but not least, thanks should be forwarded to all people who supported me to finish this book.

Y. B.

Chapter 1

Introduction

For years, during my teaching database programming and Visual Basic.NET programming in my college, I found that it was too difficult to find a good textbook for this topic, so I had to combine a few different professional books together as references to teach this course. Most of those books are specially designed for programmers or software engineers, which cover a lot of programming strategies and huge blocks of codes, which is a terrible headache to college students or beginning programmers who are new to the Visual Basic.NET and database programming. I have to prepare my class presentations and figure out all homework and exercises for my students. I dream that one day I could find a good textbook that is suitable for college students or beginning programmers and help them to learn and master database programming with Visual Basic.NET easily and conveniently. Finally, I decided that I needed to do something for this dream myself after waiting for a long time.

Another reason for me to have this idea is the job market. As you know, most industrial and commercial companies in United States belong to database applications businesses, such as manufactures, banks, hospitals, and retails. 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 those companies, we need to create a book like this one.

Unlike most database programming books in 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 tries to use a new writing style to show readers, especially to college students, how to develop professional and practical database programs in Visual Basic.NET 2010 by using Visual Studio.NET Design Tools and Wizards related to ADO.NET 4.0, and to apply codes that are autogenerated by using Wizards. By using this new style, the headache caused by using those huge blocks of programming codes can be removed; instead, a simple and easy way to create database programs using the Design Tools can be developed to attract students' learning interest, and furthermore to enable students to build professional and practical database programming in more efficient and interesting ways.

There are so many different database programming books available on the market, but rarely can you find a book like this one, which implemented a novel writing style to

attract the students' learning interests in this topic. To meet the needs of some experienced or advanced students or software engineers, the book contains two programming methods: the interesting and easy-to-learn fundamental database programming method—Visual Studio.NET Design Tools and Wizards, and advanced database programming method—runtime object method. In the second method, all database-related objects are created and applied during or when your project is running by utilizing quite a few blocks of codes.

OUTSTANDING FEATURES ABOUT THIS BOOK

1. A novel writing style is adopted to try to attract students' or beginning programmers' interests 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. Updated database programming tools and components are covered in the book, such as .NET Framework 4.0, LINQ, ADO.NET 4.0, and ASP.NET 4.0, to enable readers to easily and quickly learn and master advanced techniques in database programming and develop professional and practical database applications.
3. A real completed sample database CSE_DEPT with three versions, Microsoft Access 2007, SQL Server 2008, and Oracle Database 11g XE, is provided and used for the entire book. Step by step, a detailed illustration and description about how to design and build a practical relational database are provided.
4. Covered both fundamental and advanced database programming techniques to convenience both beginning students and experienced programmers.
5. 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 Basic.NET 2010 directly to perform data operations.
6. Provides homework and teaching materials, and these allow instructors to organize and prepare their courses easily and rapidly, and enable students to understand what they learned better by doing something themselves.
7. Good textbook for college students and good reference book for programmers, software engineers, and academic researchers.

WHO THIS BOOK IS FOR

This book is designed for college students and software programmers who want to develop practical and commercial database programming with Visual Basic.NET and relational databases, such as Microsoft Access, SQL Server 2008, and Oracle Database 11g XE. Fundamental knowledge and understanding on Visual Basic.NET and Visual Studio.NET IDE is assumed.

WHAT THIS BOOK COVERS

Nine chapters are included in this book. The contents of each chapter can be summarized as below.

- Chapter 1 provides an introduction and summarization to 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 database are also given and discussed in this part. The procedure and components used to develop a practical relational database with three database versions, such as Microsoft Access 2007, SQL Server 2008, and Oracle Database 11g XE, are analyzed in detail with some real data tables in our sample database CSE_DEPT.
- Chapter 3 provides an introduction to the ADO.NET, which includes the architectures, organizations, and components of the ADO.NET. Detailed discussions and descriptions are provided in this chapter to give readers both fundamental and practical ideas and pictures in 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 preferred projects based on the different organizations of the ADO.NET. Four popular Data Provides, 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.
- 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 at the first part in this chapter. Some popular interfaces widely used in LINQ, such as IEnumerable, IEnumerable(Of T), IQueryable and IQueryable(Of 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 the LINQ queries that were implemented for different data sources is provided in detail.
- Starting from Chapter 5, the real database programming techniques with Visual Basic.NET, such as data selection queries, are provided and discussed. Two parts are covered in this chapter: Part I contains the detailed descriptions in 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 hiding codes that are created by Visual Basic.NET automatically when using those design tools and wizards. Therefore, the coding for this part is very simple and easy. Part II covers an advanced technique, the runtime object method, 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.
- Chapter 6 provides detailed discussions and analyses about three popular data insertion methods with three different databases—Microsoft Access 2007, SQL Server 2008, 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.
 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 object and therefore it is covered in Part II. Nine real projects are used to illustrate how to perform the data insertion into three different databases: Microsoft Access 2007, SQL Server 2008, and Oracle Database 11g XE. Some professional and practical data validation methods are also discussed in this chapter to confirm the data insertion.

4 Chapter 1 Introduction

- Chapter 7 provides discussions and analyses on three popular data updating and deleting methods with seven real project examples:
 1. Using TableAdapter DBDirect methods, such as TableAdapter.Update() and TableAdapter.Delete(), to update and delete data directly against the databases.
 2. Using 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. Using the run time object method to develop and execute the Command's method ExecuteNonQuery() to update or delete data against the database directly.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 object and it is covered in Part II. Seven real projects are used to illustrate how to perform the data updating and deleting against three different databases: Microsoft Access, SQL Server 2008, and Oracle Database 11g 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.
- Chapter 8 provides introductions and discussions about the developments and implementations of ASP.NET Web applications in Visual Basic.NET 2010 environment. At the beginning of Chapter 8, a detailed and complete description about the ASP.NET and the .NET Framework is provided, and this part is especially useful and important to students or programmers 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 from Section 8.3, the detailed development and building process of ASP.NET Web applications to access databases are discussed with six real Web application projects. Two popular databases, SQL Server and Oracle, are utilized as the target databases for those development and building processes.
- Chapter 9 provides introductions and discussions about the developments and implementations of ASP.NET Web services in Visual Basic.NET 2010 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:
 1. WebServiceSQLSelect and WebServiceOracleSelect
 2. WebServiceSQLInsert and WebServiceOracleInsert
 3. 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 consume 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. A total of 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 operating system, such as Windows 2000, XP, Vista, and Windows 7.

HOW THIS BOOK IS ORGANIZED AND HOW TO USE THIS BOOK

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

Chapters 2, 3, and 4 provide the fundamentals on database structures and components, ADO.NET and LINQ components. Starting from Chapter 5, and then to Chapters 6 and 7, each chapter is divided into two parts: fundamental part and 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, they are more suitable to students or programmers who are new to the database programming with Visual Basic.NET. Part II contains the runtime object method, and it covers a lot of coding developments to perform the different data actions against the database, and this method is more flexible and convenient to experienced programmers even a lot of coding jobs is concerned.

Chapters 8 and 9 give a full discussion and analysis about the developments and implementations 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 Internet.

Based on the organization of this book we 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 those are fundamental knowledge and techniques in database programming with Visual Basic.NET 2010. For Chapters 8 and 9, it is optional to instructors, and it depends on the time and schedule.

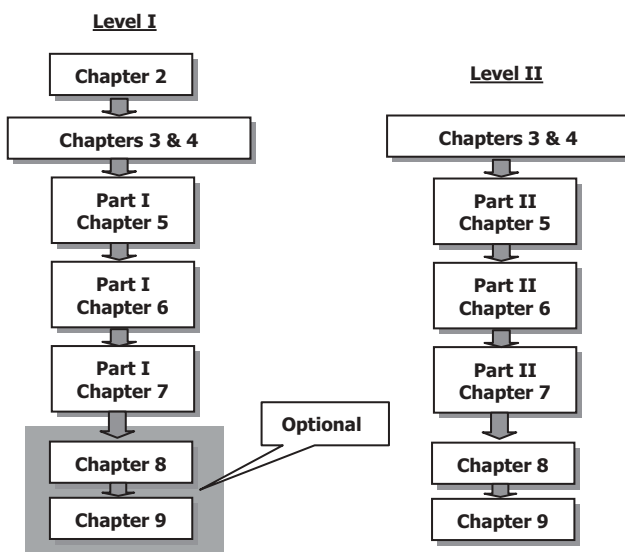


Figure 1.1. Two study levels in the book.

For experienced college students or software programmers who have already some knowledge and techniques in database programming, it is recommended to learn and understand the contents of Part II of Chapters 5–7, as well as Chapters 8 and 9, since the runtime data objects method and some sophisticated database programming techniques, such as joined-table query, nested stored procedures, and Oracle Package, are discussed and illustrated in those chapters 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 2008 and Oracle Database 11g XE.

HOW TO USE THE SOURCE CODE AND SAMPLE DATABASES

All source codes in each real project developed in this book are available. All projects are categorized into the associated chapters that are located at the folder **DBProjects** that is located at the site ftp://ftp.wiley.com/public/sci_tech_med/practical_database_vb. You can copy or download those codes into your computer and run each project as you like. To successfully run those projects on your computer, the following conditions must be met:

- Visual Studio.NET 2010 or higher must be installed in your computer.
- Three databases' management systems, Microsoft Access 2007 (Microsoft Office 2007), Microsoft SQL Server 2008 Management Studio Express, and Oracle Database 11g Express Edition (XE) must be installed in your computer.
- Three versions of sample database, CSE_DEPT.accdb, CSE_DEPT.mdf, and Oracle version of CSE_DEPT, must be installed in 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 in your computer, and it works as a pseudoserver for those projects.

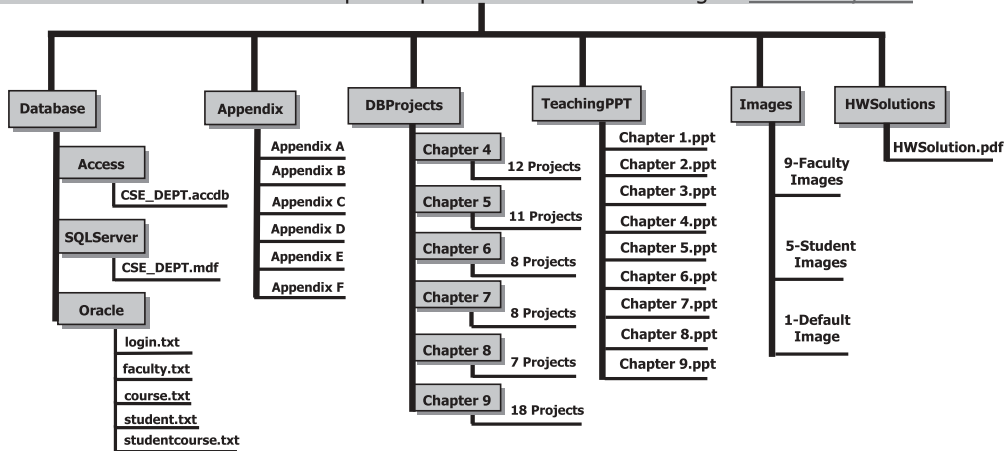
All book related teaching and learning materials, including the sample databases, example projects, appendices, faculty and student images, as well as sample Windows forms and Web pages, can be found from the associated folders located at the Wiley ftp site ftp://ftp.wiley.com/public/sci_tech_med/practical_database_vb-net-2e, as shown in Figure 1.2.

These materials are categorized and stored at different folders in two different sites based on the teaching purpose (for instructors) and learning purpose (for students):

1. **Appendix Folder:** Contains all appendices that provide useful references and practical knowledge to download and install database, database server and management systems and develop actual database application projects.
 - **Appendix A:** Provides detailed descriptions about the download and installation of Microsoft SQL Server 2008 R2 Express.
 - **Appendix B:** Provides detailed descriptions about download and installation of Oracle Database 11g Express Edition (XE).
 - **Appendix C:** Provides detailed discussions in how to use three sample databases: CSE_DEPT.accdb, CSE_DEPT.mdf, and Oracle version of CSE_DEPT.

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_vb

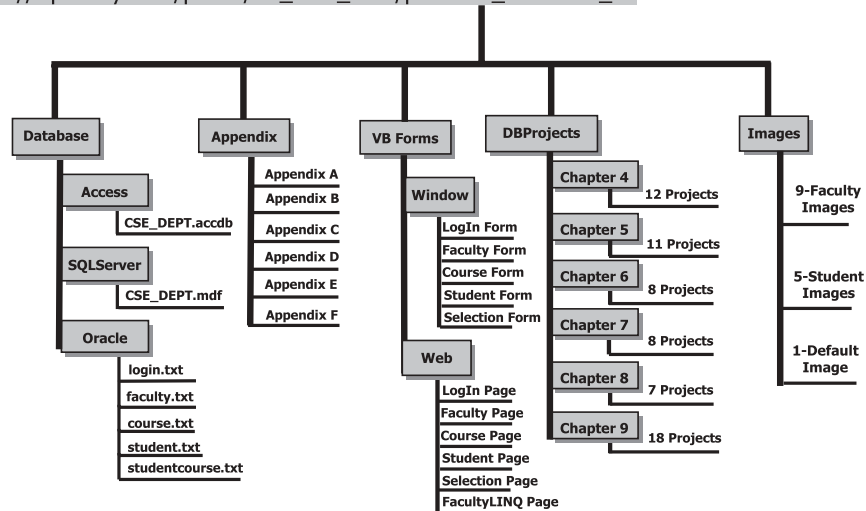


Figure 1.2. Book-related materials on website.

- **Appendix D:** Provides detailed discussions in how to create a user database in Oracle Database 11g XE using Unload and Load methods.
 - **Appendix E:** Provides detailed discussions in how to add Existing Oracle Stored Procedures into the VB Project Using the DataSet Configuration Wizard.
 - **Appendix F:** Provides detailed discussions in how to download and install a third-party Oracle Database driver **dotConnect 6.30**.
2. **Database Folder:** Contains three types of sample databases, CSE_DEPT, such as CSE_DEPT.accdb (Microsoft Access 2007), CSE_DEPT.mdf (SQL Server 2008), and Oracle version of CSE_DEPT. These sample databases are located at three subfolders, **Access**, **SQLServer**, and **Oracle**. Refer to Appendix F to get details in how to use these databases for your applications or sample projects.
 3. **DBProjects Folder:** Contains all sample projects developed in the book. Projects are categorized and stored at different chapter subfolder based on the book chapter sequence. Readers can directly use the codes and GUIs of those projects by downloading them from the DBProjects folder at the Wiley ftp site.
 4. **Images Folder:** Contains all sample faculty and student image files used in all sample projects in the book. Readers can copy and paste those image files to their projects to use them.
 5. **VB Forms Folder:** Contains all sample Windows-based forms and Web-based pages developed and implemented in all sample projects in the book. All Windows-based Forms are located at the Window subfolder, and all Web-based Pages are located at the Web subfolder. Readers can use those Forms or Pages by copying and pasting them into their real projects.
 6. **TeachingPPT Folder:** Contains all MS-PPT teaching slides for each chapter.
 7. **HWSolutions Folder:** Contains selected solutions for the homeworks developed and used in the book. The solutions are categorized and stored at the different chapter subfolder based on the book chapter sequence.

Folders 1~5 belong to learning materials for students; therefore they are located at the student site: ftp://ftp.wiley.com/public/sci_tech_med/practical_database_vb. Folders 1~7 belong to teaching materials for instructors; they are located at the Wiley teaching site and available upon requests by instructors.

INSTRUCTORS AND CUSTOMERS SUPPORTS

The teaching materials for all chapters have been extracted and represented by a sequence of Microsoft Power Point files, each file for one chapter. The interested instructors can find those teaching materials from the folder **TeachingPPT** that is located at the site <http://www.wiley.com>, and those instructor materials are available upon request from the book's listing on <http://www.wiley.com>.

A selected homework solution is also available upon request from the book's listing on <http://www.wiley.com>.

E-mail support is available to readers of this book. When you send an e-mail 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.
- How long you expect to get the answer to your questions.

Please send all questions to the e-mail address: baidbbook@gmail.com.

Detailed structure and distribution of all book-related materials in the Wiley site, including the teaching materials for instructors and learning materials for students, are shown in Figure 1.2.

Chapter 2

Introduction to Databases

SATISH BHALLA AND YING BAI

Databases have become an integral part of our modern-day 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 that was jointly sponsored by the Department of Energy and the National Institute 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 generation of huge amount of genome data, currently stored in databases around the world. The scientists will be analyzing this data in 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 albums in selected regions based on the music preference of teenagers. Grocery stores display more popular items at the eye level, and reorders are based on the inventories taken at regular intervals. Other examples include book orders by libraries, club memberships, auto part orders, winter cloth 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 and microcomputer-based data management programs, like 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:

- 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 2008 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 DBMS' 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

The 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 with 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 are 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 DBMS's became popular, the data were 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 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 are available to a large numbers 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 exist in a single database, there is a minimal need of data duplication. The duplication is needed to maintain relationship between various data items.
3. **Data Consistency and Data Integrity:** Reducing data redundancy will lead to data consistency. Since data are 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. DBA takes into account the needs of various departments and balances it against the overall need of the organization. DBA defines various rules, such as documentation standards, naming conventions, update and recovery procedures, and so on. 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 on.
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 allowed to access the database 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. Further more, since DBMS consists of a complex set of programs, trained personnel are needed to maintain it.

2.2 DEVELOP A DATABASE

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 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 so on.
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 which 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 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, 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 (ERDs) 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 key for each table. Define relationships between the tables.

4. **Physical Design:** Develop physical data structures; specify file organization, and data storage, and so on. 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 the user needs. Implement the physical design. Perform testing. Modify if necessary or until the database functions satisfactorily.

2.3 SAMPLE DATABASE

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

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

2.3.1 Relational Data Model

Data model is like a blue print for developing a database. It describes the structure of the database and various data relationships and constraints on the data. This information

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	germany	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
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

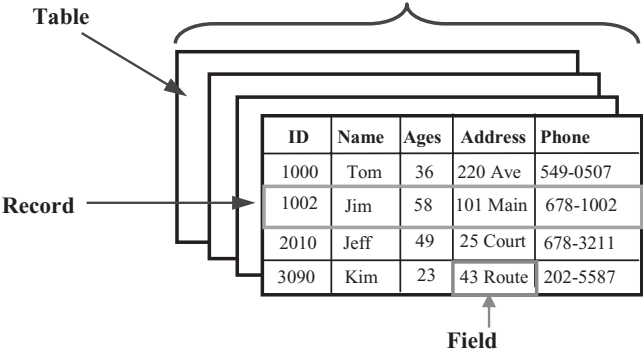


Figure 2.1. Records and fields in a table.

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 are represented correctly. In order to achieve these goals, designers use various tools. The most commonly used tool is the 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.

2.3.2 Entity-Relationship Model

The ER 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 is based on the perception that the real world is made up of entities, their attributes, and relationships. The ER is graphically depicted as ERDs. 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, and 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*. *Entity integrity rule* states that no attribute that is a member of the primary (composite) key may accept a null value.

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

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 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 key in another table. Foreign 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**, which 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**, which 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

Table 2.7. Course (Partial data shown) Faculty (Partial data shown)

course_id	course	faculty_id	faculty_id	faculty_name	office
CSC-132A	Introduction to Programming	J33486	A52990	Black Anderson	MTC-218
CSC-132B	Introduction to Programming	B78880	A77587	Debby Angles	MTC-320
CSC-230	Algorithms & Structures	A77587	B66750	Alice Brown	MTC-257
CSC-232A	Programming I	B66750	B78880	Ying Bai	MTC-211
CSC-232B	Programming I	A77587	B86590	Satish Bhalla	MTC-214
CSC-233A	Introduction to Algorithms	H99118	H99118	Jeff Henry	MTC-336
CSC-233B	Introduction to Algorithms	K69880	J33486	Steve Johnson	MTC-118
CSC-234A	Data Structure & Algorithms	B78880	K69880	Jenney King	MTC-324

reflect in the Course table, meaning that all records in the Course table must be modified using a cascade update or cascade delete (Table 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 ERDs, these are indicated by placing 1, M, or N at one of the two ends of the relationship diagram. Figures illustrate 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 (Fig. 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, one-to-many relationship is represented by **faculty_id** in the Faculty table and **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 (Fig. 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 (Fig. 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

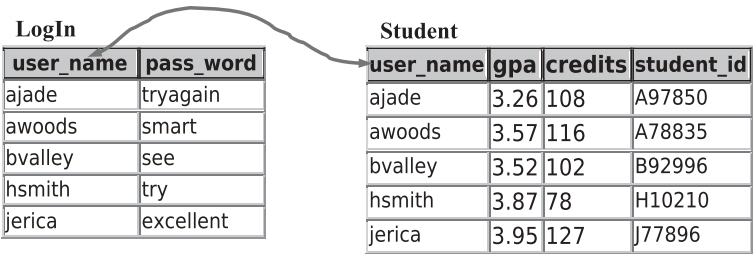


Figure 2.2. One-to-one relationship in the LogIn and the Student tables.

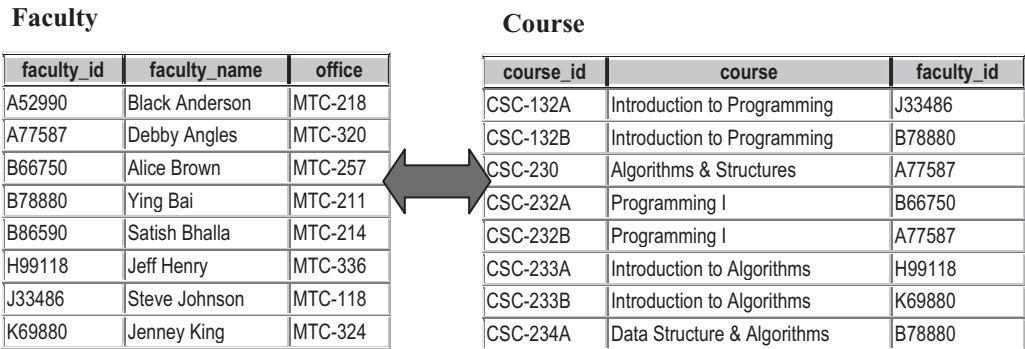


Figure 2.3. One-to-many relationship between Faculty and Course tables.

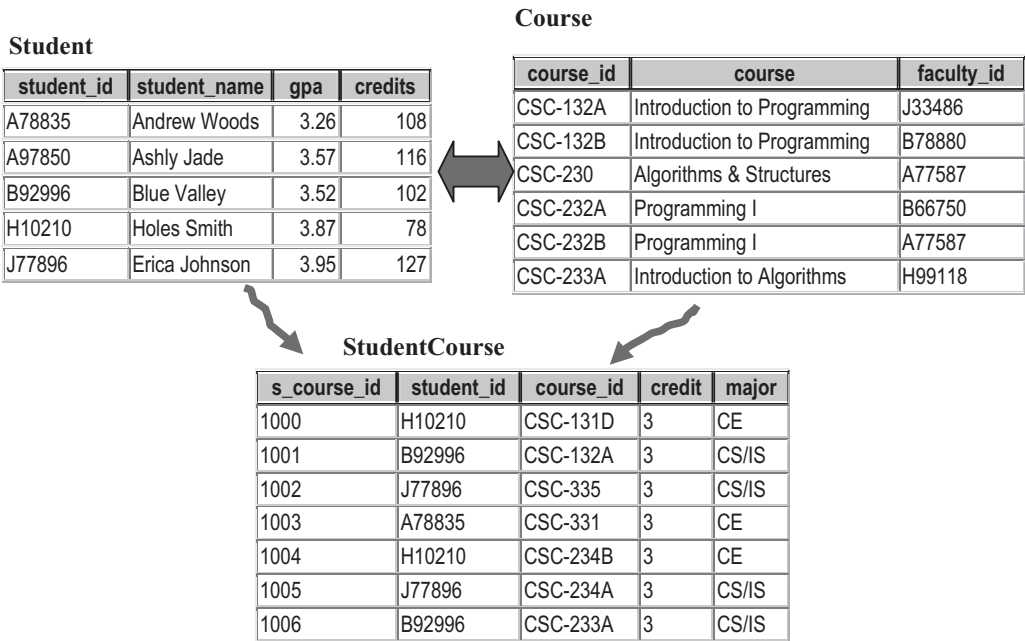


Figure 2.4. Many-to-many relationship between Student and Course tables.

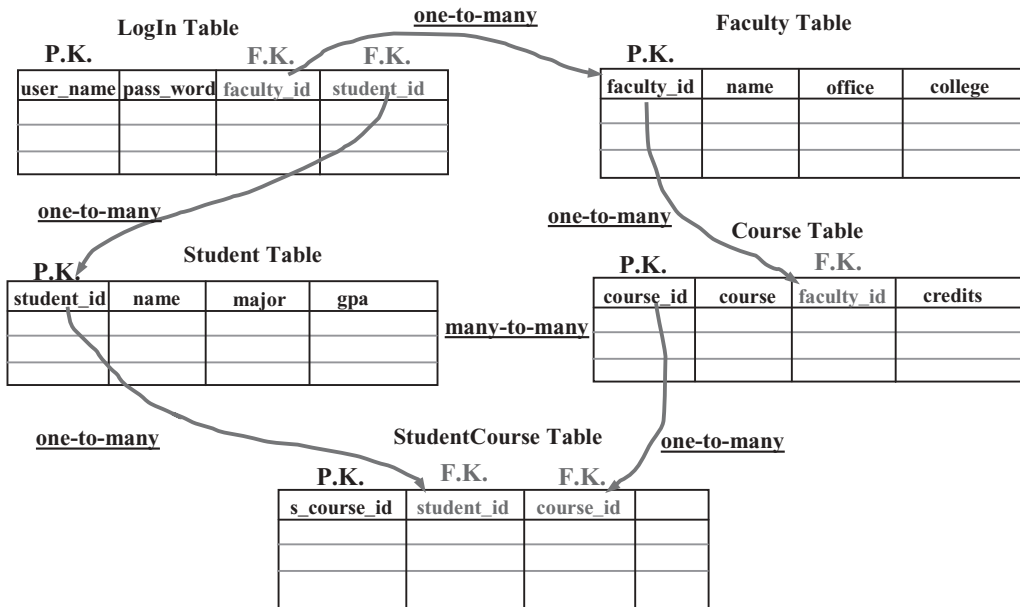


Figure 2.5. Relationships in CSE_DEPT database.

Course table, since one course may be taught by the different faculty for the different semester.

The relationships in CSE_DEPT database are summarized in Figure 2.5.

Database name: **CSE_DEPT**

The five entities are:

- LogIn
- Faculty
- Course
- Student
- StudentCourse

The relationships between these entities are shown in Figure 2.5. **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. One-to-many relationships 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, Crow's foot, and a few others. There is no consensus on the symbols and the styles used to draw ERDs.

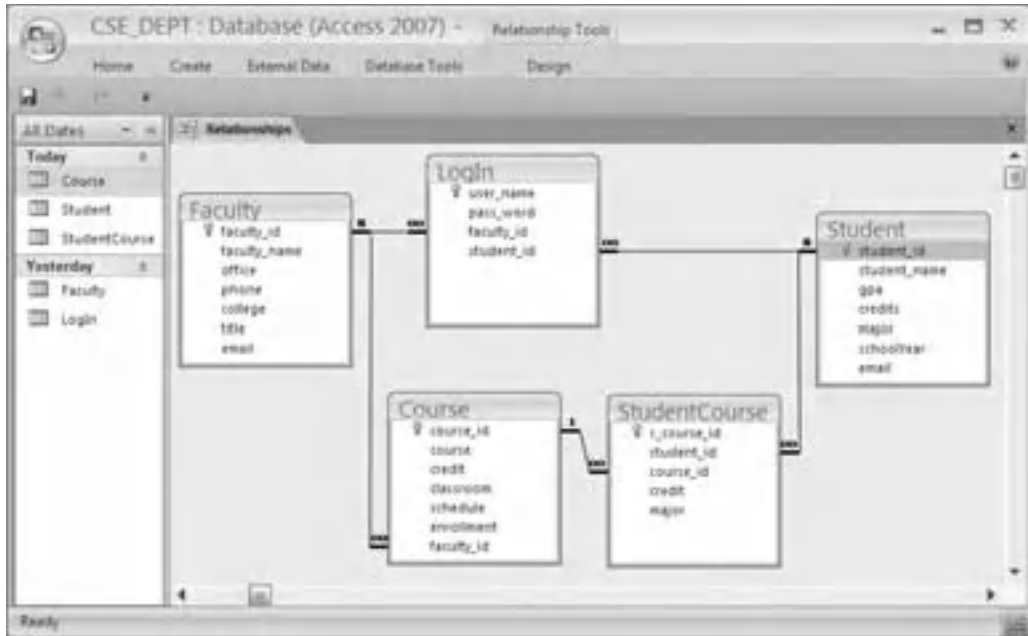


Figure 2.6. Relationships are illustrated using MS Access in the CSE_DEPT database.

A number of drawing tools are available to draw ERDs. These include ER Assistant, Microsoft Visio, 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, 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 (1NF), and the one in third normal form (3NF) is better than the one in second normal form (2NF). To be in 3NF, a database has to be in the first and second normal form. There are fourth and fifth normal forms, but for most practical purposes, a database meeting the criteria of 3NF is considered to be of good design.

2.7.1 First Normal Form (1NF)

A table is in 1NF if values in each column are atomic, that is, there are no repeating groups of data.

Entity is represented by a rectangular box.



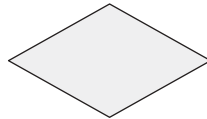
A weak entity is represented by a double rectangular box.



An attribute is represented by an oval.



Relationship is represented by a diamond with lines connecting the entities involved.



Cardinality is indicated by placing 1, N, or M near the entity it is associated with.



A line links entities to attributes and relationships.



Figure 2.7. Commonly used symbols for ER notation.

The following 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 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 1NF.

2.7.2 Second Normal Form (2NF)

A table is in 2NF 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 column then the table in 1NF is automatically in 2NF. The second part of the definition implies that if the key is

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

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 1NF. 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 (Fig. 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 only.

2.7.3 Third Normal Form (3NF)

A table is in 3NF 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.

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 faulty table into 2NF by decomposing the old table in two, Faculty and Office.**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

For the Course table in Table 2.10, all nonkey columns depend on the primary key—**course_id**. In addition, name and phone columns also depend on **faculty_id**. This table is in 2NF, but it suffers from update, addition, and deletion anomalies because of transitive dependencies. In order to conform to 3NF, 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 2.11. 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 data. Simple databases provide basic services to accomplish these tasks. Many database providers, like Microsoft SQL Server and Oracle, provide additional services, which necessitates storing many components in the database other than data. These components, such as views, stored procedures, and so on, 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 are 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, data are 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

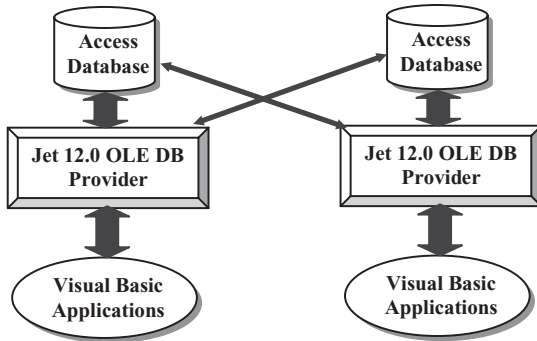


Figure 2.9. Microsoft Access database illustration.

Technology), it allows the manipulation of relational database. 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 so on. 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 Java application accesses the MS Access database via ACE OLE database provider.

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 are 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 an .mdb (Microsoft DataBase) extension. Some of these objects help the user 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 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 to create queries. This bypasses writing SQL statements and makes it appealing to beginning and not so savvy users, who can use wizards or GUI interface 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 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.

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.

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, like tables and views, and system objects. These file have an extension 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 an .ldf extension. All transactions to the database are recorded in this file.

Figure 2.10 illustrates the structure of the SQL Server Database. Each Java 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, which includes tables, keys, views, stored procedures, and others. All these objects are necessary for the efficient operation of the database.

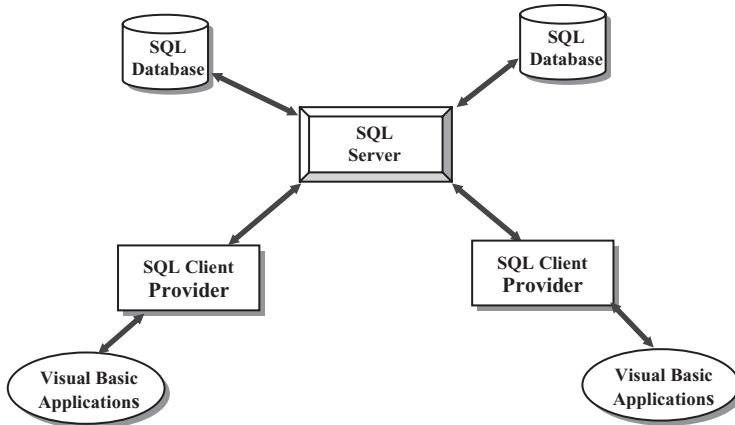


Figure 2.10. SQL Server database structure.

2.8.2.2 Tables

The data in a relational database reside 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 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 null value, that is, missing data. 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.

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. So 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 a 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 secondary level, following to 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 permit duplicate keys. Any column 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. Oracle database contains more files than 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 large installations using large number of transactions in a multiuser environment. Standard edition is also used by high-level multiuser installations. It lacks some of the utilities available in Enterprise edition. 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 log files*, 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 a *parameter file* that defines the characteristics of an Oracle instance, a *password file* used for authentication, and an *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, 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 the 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 key word **BEGIN** and consists of SQL statements. The exceptions section of the procedure begins with the keyword **EXCEPTION** and contains exception

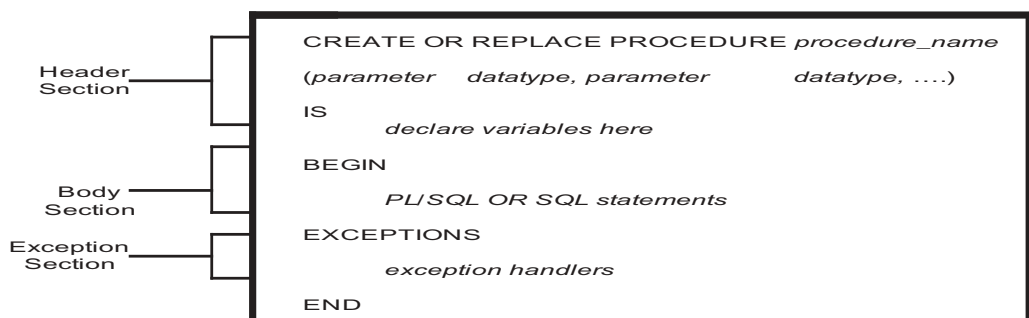


Figure 2.11. Syntax for creating a stored procedure in oracle.

handlers that are designed to handle the occurrence of some conditions that changes 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 look up 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. 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 the 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, bit maps 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: static parameter file and a persistent parameter file. An initialization parameter file contains a list of instance parameters, and the name of the database the instance is associated with, the 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 provide 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 an operating system or a password file authentication to authenticate database administrators. Oracle provides a password utility to create a password file. 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, the 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 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 the **Create** button to create this new database.

2.9.1 Create the LogIn Table

After a new blank database is created, click 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 the **OK** button to open this table in the design view. Enter the data, shown in Figure 2.12, into this design view to build our **LogIn** table.



Figure 2.12. The Design view of the LogIn table.



Starting from Office 2007, Microsoft released a new Access database format, **accdb**, which is different from old formats and contains a quite few new functionalities that the old Access formats do not have, such as allowing you to store file attachments as parts 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 and 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 these data. In this case, just enter: Primary key for the LogIn table since you need this column as the primary key for this table.

Similarly, 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 the user_name column as the primary key for this table. In the previous versions of the Microsoft Office Access, such as Office 2003 or XP, you need 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

Table 2.13. The 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	

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 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 that is shown in Figure 2.12.

Next, you need to add the data into this LogIn table. To do that, you need to open the Data Sheet 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 Data Sheet 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 that is shown in Figure 2.13. Click the Save button on the Toolbar to save this table. Then click the Close button that is located on the upper-right corner of the table to close this LogIn table.

2.9.2 Create the Faculty Table

Now, let's continue to create the second table Faculty. Click the Create menu item from the menu bar and select the Table icon from the Toolbar to create a new table. Click the Home menu item and select the Design View by clicking the drop-down arrow from the View tool on the Toolbar. Enter **Faculty** into the Table Name box of the pop-up dialog as the name for this new table, and click the 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 Field Size in the Field Properties 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 the Save tool on the Toolbar to save this table. The finished Design View of the Faculty table is shown in Figure 2.14.



CSE_DEPT : Database (Access 2007) - Table Tools

Home Create External Data Database Tools Datasheet

All Tables ▾ LogIn

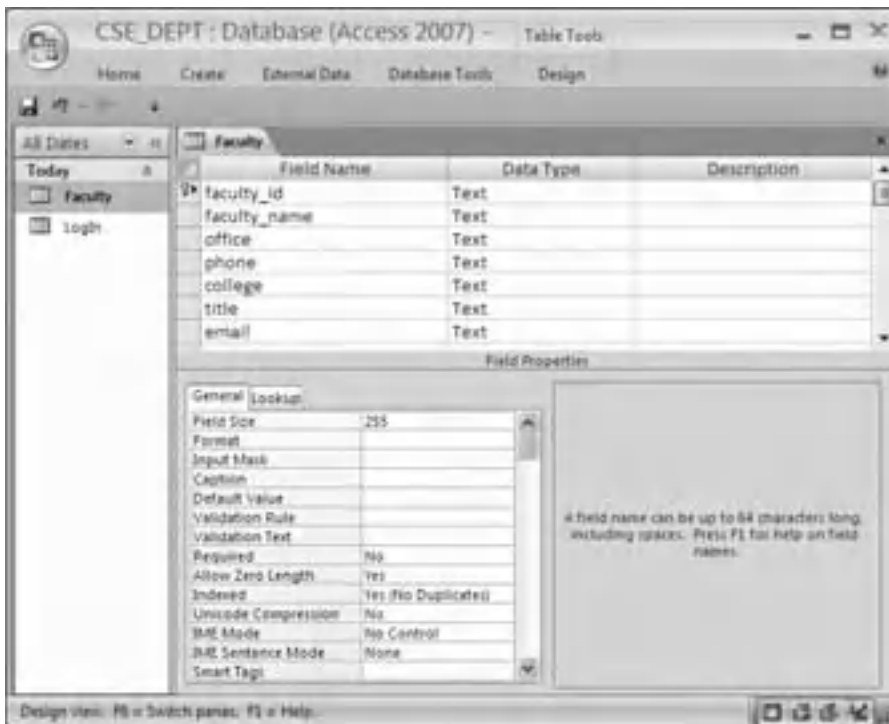
LogIn / Table

user_name	pass_word	faculty_id	student_id	Add New Field
abrown	america	866750		
ajade	tryagain		A97850	
awoods	smart		A78835	
banderson	birthday	A32990		
bvalley	see		B92996	
dangles	tomorrow	A77567		
kmuth	try		H10210	
jerica	excellent		J77896	
jhenry	test	H99113		
jking	goodman	K69880		
sbhalla	india	866590		
sjohnson	germany	J33486		
ybal	reback	878880		

Records: 13 of 13

Datasheet View

Figure 2.13. The completed LogIn table.



CSE_DEPT : Database (Access 2007) - Table Tools

Home Create External Data Database Tools Design

All Tables ▾ Faculty

Today ▾ Faculty

LogIn

Field Name	Data Type	Description
faculty_id	Text	
faculty_name	Text	
office	Text	
phone	Text	
college	Text	
title	Text	
email	Text	

Field Properties

General Lookup

Field Size	255
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	Yes (No Duplicates)
Unicode Compression	No
BRE Mode	No Control
BRE Sentence Mode	None
Smart Tags	

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Design view. F6 = Switch panes. F1 = Help.

Figure 2.14. The Design view of 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. The completed Faculty table.

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

2.9.3 Create the Other Tables

Similarly, 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 2.15–2.17). For the data type selections, follow the directions below:

The data type selections for the Course table:

- course_id—Text
- credit—Number
- enrolment—Number
- All other columns—Text

Table 2.15. The 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 2.16. The 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. The 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

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:

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

Enter the data that are shown in Tables 2.15–2.17 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.


	<div>CSE_DEPT - Database (Access 2007) - Table Tools</div> <div>Home Layout External Data Database Tools Database</div>		<div>File Edit View Database Tools</div> <div> <div> <div>Database</div> <div>Tables</div> <div>Queries</div> <div>Forms</div> <div>Reports</div> <div>Queries</div> <div>Tables</div> <div>Queries</div> <div>Forms</div> <div>Reports</div> </div> <div> <div>Database</div> <div>Tables</div> <div>Queries</div> <div>Forms</div> <div>Reports</div> <div>Queries</div> <div>Tables</div> <div>Queries</div> <div>Forms</div> <div>Reports</div> </div> </div>																																																																																																																																																																						
<div> <div>Today</div> <div>Yesterday</div> <div>Faculty</div> <div>Search</div> </div>	<table> <thead> <tr> <th>Course_ID</th> <th>Course</th> <th>Credits</th> <th>Prereq</th> <th>Prereq_ID</th> <th>Prereq_Schedule</th> <th>Prereq_Enrollment</th> <th>Prereq_Faculty_ID</th> </tr> </thead> <tbody> <tr><td>CSC-111A</td><td>Computers in Society</td><td>3</td><td>TC-108</td><td>MA-W-F: 9:00-9:30 AM</td><td>28</td><td>A52990</td><td></td></tr> <tr><td>CSC-111B</td><td>Computers in Society</td><td>3</td><td>TC-114</td><td>MA-W-F: 9:00-9:30 AM</td><td>20</td><td>866750</td><td></td></tr> <tr><td>CSC-111C</td><td>Computers in Society</td><td>3</td><td>TC-109</td><td>T-H: 11:00-12:25 PM</td><td>29</td><td>A52990</td><td></td></tr> <tr><td>CSC-111D</td><td>Computers in Society</td><td>3</td><td>TC-205</td><td>MA-W-F: 9:00-9:30 AM</td><td>10</td><td>866750</td><td></td></tr> <tr><td>CSC-111E</td><td>Computers in Society</td><td>3</td><td>TC-301</td><td>MA-W-F: 1:00-1:30 PM</td><td>25</td><td>866750</td><td></td></tr> <tr><td>CSC-111F</td><td>Computers in Society</td><td>3</td><td>TC-109</td><td>T-H: 1:00-2:25 PM</td><td>12</td><td>A52990</td><td></td></tr> <tr><td>CSC-112A</td><td>Introduction to Programming</td><td>3</td><td>TC-303</td><td>MA-W-F: 9:00-9:30 AM</td><td>21</td><td>113486</td><td></td></tr> <tr><td>CSC-112B</td><td>Introduction to Programming</td><td>3</td><td>TC-303</td><td>T-H: 1:00-2:25 PM</td><td>21</td><td>866750</td><td></td></tr> <tr><td>CSC-210</td><td>Algorithms & Structures</td><td>4</td><td>TC-304</td><td>MA-W-F: 1:00-1:30 PM</td><td>20</td><td>A77587</td><td></td></tr> <tr><td>CSC-212A</td><td>Programming I</td><td>3</td><td>TC-305</td><td>T-H: 11:00-12:25 PM</td><td>28</td><td>866750</td><td></td></tr> <tr><td>CSC-212B</td><td>Programming I</td><td>3</td><td>TC-305</td><td>T-H: 11:00-12:25 PM</td><td>17</td><td>A77587</td><td></td></tr> <tr><td>CSC-215A</td><td>Introduction to Algorithms</td><td>3</td><td>TC-302</td><td>MA-W-F: 9:00-9:30 AM</td><td>15</td><td>H99118</td><td></td></tr> <tr><td>CSC-215B</td><td>Introduction to Algorithms</td><td>3</td><td>TC-302</td><td>MA-W-F: 11:00-11:30 AM</td><td>19</td><td>866750</td><td></td></tr> <tr><td>CSC-216A</td><td>Data Structure & Algorithms</td><td>3</td><td>TC-302</td><td>MA-W-F: 9:00-9:30 AM</td><td>20</td><td>866750</td><td></td></tr> <tr><td>CSC-216B</td><td>Data Structure & Algorithms</td><td>3</td><td>TC-114</td><td>T-H: 11:00-12:25 PM</td><td>15</td><td>113486</td><td></td></tr> <tr><td>CSC-242</td><td>Programming II</td><td>4</td><td>TC-303</td><td>T-H: 1:00-2:25 PM</td><td>18</td><td>A52990</td><td></td></tr> <tr><td>CSC-320</td><td>Object Oriented Programming</td><td>3</td><td>TC-301</td><td>T-H: 1:00-2:25 PM</td><td>12</td><td>866750</td><td></td></tr> <tr><td>CSC-331</td><td>Applications Programming</td><td>3</td><td>TC-109</td><td>T-H: 11:00-12:25 PM</td><td>29</td><td>H99118</td><td></td></tr> <tr><td>CSC-333A</td><td>Computer Arch & Algorithms</td><td>3</td><td>TC-301</td><td>MA-W-F: 10:00-10:30 AM</td><td>21</td><td>A77587</td><td></td></tr> <tr><td>CSC-333B</td><td>Computer Arch & Algorithms</td><td>3</td><td>TC-303</td><td>T-H: 11:00-12:25 PM</td><td>15</td><td>A77587</td><td></td></tr> </tbody> </table> <div>Record: 11 of 27 of 27</div>	Course_ID	Course	Credits	Prereq	Prereq_ID	Prereq_Schedule	Prereq_Enrollment	Prereq_Faculty_ID	CSC-111A	Computers in Society	3	TC-108	MA-W-F: 9:00-9:30 AM	28	A52990		CSC-111B	Computers in Society	3	TC-114	MA-W-F: 9:00-9:30 AM	20	866750		CSC-111C	Computers in Society	3	TC-109	T-H: 11:00-12:25 PM	29	A52990		CSC-111D	Computers in Society	3	TC-205	MA-W-F: 9:00-9:30 AM	10	866750		CSC-111E	Computers in Society	3	TC-301	MA-W-F: 1:00-1:30 PM	25	866750		CSC-111F	Computers in Society	3	TC-109	T-H: 1:00-2:25 PM	12	A52990		CSC-112A	Introduction to Programming	3	TC-303	MA-W-F: 9:00-9:30 AM	21	113486		CSC-112B	Introduction to Programming	3	TC-303	T-H: 1:00-2:25 PM	21	866750		CSC-210	Algorithms & Structures	4	TC-304	MA-W-F: 1:00-1:30 PM	20	A77587		CSC-212A	Programming I	3	TC-305	T-H: 11:00-12:25 PM	28	866750		CSC-212B	Programming I	3	TC-305	T-H: 11:00-12:25 PM	17	A77587		CSC-215A	Introduction to Algorithms	3	TC-302	MA-W-F: 9:00-9:30 AM	15	H99118		CSC-215B	Introduction to Algorithms	3	TC-302	MA-W-F: 11:00-11:30 AM	19	866750		CSC-216A	Data Structure & Algorithms	3	TC-302	MA-W-F: 9:00-9:30 AM	20	866750		CSC-216B	Data Structure & Algorithms	3	TC-114	T-H: 11:00-12:25 PM	15	113486		CSC-242	Programming II	4	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	12	866750		CSC-331	Applications Programming	3	TC-109	T-H: 11:00-12:25 PM	29	H99118		CSC-333A	Computer Arch & Algorithms	3	TC-301	MA-W-F: 10:00-10:30 AM	21	A77587		CSC-333B	Computer Arch & Algorithms	3	TC-303	T-H: 11:00-12:25 PM	15	A77587	
Course_ID	Course	Credits	Prereq	Prereq_ID	Prereq_Schedule	Prereq_Enrollment	Prereq_Faculty_ID																																																																																																																																																																		
CSC-111A	Computers in Society	3	TC-108	MA-W-F: 9:00-9:30 AM	28	A52990																																																																																																																																																																			
CSC-111B	Computers in Society	3	TC-114	MA-W-F: 9:00-9:30 AM	20	866750																																																																																																																																																																			
CSC-111C	Computers in Society	3	TC-109	T-H: 11:00-12:25 PM	29	A52990																																																																																																																																																																			
CSC-111D	Computers in Society	3	TC-205	MA-W-F: 9:00-9:30 AM	10	866750																																																																																																																																																																			
CSC-111E	Computers in Society	3	TC-301	MA-W-F: 1:00-1:30 PM	25	866750																																																																																																																																																																			
CSC-111F	Computers in Society	3	TC-109	T-H: 1:00-2:25 PM	12	A52990																																																																																																																																																																			
CSC-112A	Introduction to Programming	3	TC-303	MA-W-F: 9:00-9:30 AM	21	113486																																																																																																																																																																			
CSC-112B	Introduction to Programming	3	TC-303	T-H: 1:00-2:25 PM	21	866750																																																																																																																																																																			
CSC-210	Algorithms & Structures	4	TC-304	MA-W-F: 1:00-1:30 PM	20	A77587																																																																																																																																																																			
CSC-212A	Programming I	3	TC-305	T-H: 11:00-12:25 PM	28	866750																																																																																																																																																																			
CSC-212B	Programming I	3	TC-305	T-H: 11:00-12:25 PM	17	A77587																																																																																																																																																																			
CSC-215A	Introduction to Algorithms	3	TC-302	MA-W-F: 9:00-9:30 AM	15	H99118																																																																																																																																																																			
CSC-215B	Introduction to Algorithms	3	TC-302	MA-W-F: 11:00-11:30 AM	19	866750																																																																																																																																																																			
CSC-216A	Data Structure & Algorithms	3	TC-302	MA-W-F: 9:00-9:30 AM	20	866750																																																																																																																																																																			
CSC-216B	Data Structure & Algorithms	3	TC-114	T-H: 11:00-12:25 PM	15	113486																																																																																																																																																																			
CSC-242	Programming II	4	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	12	866750																																																																																																																																																																			
CSC-331	Applications Programming	3	TC-109	T-H: 11:00-12:25 PM	29	H99118																																																																																																																																																																			
CSC-333A	Computer Arch & Algorithms	3	TC-301	MA-W-F: 10:00-10:30 AM	21	A77587																																																																																																																																																																			
CSC-333B	Computer Arch & Algorithms	3	TC-303	T-H: 11:00-12:25 PM	15	A77587																																																																																																																																																																			

Figure 2.16. The 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	3.37	116	Information System Engineering	Junior	ajade@college.edu
B92996	Blue Valley	3.32	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. The completed Student table.

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 the last table—StudentCourse. Click 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.

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-429	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	A57850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE
1018	A57850	CSC-331	3	ISE
1019	A97850	CSC-325	3	ISE
1020	J77896	CSE-429	3	CS/IS

Figure 2.18. The completed StudentCourse table.

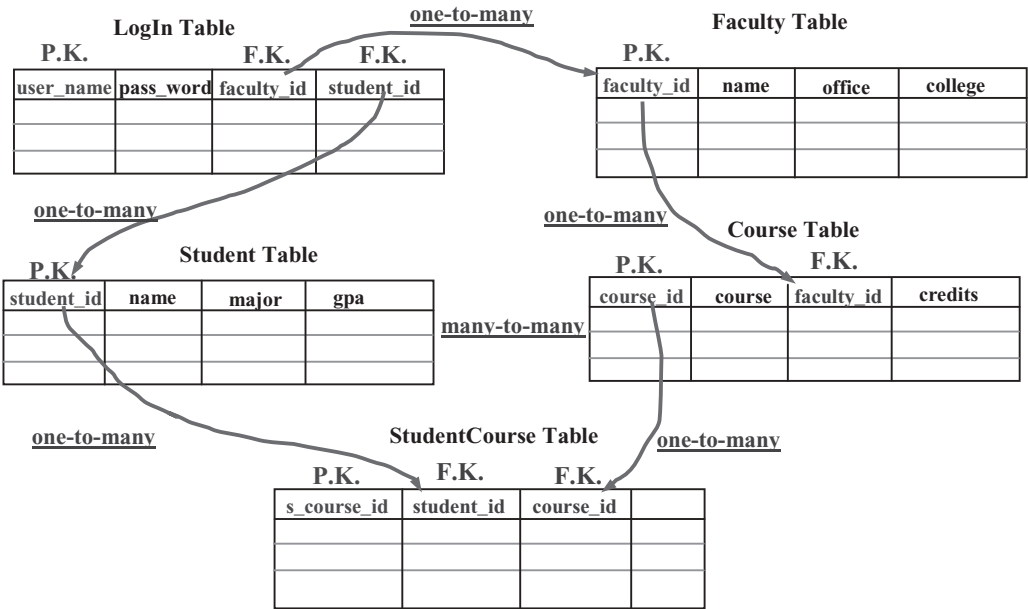


Figure 2.19. Relationships between tables.



Figure 2.20. Edit Relationships dialog box.

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 are one-to-many, since the unique primary key `faculty_id` in the Faculty table can be connected to multiple foreign key, that is, `faculty_id` located in the LogIn table.

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. Similarly, you can create all other relationships between these five tables. One point you need to remember when you perform this dragging operation is that 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` in the left is the Primary key, and the `faculty_id` in the right is the Foreign key, respectively.

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

A completed Microsoft Access 2007 database file `CSE_DEPT.accdb` can be found in the folder `Database\Access` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Refer to Appendix C if you want to use this sample database in your applications.

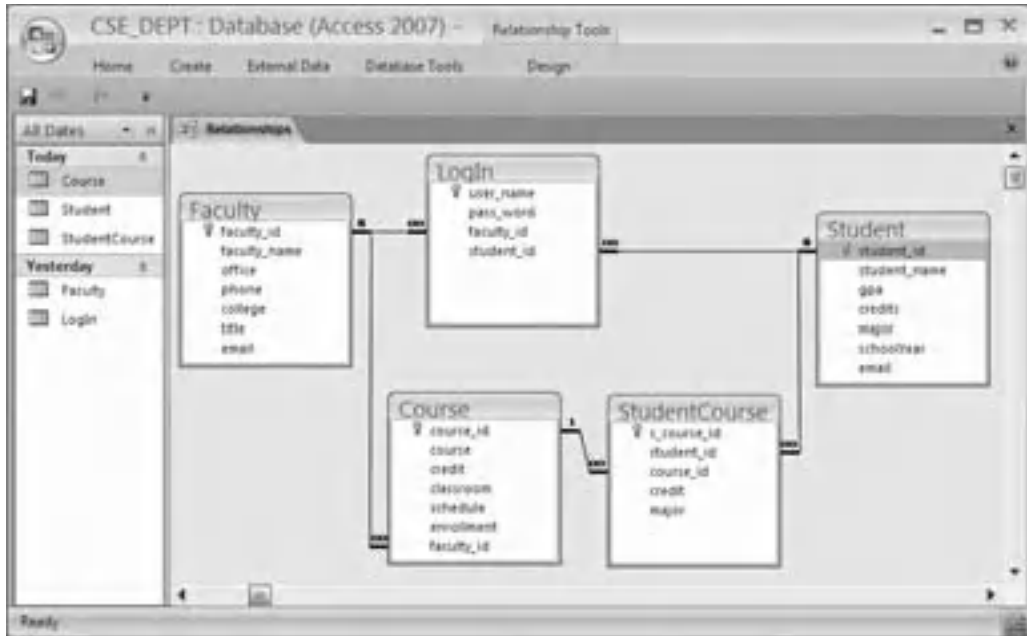


Figure 2.21. The completed relationships for tables.



During the process of creating relationships between tables, sometimes an error message may be displayed to indicate that some of the 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.

2.10 CREATE MICROSOFT SQL SERVER 2008 SAMPLE DATABASE

After you finished the installation of SQL Server 2008 Management Studio (refer to Appendix A), 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 2008** and select **SQL Server Management Studio**. 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 SMART\SQL2008EXPRESS. 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 register for your computer. Click the **Connect** button to connect your client to your server.



Figure 2.22. Connect to the SQL Server 2008.

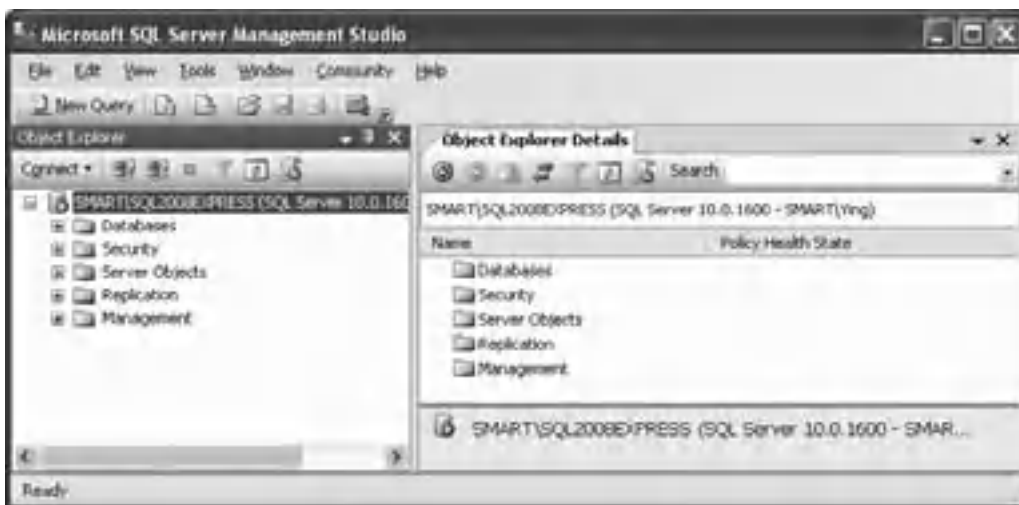


Figure 2.23. The opened server management studio.

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

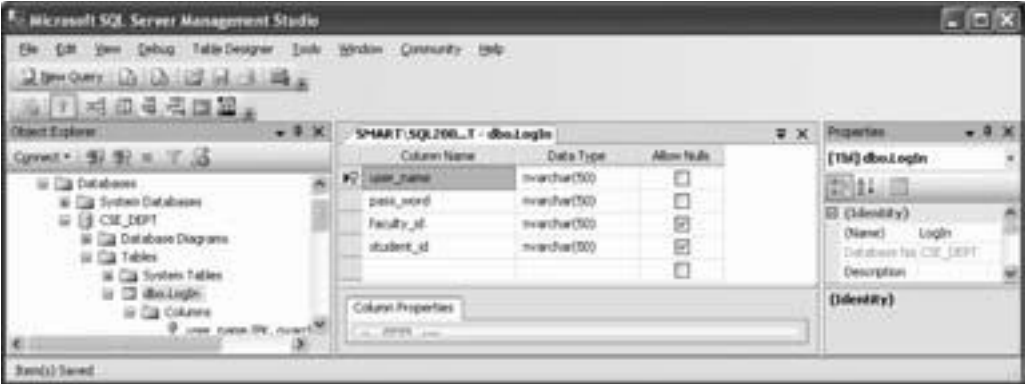


Figure 2.24. The new table window.

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 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 the 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 allows you to enter the name, the data type, and check mark 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 must 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 four Column Names columns. The data types for these four columns are all nvarchar(50), which means that this is a varied char type with a maximum of 50 letters. 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 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 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 **Edit Top 200 Rows** item from the pop-up menu. Enter all login data that are shown in Table 2.18

Table 2.18. The 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
sballa	india	B86590	NULL
sjohnson	jermany	J33486	NULL
ybai	reback	B78880	NULL

**Figure 2.25.** The finished LogIn table.

into this table. Your finished LogIn table should match the one that is shown in Figure 2.25.

One point to be noted 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 **FileSave All** item to save this table. Now let's continue to create the second table **Faculty**.

2.10.2 Create the 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 types for the columns `faculty_id` and `faculty_name` are `nvarchar(50)`, and all other data types can be either `text` or `nvarchar(50)`, 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 that is 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 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 that are shown in Table 2.19 into this Faculty table.

Your finished Faculty table should match the one that is shown in Figure 2.27.

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` that has been added into the new database CSE_DEPT under the folder Tables in the Object Explorer window.

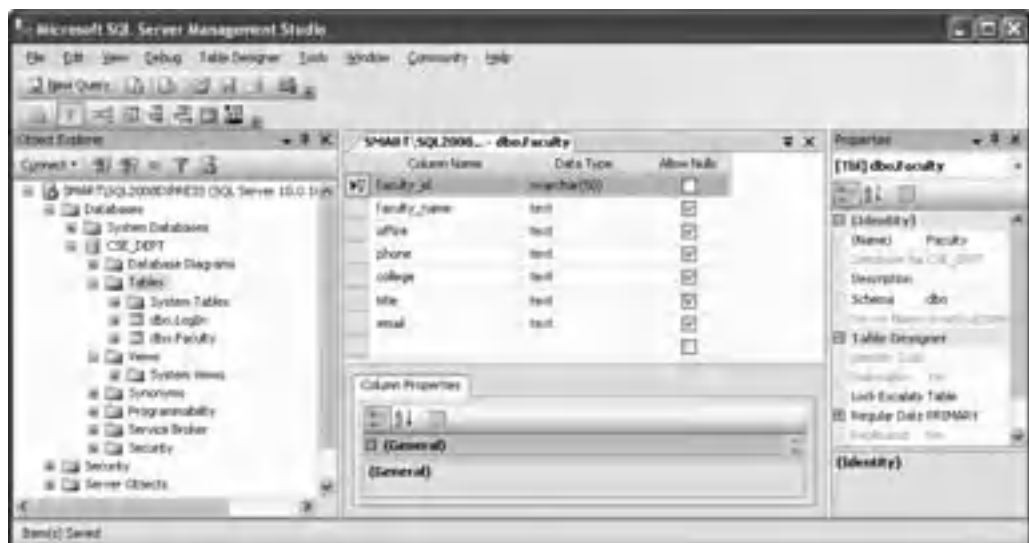
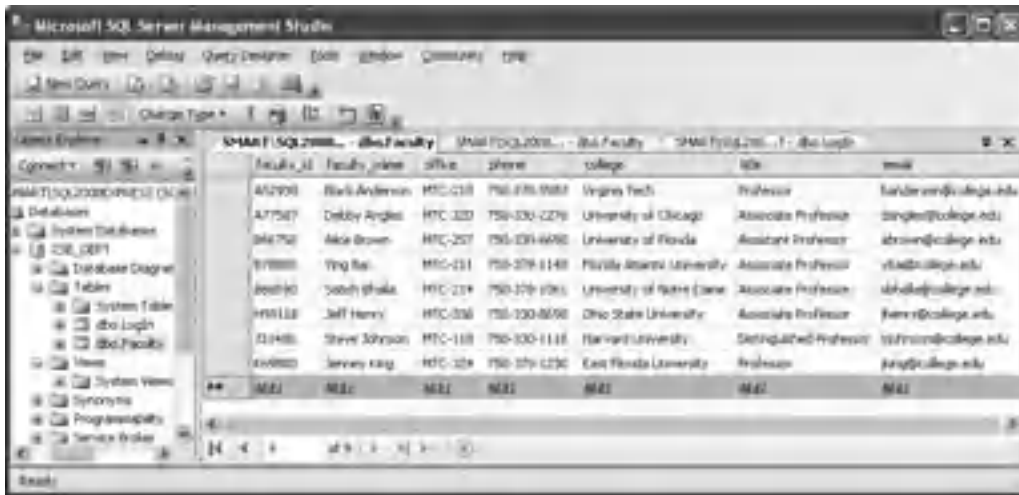


Figure 2.26. The design view of the Faculty table.

Table 2.19. The 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.** The completed Faculty table.

2.10.3 Create Other Tables

Similarly, you need to create the rest of three 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 2.20–2.22). For the data type selections, follow the directions below:

The data type selections for the Course table:

- `course_id`—`nvarchar(50)` (Primary key)
- `credit`—`smallint`
- `enrolment`—`int`
- `faculty_id`—`nvarchar(50)`
- All other columns—either `nvarchar(50)` or `text`

The data type selections for the Student table:

- `student_id`—`nvarchar(50)` (Primary key)
- `student_name`—`nvarchar(50)`
- `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

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

Table 2.20. The 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. The 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. The 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 finished Course table should match the one that is shown in Figure 2.28.

The finished Student table should match the one that is shown in Figure 2.29. The finished StudentCourse table should match the one that is shown 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—table in the Microsoft SQL Server database, and then select all data

Microsoft SQL Server Enterprise Manager

Server: SMARTSQL2008; Database: SMARTSQL2008; Table: Course

course_id	course	credit	discussion	schedule	enrollment	faculty_id
CSC-110A	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	25	A7C994
CSC-110B	Computers in Society	3	TC-114	M-W-F: 9:00-9:55 AM	20	B66750
CSC-110C	Computers in Society	3	TC-109	T-We: 11:00-12:25 PM	25	A6C994
CSC-110D	Computers in Society	3	TC-109	M-W-F: 9:00-9:55 AM	30	B66750
CSC-110E	Computers in Society	3	TC-001	M-W-F: 1:00-1:55 PM	25	B66750
CSC-111	Computers in Society	3	TC-109	T-We: 1:00-2:25 PM	30	A6C994
CSC-112A	Introduction to Programming	3	TC-300	M-W-F: 9:00-9:55 AM	21	315406
CSC-112B	Introduction to Programming	3	TC-302	T-We: 1:00-2:25 PM	21	B70880
CSC-210	Algorithms & Structures	3	TC-305	M-W-F: 9:00-9:55 AM	20	A77987
CSC-210A	Programming I	3	TC-305	T-We: 11:00-12:25 PM	20	B66750
CSC-210B	Programming I	3	TC-305	T-We: 1:00-2:25 PM	17	A77987
CSC-210A	Introduction to Algorithms	3	TC-300	M-W-F: 9:00-9:55 AM	18	B69113
CSC-210B	Introduction to Algorithms	3	TC-302	M-W-F: 11:00-11:55 AM	13	B68800
CSC-210A	Data Structure & Algorithms	3	TC-302	M-W-F: 9:00-9:55 AM	25	B70880
CSC-210B	Data Structure & Algorithms	3	TC-114	T-We: 11:00-12:25 PM	15	315406
CSC-240	Programming II	3	TC-303	T-We: 1:00-2:25 PM	18	A6C994
CSC-250	Object Oriented Programming	3	TC-303	T-We: 1:00-2:25 PM	22	B66750
CSC-310	Applications Programming	3	TC-109	T-We: 11:00-12:25 PM	18	B69113
CSC-310A	Computer Arch & Algorithms	3	TC-303	M-W-F: 10:00-10:55 AM	22	A77987
CSC-310B	Computer Arch & Algorithms	3	TC-302	T-We: 11:00-12:25 PM	15	A77987
CSC-315	Internet Programming	3	TC-303	M-W-F: 1:00-1:55 PM	25	B66750

Figure 2.28. The completed Course table.

Microsoft SQL Server Enterprise Manager

Server: SMARTSQL2008; Database: SMARTSQL2008; Table: Student

student_id	student_name	gpa	credits	major	schoolyear	email
A78285	Andrew Woods	3.26	128	Computer Science	Senior	awoods@college.edu
A77983	Ashley Jade	3.57	110	Information System Engineering	Junior	ajade@college.edu
B6C996	Blue Valley	3.52	102	Computer Science	Senior	blvalley@college.edu
H6C993	Haley Smith	3.87	98	Computer Engineering	Sophomore	hsmith@college.edu
379896	Erica Schmidt	3.95	127	Computer Science	Senior	eschmidt@college.edu
NA01	NA01	NA01	NA01	NA01	NA01	NA01

Figure 2.29. The completed Student table.

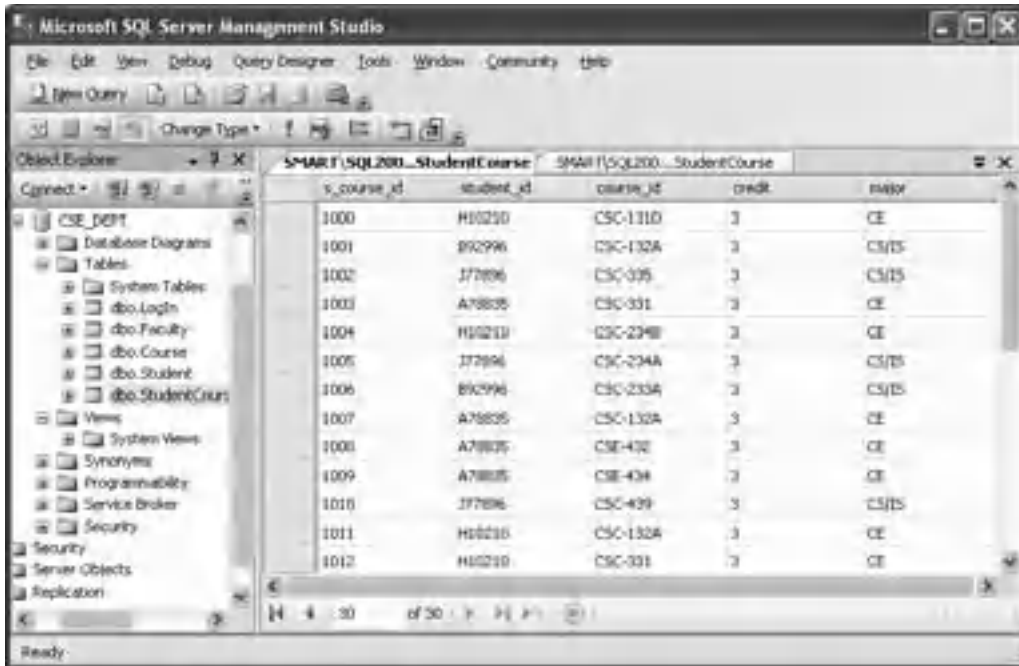


Figure 2.30. The completed StudentCourse table.



Figure 2.31. An error message when performing a paste job.

rows from your source table—Microsoft Access database file by highlighting them, and choose the Copy menu item. Next, you need to paste those rows by clicking that blank row in the Microsoft SQL Server database and then click the Paste item from the Edit menu item. An error message may be displayed as shown in Figure 2.31.

Just click OK button and your data will be pasted to your destination table without problem. The reason for that error message is because of the primary key that 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 your source row that will be pasted in this column or not.

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 Microsoft SQL Server 2008 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 the LogIn and the 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 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, exists 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 that is under the Identity pane, and then press the Enter key from 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 that is located on the right of the Tables And

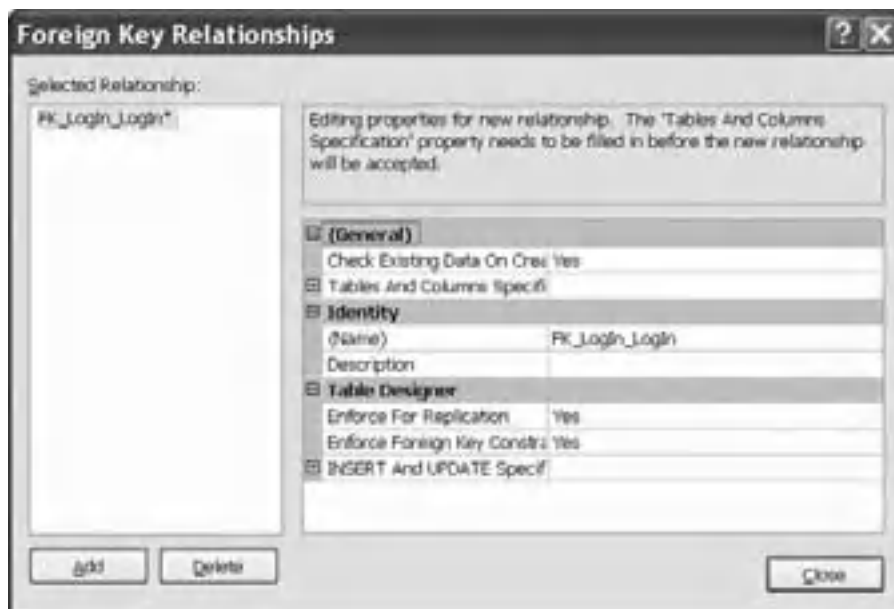


Figure 2.32. The opened Foreign Key Relationships dialog box.

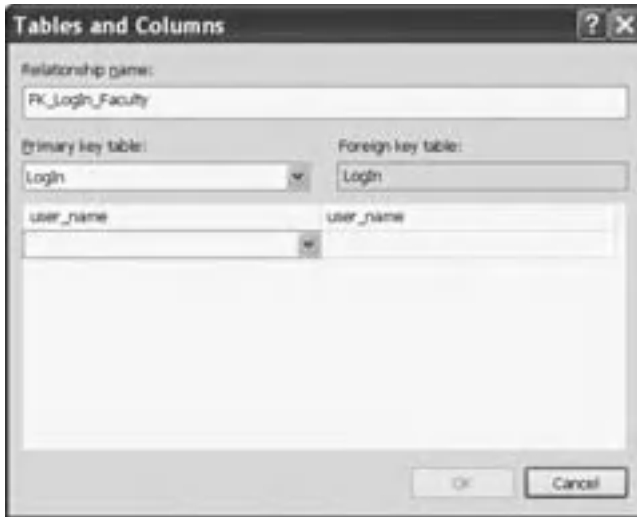


Figure 2.33. The opened Tables and Columns dialog box.

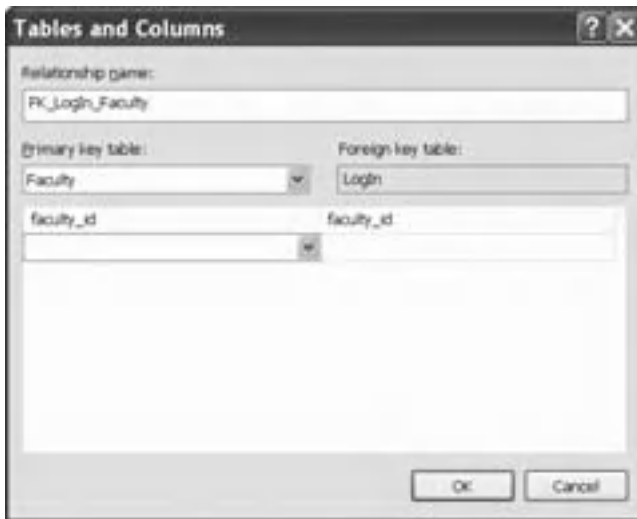


Figure 2.34. The finished Tables and Columns dialog box.

Columns Specification item to open the Tables and Columns dialog, which is shown in Figure 2.33.

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

Click on the OK button to close this dialog.

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

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

- Delete Rule
- Update Rule

The default value for both subitems is 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 that is 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 the Close button to close this dialog.

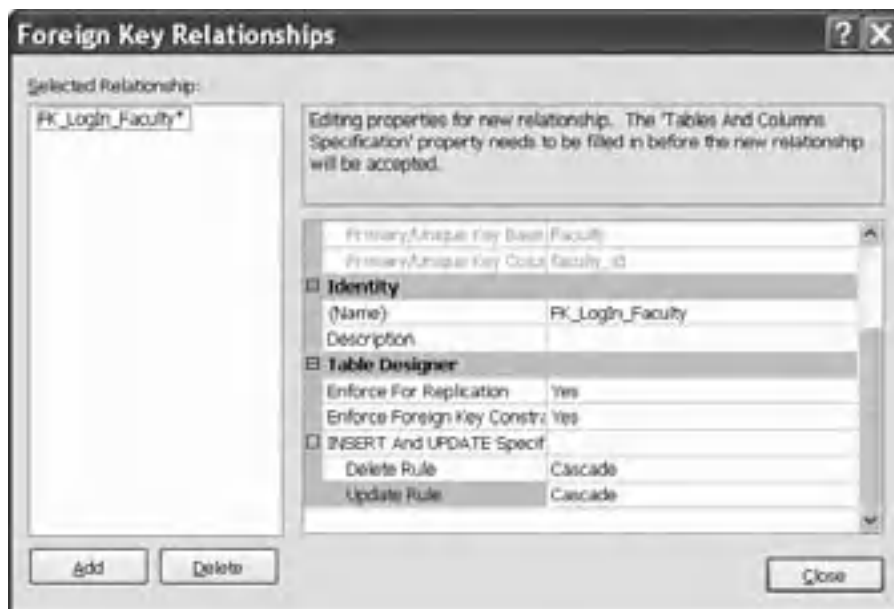


Figure 2.35. The finished Foreign Key Relationships dialog.

Go to the **File | Save LogIn** menu item to open the Save dialog, and click 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 that a new foreign key named **FK_LogIn_Faculty** has appeared under this **Keys** folder. This is our newly created foreign key, which sets the relationship between our **LogIn** and **Faculty** tables. You can confirm and find this newly created foreign key by right-clicking on the **Keys** folder that is under the **Faculty** table.

2.10.4.2 Create Relationship between the LogIn and the Student Tables

Similarly, 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 the **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 from 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** combo box and **student_id** from the box that is under the **Primary key table** combo box. Select the **student_id** from the box that is under the **Foreign key table** combo box. Your finished **Tables and Columns** dialog should match the one that is shown in Figure 2.36.

Click 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

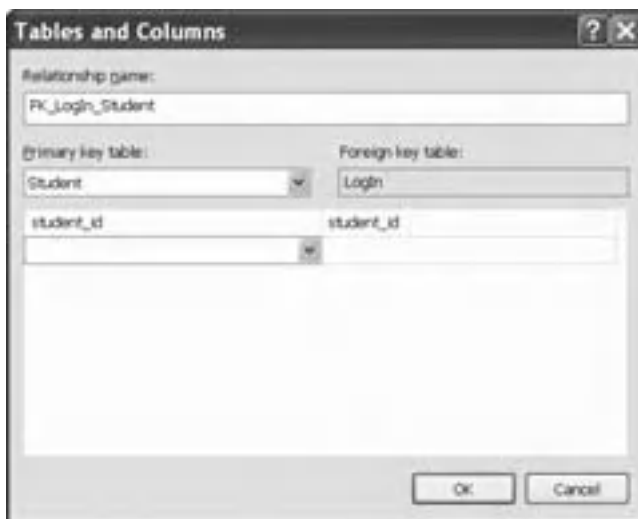


Figure 2.36. The completed **Tables and Columns** dialog.

the INSERT And UPDATE Specifications items, respectively. Click the Close button to close the Foreign Key Relationships dialog box.

Go to the File|Save LogIn menu item to save this relationship. Click 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 newly created foreign key FK_LogIn_Student.

2.10.4.3 Create Relationship between the Faculty and the 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.

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 from the keyboard. In the opened Tables and Columns dialog box, select the Faculty table from the Primary key table combo box and select the faculty_id from the box that is just under the Primary key table combo box. Then select the faculty_id from the box that is just under the Foreign key table combo box. Your finished Tables and Columns dialog should match the one that is shown in Figure 2.37.

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

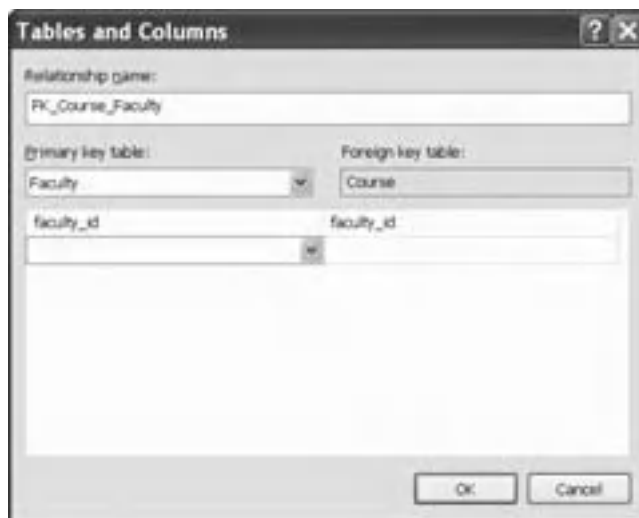


Figure 2.37. The finished Tables and Columns dialog.

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

2.10.4.4 Create Relationship between the Student and the 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 from the keyboard. In the opened Tables and Columns dialog box, select the Student table from the Primary key table combo box and select the `student_id` from the box that is just under the Primary key table combo box. Then select the `student_id` from the box that is just under the Foreign key table combo box. The finished Tables and Columns dialog should match the one that is shown in Figure 2.38.

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

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

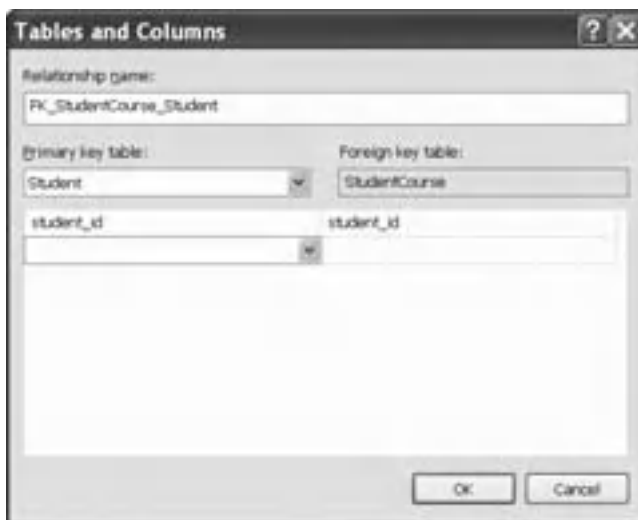


Figure 2.38. The finished Tables and Columns dialog.

2.10.4.5 Create Relationship between the Course and the 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 from the keyboard. In the opened Tables and Columns dialog box, select the Course table from the Primary key table combo box and select the `course_id` from the box that is just under the Primary key table combo box. Then select the `course_id` from the box that is just under the Foreign key table combo box. Your finished Tables and Columns dialog should match the one that is shown in Figure 2.39.

Click the OK button 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 the Close button to close the Foreign Key Relationships dialog box. Then go to the File | Save StudentCourse menu item and click 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 newly created relationship key `FK_StudentCourse_Course`.

At this point, we complete setting the relationships among our five data tables.

A completed Microsoft SQL Server 2008 sample database file `CSE_DEPT.mdf` can be found in the folder `Database\SQLServer` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). The completed relationships for these tables are shown in Figure 2.40.

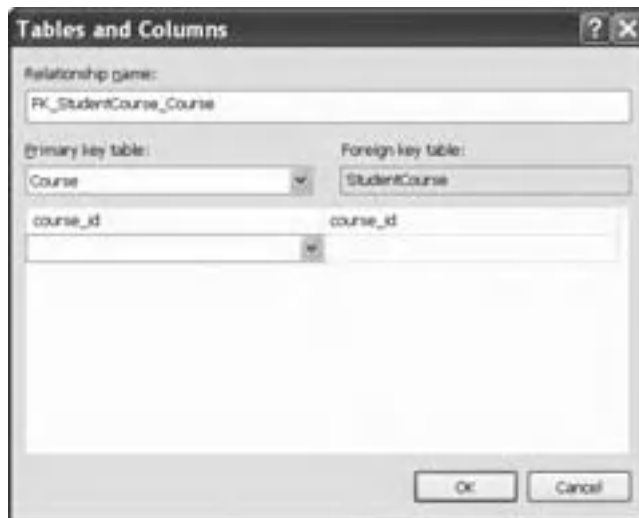


Figure 2.39. The finished Tables and Columns dialog.

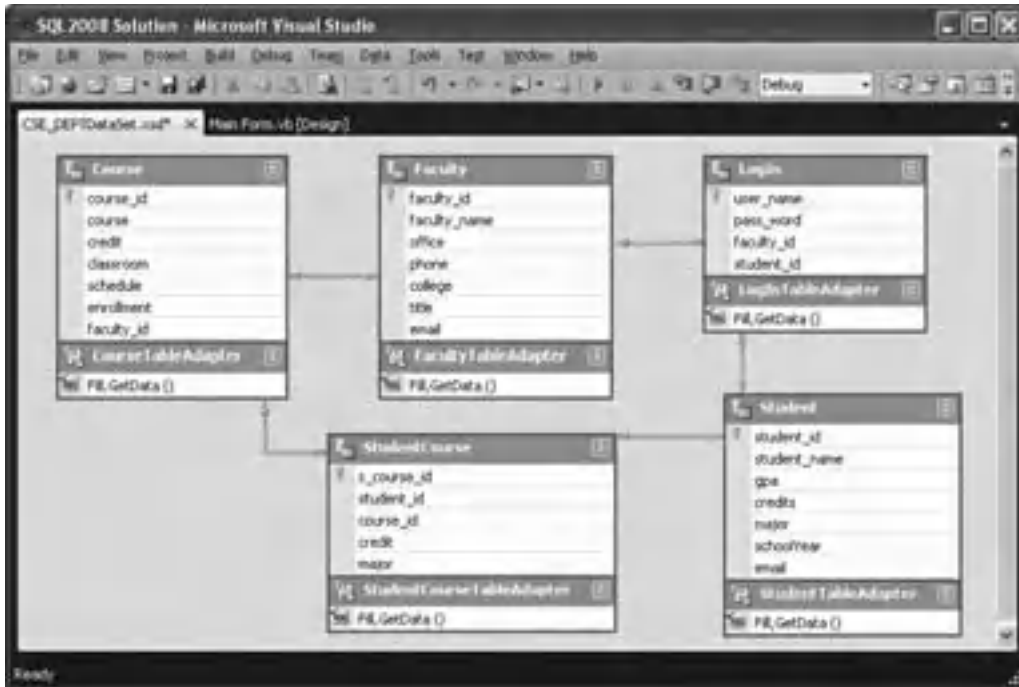


Figure 2.40. Relationships among tables.

2.11 CREATE ORACLE 11g XE SAMPLE DATABASE

After you download and install Oracle Database 11g XE (refer to Appendix B), you need to create a customer Oracle database. Creating the customer's database in Oracle Database 11g XE is different from creating a customer database in Microsoft Access or in SQL Server database management system (MDBS). In Oracle Database 11g XE, you need to create a new user or user account if you want to create a new customer database. Each user or user account is related to a schema or a database, and the name of each user is equal to the name of the associated schema or database.

Therefore, you need to perform two steps to create a customer Oracle database:

1. Create a new customer user or user account
2. Create Oracle database objects, such as tables, schemas and relations

To do that, you need to start this job from the Oracle 11g XE Home page in the server. To connect your computer to your Oracle server, go to **Start\All Programs\Oracle Database 11g Express Edition\Get Started** to open the Home page, which is shown in Figure 2.41.

It looks totally different with Oracle Database 10g XE. Yes, starting from 11g, a lot of new functions have been added into the Oracle database server and tools. Different tabs have the different purposes. Most newly added tabs are used for the Web and network database controls and operations. You can go through the entire workspace to



Figure 2.41. The opened Home page.

get a fully understanding about this new product by clicking and viewing each tab one by one.

The most important tab to us is the Application Express (APEX) tab, and we will mainly use this component to create and build our sample database later.

Oracle Database 11g XE provides two ways to enable us to create and build our customer database, which are:

1. Use the Oracle APEX
2. Use the Oracle SQL Developer

In this section, we will concentrate on using the Oracle APEX to create and build our customer Oracle database CSE_DEPT. Refer to Appendix C to get more details in how to use the Oracle SQL Developer to create and build a customer Oracle database.

To access any component in the Oracle Database 11g XE home page shown in Figure 2.41, including the Storage, Sessions, Parameters, and APEX components, you need to log in as an Administrator using the **SYSTEM** as the username and the password you used when you installed the Oracle Database 11g XE. In our applications, we used **reback** as this password. Refer to Appendix B to get more details about this password.

In Oracle Database 11g 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.

Now let's begin this customer database creation process by starting to create a new customer user or user account CSE_DEPT using the APEX.

2.11.1 Create a New Oracle Customer User or User Account

To use Oracle APEX, you must create at least one APEX workspace. For this application, you will create a workspace for the CSE_DEPT user, so that you can develop the sample application using the CSE_DEPT database account.

To create a new APEX workspace, perform the following steps:

1. Open the Oracle Database 11g XE Home page by going to Start\\All Programs\\Oracle Database 11g Express Edition\\Get Started.
2. On the opened database home page, click on the **APEX** tab.
3. On the Login page, log in as an administrator by entering the **SYSTEM** into the Username box and the password you used when you installed the Oracle Database 11g XE into the Password box. In this application, just enter **reback** into the Password box. Then click on the Login button to go to the next page.
4. On the opened Oracle APEX page, which is shown in Figure 2.42, check the **Create New** radio button to create a workspace for a new database user CSE_DEPT. Since we want to use the same user for both workspace and database, enter **CSE_DEPT** into the Database Username and the APEX Username boxes. To make the password simple, we still use **reback** as the password for this user account. Enter **reback** to the Password and Confirm Password boxes, and click on the **Create Workspace** button to continue.



Figure 2.42. The opened Oracle Application Express page.

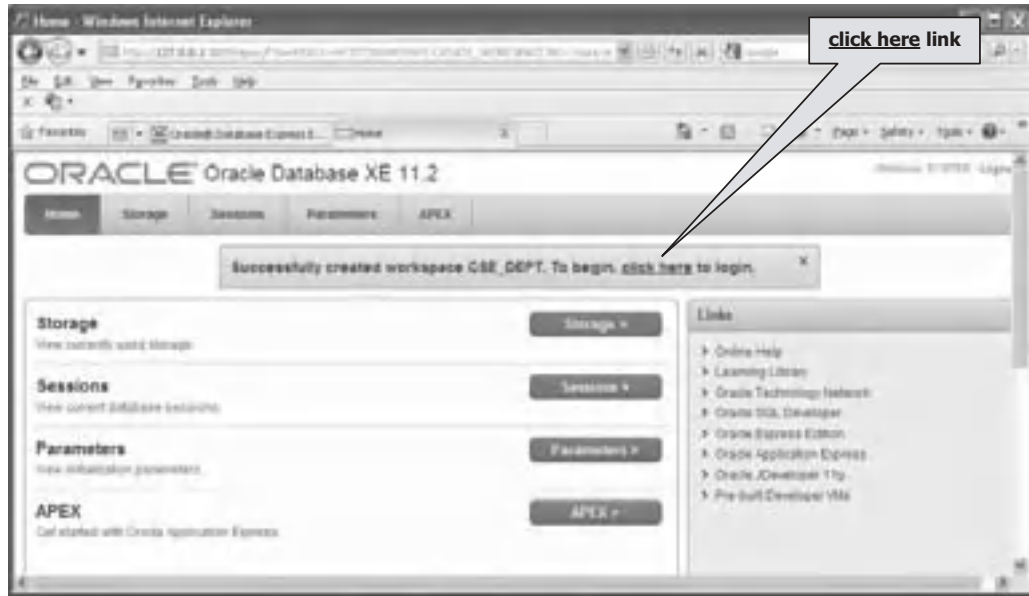


Figure 2.43. The newly created CSE_DEPT workspace.

5. On the next page shown in Figure 2.43, click on the **click here** link to log in to the newly created workspace.
6. The first time you attempt to access the workspace, you will be prompted to reset the password for the workspace. You can specify the same password or a different one. In this application, we will use the same password, **reback**, to log in to the workspace. Enter **reback** into both Current, New, and Confirm New Password boxes, and click on the Apply Changes button to complete this login process.
7. Click on the Return button to go back to the workspace login page and relog in to the workspace using the updated password.

All tools and components provided by Oracle Database 11g XE are displayed in the opened workspace, which is shown in Figure 2.44.

The functions of each component are briefly introduced below:

- **Application Builder** is the starting point you need to follow to create and implement your customer database and objects in the APEX. For any customer database or objects, you need first to create an application if you want to build and implement your customer database in the APEX environment. However, if you want to create a new customer database that is connected and used by a third-party system, as in our applications, you do not need to create any application.
- **SQL Workshop** provides five SQL-related components to enable users to create and build customer database and objects. By using this workshop, you can
 1. Create new data tables, including data columns and constraints, using the **Object Browser**.
 2. Debug and test the SQL queries using the **SQL Commands**.
 3. Create SQL statements using the **SQL Scripts**.
 4. Build the specified SQL queries using the **Query Builder**.
 5. Create database related reports and DDL using the **Utilities**.



Figure 2.44. Development tools provided by Oracle 11g XE.

- Team Development provides all tools and utilities used to support team development procedures and environment.
- Administration provides functions in creating and managing all user accounts and general database services.

In the following section, we will use the SQL Workshop, the Object Browser, to be exact, to create and build our customer database CSE_DEPT.

2.11.2 Create New Customer Sample Database CSE_DEPT

After logging in to the APEX workspace, click on the SQL Workshop component, and then click on the Object Browser component to open the Object Browser page, which is shown in Figure 2.45.

Perform the following operations to create our new sample database CSE_DEPT:

1. On the opened Object Browser page, click on the **Create** button, as shown in Figure 2.45.
2. On the next page, click on the **Table** icon to open the Create Table page, which is shown in Figure 2.46.

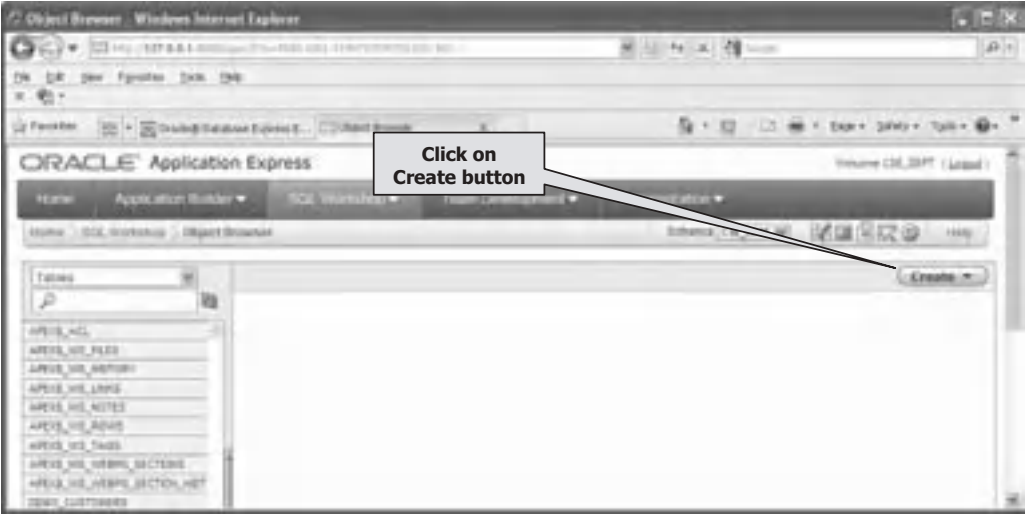


Figure 2.45. The opened Object Browser page.



Figure 2.46. The Column wizard.

3. A flowchart of developing the table is shown in the left pane of this Create Table page. 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. First, enter LogIn into the Table Name box. For our LogIn table, we have four columns: user_name, pass_word, faculty_id, and student_id. The data type for all columns is VARCHAR2, since this data type is flexible, and it can contain varying-length

characters. The upper bound of the length is 30, which is determined by the number you entered in the Scale box, and it means that each column can contain up to 30 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 2.46.

4. Click the **Next** button to go to the Primary Key page to assign the primary key for this table, which is shown in Figure 2.47.
5. Since we have defined the `user_name` column as the primary key for the `LogIn` table, therefore, check the **Not Populated** radio button and select the `USER_NAME` column from the **Primary Key** combo box. Since we do not have any Composite Primary Key for this table, just keep this box unchanged. Your finished Primary Key page should match the one that is shown in Figure 2.47. Click on the **Next** button to go to the Foreign Key page to assign foreign keys for this table.
6. Since we have not created any other table, therefore, 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 to go to the next page. The next page allows you to set up some constraints on this table, which is shown in Figure 2.48.
7. No constraint is needed for this sample database at this moment, so you can click on the **Next** button to go to the last page to confirm our `LogIn` table. The opened Confirm page is shown in Figure 2.49.
8. Click on the **Create** button to confirm and create this new `LogIn` table. Your created `LogIn` table should match the one that is shown in Figure 2.50 if it is successful. The newly 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.



Figure 2.47. The opened Primary Key wizard.



Figure 2.48. The opened setup constraints wizard.



Figure 2.49. The opened Confirmation wizard.

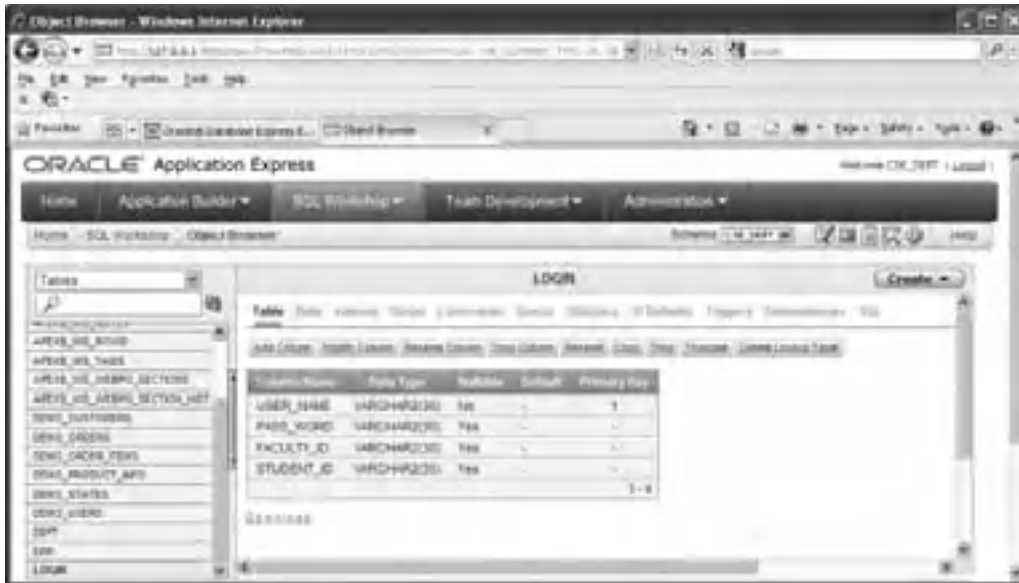


Figure 2.50. The created LogIn table.

To add data into this new LogIn table, you need to use and open the Data object tool in the top row.

2.11.3 Create the LogIn Data Table

Perform the following operations to add all columns to this newly created LogIn table:

1. Click on the Data tool to open the Data page, which is shown in Figure 2.51.
2. Click on the Insert Row button to open the datasheet view of the LogIn table, which is shown in Figure 2.52.
3. Add the following data columns into the first row:
 - A. User Name: abrown
 - B. Pass Word: america
 - C. Faculty Id: B66750
 - D. Student Id:

Since this user is a faculty, leave the Student Id column 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 2.52.

4. Click on the Create and Create Another button to create the next row. Similarly, add each row that is shown in Table 2.23 into each row on the LogIn table. For any blank column, either faculty_id or student_id, on each row shown in Table 2.23, leave that column blank and do not place a NULL for that column since it is different for a blank column in the Microsoft Access and Oracle database system.

You can click on the Create button after you add the final row into your table. Your finished LogIn table should match the one that is shown in Figure 2.53.

Figure 2.52. The opened datasheet view of the LogIn table.

Table 2.23. The 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	



A significant difference for the blank column in the Microsoft Access database and the Oracle database system is that a NULL should be placed for those blank columns in the Microsoft Access database. However, you cannot express a blank column in a same way as that in Microsoft Access database system. Instead, in Oracle database, you just leave a blank column as it without entering any staff for that column. Otherwise, you would meet trouble when you create foreign keys for those blank columns in Oracle database system.

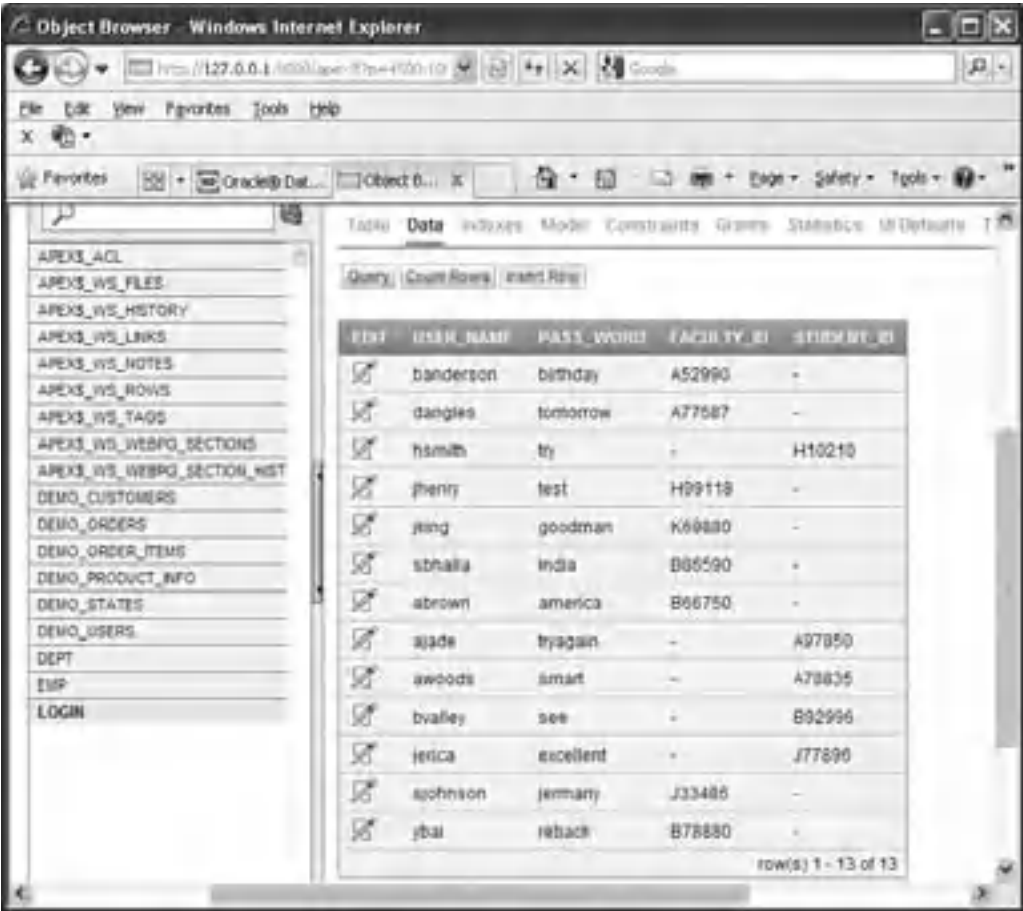
Next let's create our second table—Faculty table.

2.11.4 Create the Faculty Data 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 table:

- faculty_id: VARCHAR2(10)
- faculty_name: VARCHAR2 (30)
- office: VARCHAR2 (30)
- phone: CHAR(30)
- 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 length of the



The screenshot shows a web browser window titled "Object Browser - Windows Internet Explorer". The address bar shows a URL starting with "http://127.0.0.1:8080/apex-07p4403010". The browser displays a table named "LogIn" with the following columns: ID, USER_NAME, PASS_WORD, FACULTY_ID, and STUDENT_ID. The table contains 13 rows of data. The left sidebar shows a tree view of database objects, including APEX_*, DEMO_*, and EMP tables. The top navigation bar includes tabs for Table, Data, Indexes, Model, Constraints, Grants, Statistics, and Defaults. The "Data" tab is selected, and the "Query" button is active.

ID	USER_NAME	PASS_WORD	FACULTY_ID	STUDENT_ID
<input checked="" type="checkbox"/>	banderson	birthday	A52990	-
<input checked="" type="checkbox"/>	dangles	tomorrow	A77587	-
<input checked="" type="checkbox"/>	hsmith	ty	-	H10210
<input checked="" type="checkbox"/>	pherry	test	H99118	-
<input checked="" type="checkbox"/>	jring	goodman	K69880	-
<input checked="" type="checkbox"/>	sthalia	india	B66590	-
<input checked="" type="checkbox"/>	abrown	america	B66750	-
<input checked="" type="checkbox"/>	ajade	tryagain	-	A97850
<input checked="" type="checkbox"/>	awoods	smart	-	A79835
<input checked="" type="checkbox"/>	bvalley	see	-	B92996
<input checked="" type="checkbox"/>	jerica	excellent	-	J77890
<input checked="" type="checkbox"/>	scohnson	jermany	J33486	-
<input checked="" type="checkbox"/>	jbal	reback	B78880	-

row(s) 1 - 13 of 13

Figure 2.53. The completed LogIn table.

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 30 letters, since our phone number is composed of 10 digits, and we can extend this length to 30 with two dashes. For all other columns, the length varies with the different input information, so the VARCHAR2 is selected for those columns.

The finished Columns page of your Faculty table is shown in Figure 2.54. You need to check the Not Null checkbox for the faculty_id column since we have 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 2.55.

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 number as our primary key. Then select the FACULTY_ID(VARCHAR2) from the Primary Key combo box to select this column as the primary key for this table. Keep the Composite Primary



Figure 2.54. The finished design view of the faculty table.



Figure 2.55. The opened primary key wizard.

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 Foreign Keys page.

Since we have not finished creating all five tables to use any of them as our reference tables with the foreign key, just click on the Next button at this moment to continue and we will do the foreign key for this table later. Click on the Next button for the Constraints

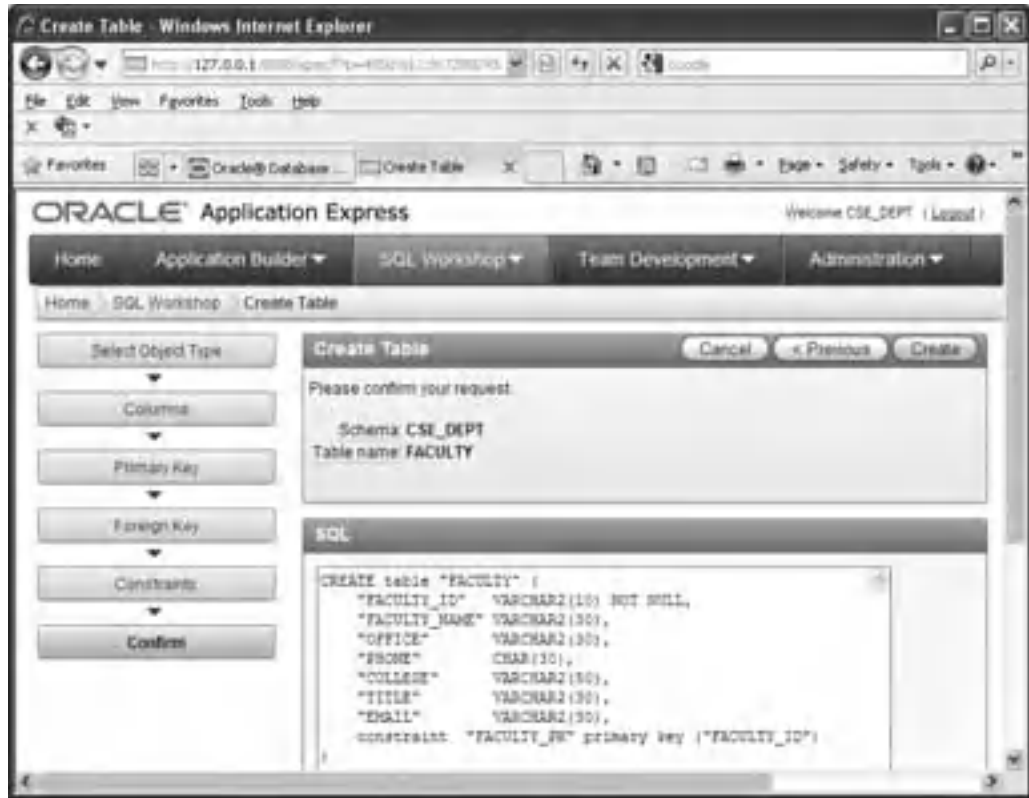


Figure 2.56. The completed columns in the Faculty table.

page since we will do that later when all five tables are created. Your opened Confirm page is shown in Figure 2.56.

Click on the **Create** button to create this Faculty table. Your finished design view of the Faculty table is shown in Figure 2.57.

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 that is shown in Figure 2.58.

Next, let's create the rest of three tables, **Course**, **Student**, and **StudentCourse**.

2.11.5 Create Other Tables

Similarly, you can continue to create the following three tables: **Course**, **Student**, and **StudentCourse** based on the data shown in Tables 2.25–2.27.

The data types used in the **Course** table are:

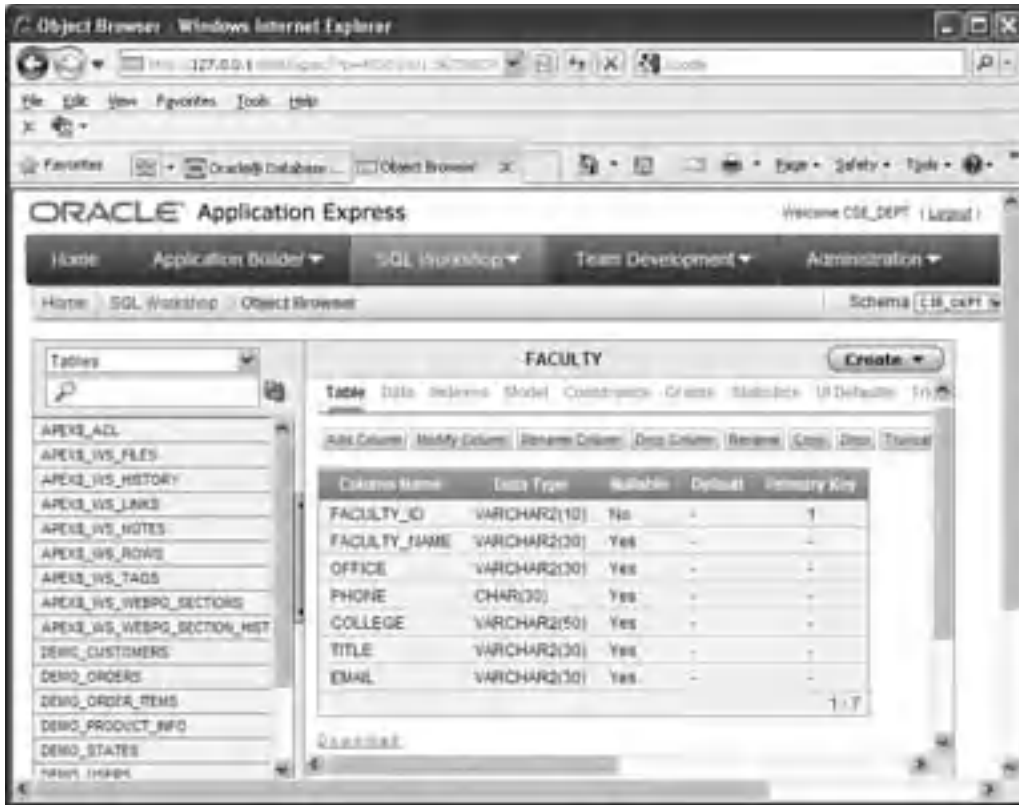


Figure 2.57. The completed columns in the Faculty table.

Table 2.24. The 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

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

The screenshot shows a web browser window titled 'Object Browser - Windows Internet Explorer'. The address bar shows a URL starting with 'http://127.0.0.1'. The browser displays a table titled 'FACULTY'. Above the table, there is a message box that says 'Row created'. Below the table, there are tabs for 'Data', 'Structure', 'Model', 'Constraints', 'Security', 'Statistics', 'SQL Scripts', 'Triggers', 'Stored Procedures', and 'SQL'. The 'Data' tab is selected. The table has the following columns: ID#, FACULTY_ID, FACULTY_NAME, OFFICE, PHONE, COLLEGE, TITLE, and EMAIL. The table contains 8 rows of data, each with a checkbox in the first column. The last row of the table is highlighted, and the status bar at the bottom indicates 'row(s) 1 - 8 of 8'.

ID#	FACULTY_ID	FACULTY_NAME	OFFICE	PHONE	COLLEGE	TITLE	EMAIL
<input checked="" type="checkbox"/>	H99118	Jeff Henry	MTG-326	750-330-6650	Ohio State University	Associate Professor	jhenry@college.edu
<input checked="" type="checkbox"/>	J33485	Steve Johnson	MTG-118	750-330-1118	Harvard University	Distinguished Professor	sjohnson@college.edu
<input checked="" type="checkbox"/>	A80980	James King	MTG-324	750-378-1230	East Florida University	Professor	jking@college.edu
<input checked="" type="checkbox"/>	A52885	Brace Anderson	MTG-218	750-378-9997	Virginia Tech	Professor	banderson@college.edu
<input checked="" type="checkbox"/>	A77587	Daisy Angles	MTG-329	750-378-2278	University of Chicago	Associate Professor	dangles@college.edu
<input checked="" type="checkbox"/>	B86750	Alice Brown	MTG-357	750-330-8050	University of Florida	Assistant Professor	abrown@college.edu
<input checked="" type="checkbox"/>	B78880	King Bai	MTG-211	750-378-1140	Florida Atlantic University	Associate Professor	ibai@college.edu
<input checked="" type="checkbox"/>	B91080	Sarah Brada	MTG-254	750-378-1001	University of Notre Dame	Associate Professor	sbrada@college.edu

Figure 2.58. The finished Faculty table.

The data types used in the **Student** table are:

- student_id: VARCHAR2(10)—Primary Key
- student_name: VARCHAR2(30)
- 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(30)

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

Table 2.25. The 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. The 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. The 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

Your finished Course, Student, and StudentCourse tables are shown in Figure 2.59–2.61, respectively.

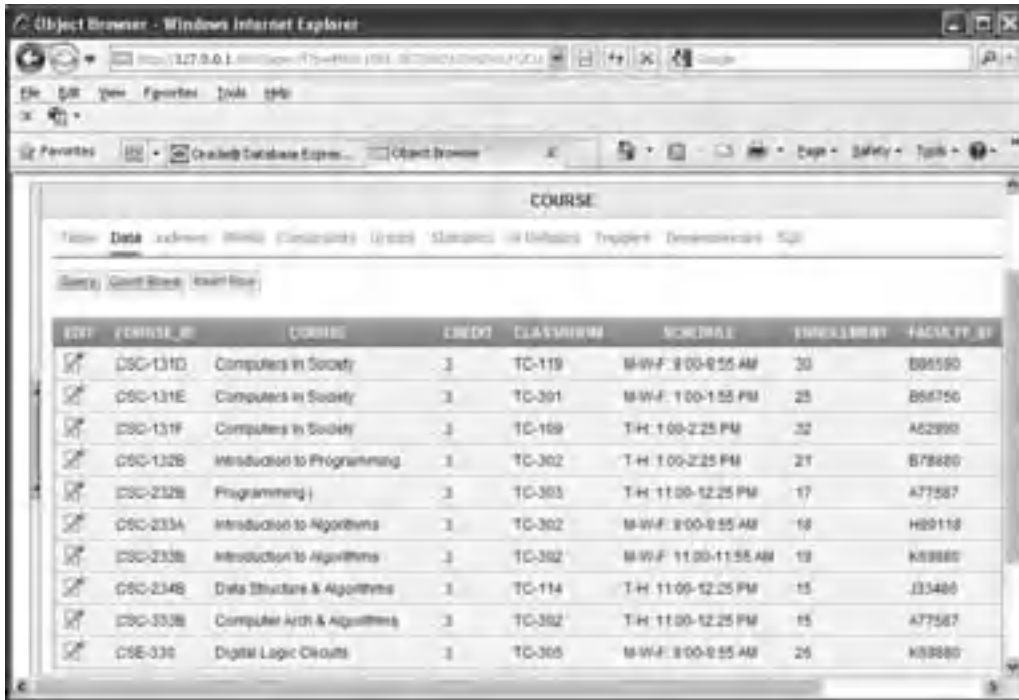
Next, let's create the constraints between these five tables to get relationships among these tables.

2.11.6 Create the 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.

2.11.6.1 Create the Constraints between the LogIn and Faculty Tables

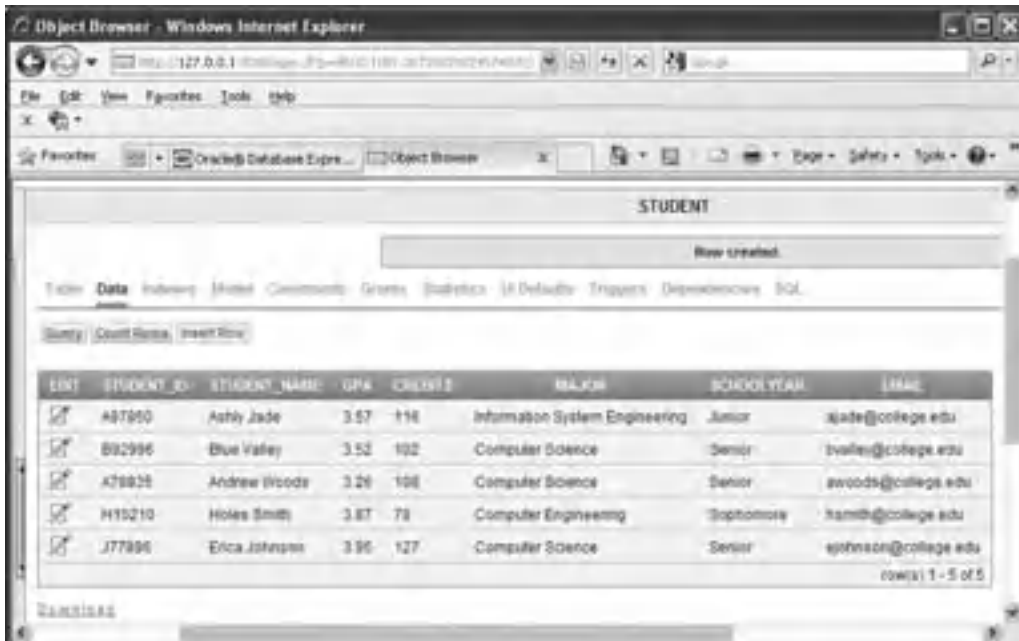
Now let's create the constraints between the LogIn and the Faculty tables by using a foreign key. Exactly, create a foreign key for the LogIn table and connect it to the primary



The screenshot shows the Oracle Object Browser interface in a Windows Internet Explorer window. The browser is displaying the 'COURSE' table. The table has columns: ENID, COURSE_ID, COURSE, CREDIT, CLASSROOM, SCHEDULE, ENROLLMENT, and FACULTY_ID. The table contains 10 rows of data.

ENID	COURSE_ID	COURSE	CREDIT	CLASSROOM	SCHEDULE	ENROLLMENT	FACULTY_ID
✓	CSC-131D	Computers in Society	3	TC-119	M-W-F 8:00-9:55 AM	30	B05190
✓	CSC-131E	Computers in Society	3	TC-301	M-W-F 1:00-1:55 PM	25	B05150
✓	CSC-131F	Computers in Society	3	TC-109	T-H 1:00-2:25 PM	32	A02990
✓	CSC-132B	Introduction to Programming	3	TC-302	T-H 1:00-2:25 PM	21	B79880
✓	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 8:00-9:55 AM	18	H09118
✓	CSC-233B	Introduction to Algorithms	3	TC-302	M-W-F 11:00-11:55 AM	19	K09880
✓	CSC-234B	Data Structure & Algorithms	3	TC-114	T-H 11:00-12:25 PM	15	J33480
✓	CSC-333B	Computer Arch & Algorithms	3	TC-302	T-H 11:00-12:25 PM	15	A77587
✓	CSE-330	Digital Logic Circuits	3	TC-305	M-W-F 8:00-9:55 AM	26	K09880

Figure 2.59. The completed Course table.



The screenshot shows the Oracle Object Browser interface in a Windows Internet Explorer window. The browser is displaying the 'STUDENT' table. The table has columns: ENID, STUDENT_ID, STUDENT_NAME, GPA, CREDIT, MAJOR, SCHOOL YEAR, and EMAIL. The table contains 5 rows of data.

ENID	STUDENT_ID	STUDENT_NAME	GPA	CREDIT	MAJOR	SCHOOL YEAR	EMAIL
✓	A07950	Ashly Jade	3.57	114	Information System Engineering	Junior	ajade@college.edu
✓	B02996	Blue Valley	3.52	102	Computer Science	Senior	bvalley@college.edu
✓	A79828	Andrew Woods	3.26	108	Computer Science	Senior	awoods@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 2.60. The completed Student table.

ENR	S_COURSE_ID	STUDENT_ID	COURSE_ID	CREDIT	GRADE
	1001	B92996	CSC-132A	3	C888
	1002	J77896	CSC-338	3	C888
	1006	B92996	CSC-233A	3	C888
	1007	A78835	CSC-132A	3	CE
	1008	A78835	CSE-432	3	CE
	1010	J77896	CSC-439	3	C888
	1019	A97850	CSC-335	3	ISE
	1021	B92996	CSC-230	3	C888
	1025	H10210	CSE-433	3	CE
	1000	H10210	CSC-131D	3	CE
	1003	A78835	CSC-331	3	CE
	1004	H10210	CSC-234B	3	CE
	1005	J77896	CSC-234A	3	C888
	1009	A78835	CSE-434	3	CE

Figure 2.61. The completed StudentCourse table.

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.

Perform the following operations to set up this one-to-many relationship between the Faculty and the LogIn tables:

1. Log in to the Oracle Database 11g XE APEX using the **SYSTEM** as the Username and the administration password as the password.
2. Log in to the Workspace using the customer user name, **CSE_DEPT** and the customer database password.
3. Click on the SQL Workshop and then the Object Browser icon to list all tables.
4. Select the LogIn table from the left pane to open it.
5. Click on the Constraints tab and then the Create button that is the first button in the second row.

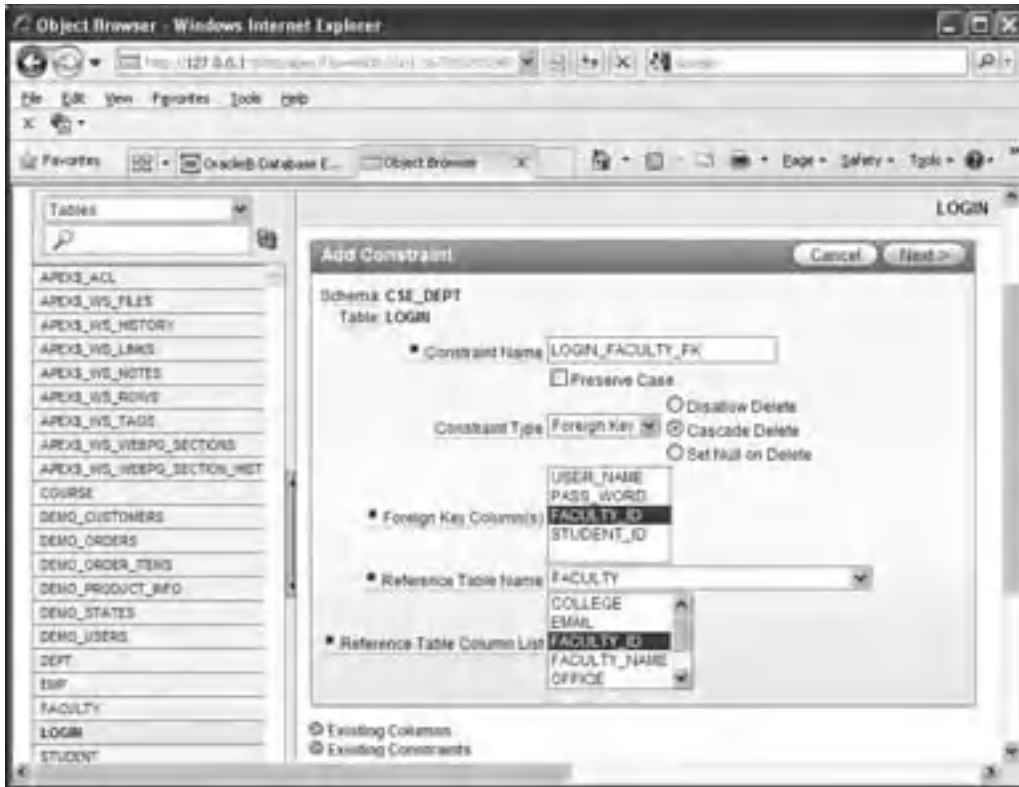


Figure 2.62. Create the foreign key between the LogIn and the Faculty table.

6. Enter LOGIN_FACULTY_FK into the Constraint Name box, and select the Foreign Key from the Constraint Type box, which is shown in Figure 2.62.
7. Check the Cascade Delete checkbox. Then select the FACULTY_ID from the LogIn 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 wizard should match the one that is shown in Figure 2.62.
8. Click on the Next button to go to the next wizard, and then click on the Finish button to confirm this foreign key's creation.

Next, let's continue to create the constraint relationship between the LogIn and the Student table.

2.11.6.2 Create the Constraints between the LogIn and Student Tables

The relationship between the Student 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_id can exist in the LogIn table, but only one or unique student_id can be found from the Student table.

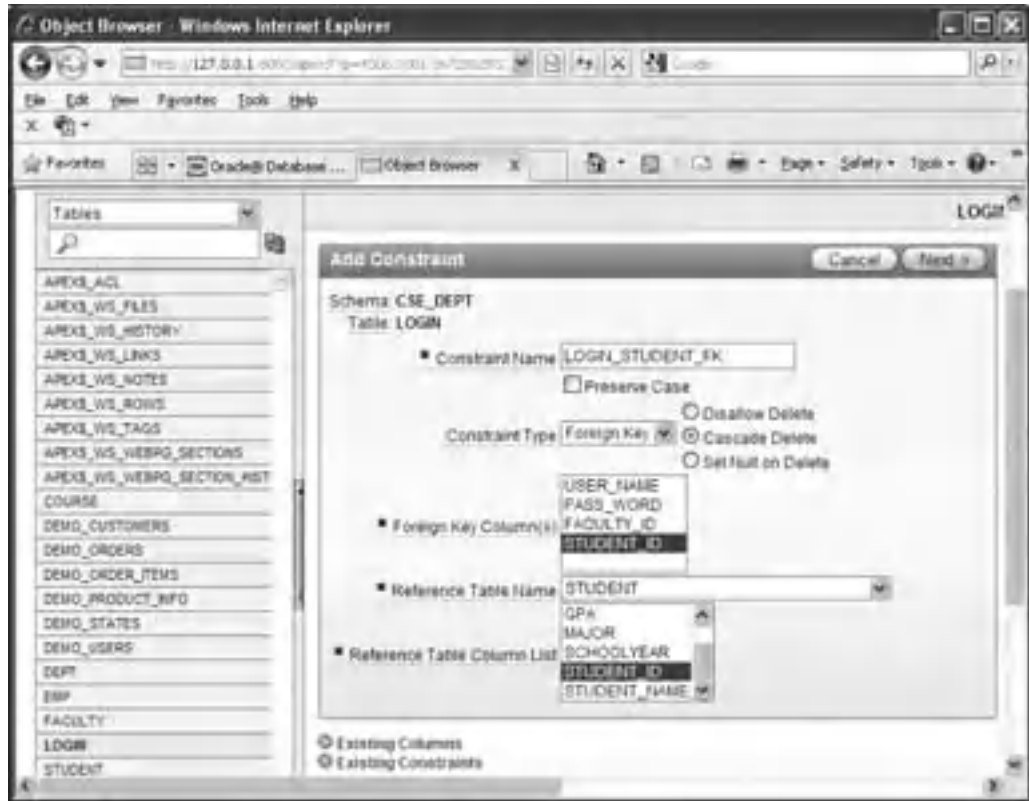


Figure 2.63. Create the foreign key between the LogIn and the Student table.

To create a foreign key from the LogIn table and connect it to the primary key in the Student table, perform the following operations:

1. 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 wizard.
2. Enter LOGIN_STUDENT_FK into the Constraint Name box, and select the Foreign Key from the Constraint Type box, which is shown in Figure 2.63.
3. Check the Cascade Delete 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 wizard should match the one that is shown in Figure 2.63.
4. Click on the Next button to go to the next wizard, and then the Finish button to confirm this foreign key's creation. Your finished foreign key creation wizard for the LogIn table should match the one that is shown in Figure 2.64.

Recall that when we created the LogIn table in Section 2.11.3, we emphasized that for the blank fields in both `faculty_id` and `student_id` columns, you should not place a NULL into these fields and just leave those fields blank. The reason for this is that an

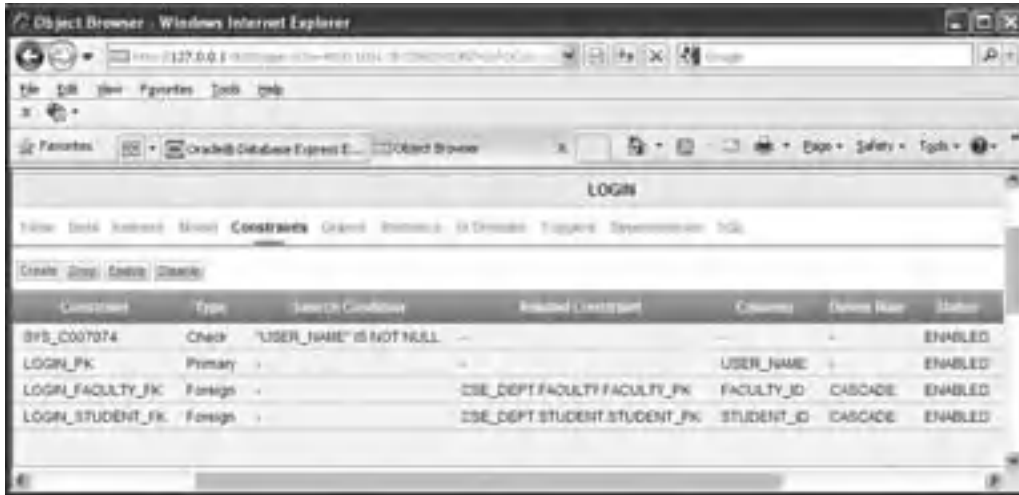


Figure 2.64. The finished foreign key creation wizard for the LogIn table.

ALTER TABLE command will be issued when you create a foreign key for the LogIn table, since the NULL cannot be recognized by this command, therefore an error ORA-02298 occurs and your creation of a foreign key will fail.

2.11.6.3 Create the Constraints between the 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 unique `faculty_id` exist in the Faculty table, but multiple `faculty_id` can exist in the Course table since one faculty can teach multiple courses.

Open the Course table by clicking on it from the left pane. Click on the Constraints tab and then 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 2.65. Check on the Cascade Delete 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 wizard should match the one that is shown in Figure 2.65.

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

2.11.6.4 Create the Constraints between the 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 StudentCourse table since one student can take multiple different courses. In order to

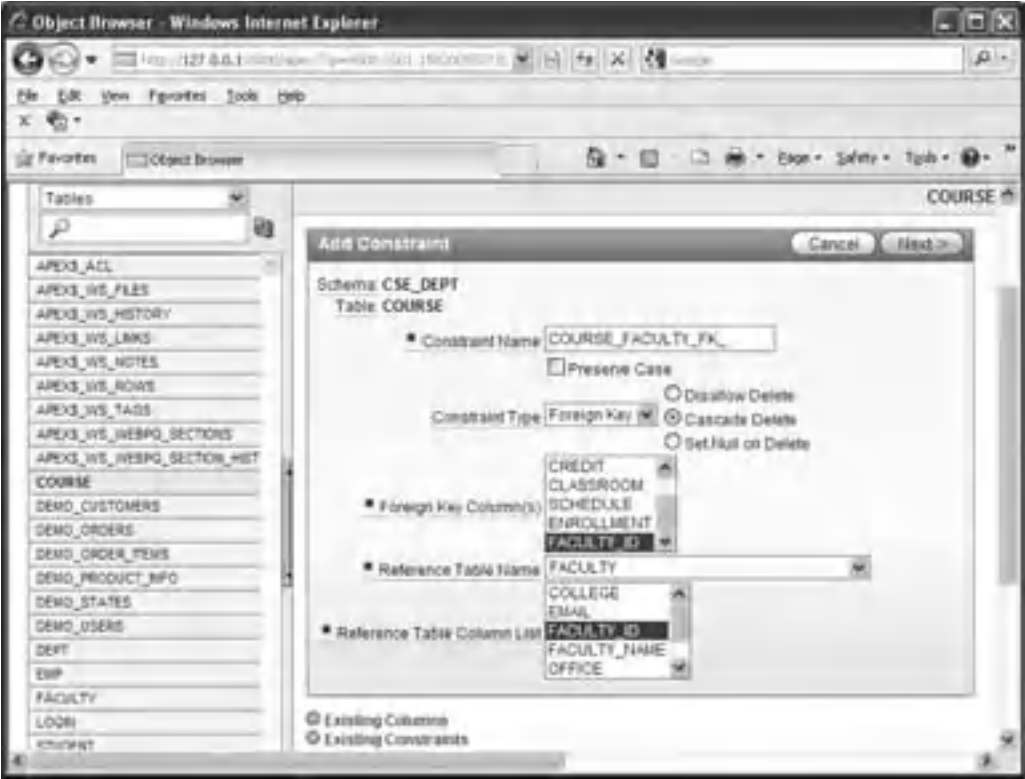


Figure 2.65. Create the foreign key between the Course and the Faculty table.



Figure 2.66. The finished foreign key creation wizard for the Course table.

create this relationship by using the foreign key, first let's open the **StudentCourse** table by clicking on it from the left pane.

Click on the **Constraints** tab and then 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

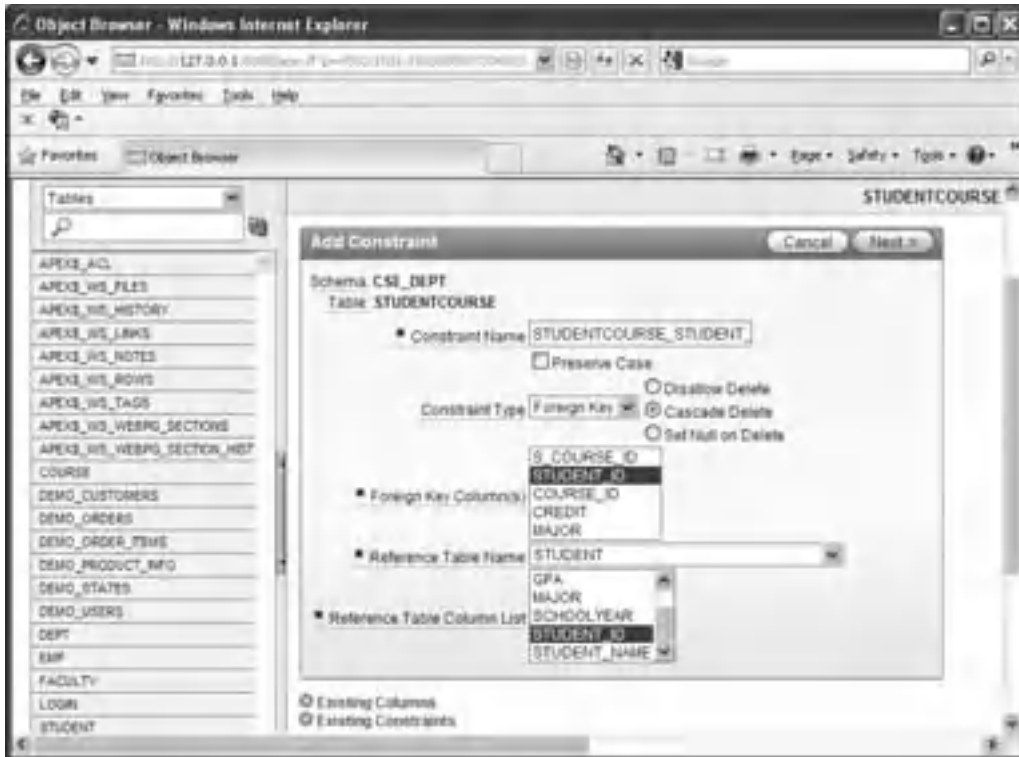


Figure 2.67. Create the foreign key between the StudentCourse and the Student table.

Figure 2.67. Check the **Cascade Delete** 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 wizard should match the one that is shown in Figure 2.67.

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

Finally, let's handle and create the constraint relationship between the StudentCourse and the Course tables.

2.11.6.5 Create the Constraints between the 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 and the Course tables.

To create this relationship by using the foreign key, open the StudentCourse table by clicking on it from the left pane. Click on the **Constraints** tab and then the **Create** button



Figure 2.68. Create the foreign key between the StudentCourse and the Course table.

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 2.68. Check the **Cascade Delete** 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 reference table column. Your finished Add Constraint wizard should match the one that is shown in Figure 2.68.

Click on the **Next** button to go to the next wizard, and then click on the **Finish** button to confirm this foreign key's creation. Your finished foreign key creation wizard for the StudentCourse table should match the one that is shown in Figure 2.69.

Our customer database creation for Oracle Database 11g Express Edition is completed. A completed Oracle 11g XE sample database **CSE_DEPT** that is represented by a group of table files can be found in the folder **Database\Oracle** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

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 the different applications we will develop in this book.



Figure 2.69. The finished foreign key creation wizard for the StudentCourse table.

Since the Oracle Database 11g XE is very different with other databases, such as Microsoft Access and SQL Server 2008, it can be seen that the creation and building process for this customer database is relatively complicated. To convenience readers and make this process simple, we have developed these five tables and converted them to the associated text files. To use those five text files to create a customer database CSE_DEPT, you need to refer to Appendix D to get the knowledge in how to use the Utilities of Oracle Database 11g 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 popular database systems are provided in this chapter. Some key technologies in developing and designing database are also given and discussed in this part. The procedure and components to develop a relational database are analyzed in detail with some real data tables in our sample database CSE_DEPT. The process in developing and building a sample database is 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 are developed with three popular DBMS, such as Microsoft Access 2007, SQL Server 2008, and Oracle Database 11g XE, are provided in detail. All of these three sample databases will be used in the following chapters throughout the whole book.

HOMework

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 1NF 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. 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. It can be *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 11g 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 story 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 method. The methodology development has been focused on Microsoft-based APIs, such as Open Database Connectivity (ODBC), Object Linking And Embedding, Database (OLEDB), Microsoft Jet, Data Access Objects (DAOs), and Remote Data Objects (RDOs), in addition to many non-Microsoft-based APIs. These APIs did not bridge the gap between object-based and semi-structured (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 you should have a tremendous opportunity for complete rearchitecture. The ADO.NET is a good solution for these challenges.

3.1 THE 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. In the recent decade, ADO has been the preferred 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 the ODBC. In the late 1990s, ADO, which is based on OLEDB, was developed. This technology is widely applied in most object-oriented programming and database applications during the last decade.

Starting from ADO.NET 2.0, Microsoft released some new versions for this product, such as ADO.NET 3.5, with Visual Studio.NET 2008, and ADO.NET 4.0, which is released

with Visual Studio.NET 2010, and it is the updated version of ADO.NET that is based mainly on the Microsoft .NET Framework 4.0.

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 its new developments, ADO.NET 3.5 is highly recommended as a preferred technology because of its powerful managed runtime environment and services.

ADO.NET 4.0 provides the following new features and components compared with the earlier versions:

- LINQ to DataSet
- LINQ to SQL
- LINQ to Entities (ADO.NET Entity Framework)
- WCF Data Services (ADO.NET Data Services)
- XML and ADO.NET

Figure 3.1 shows an overview of how the ADO.NET LINQ technologies relate to high-level programming languages and LINQ-enabled data sources.

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

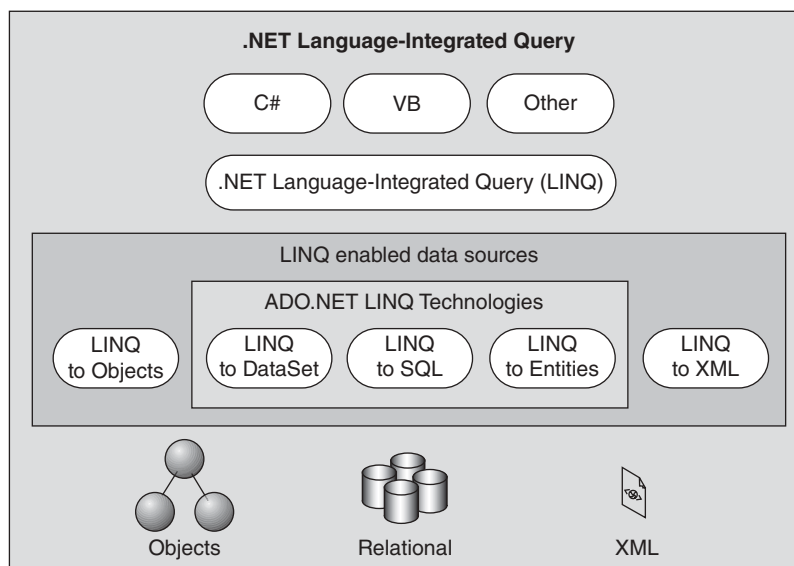


Figure 3.1. ADO.NET LINQ techniques.

In this chapter, you will:

- Learn the basic classes in ADO.NET and its architecture
- Learn the different ADO.NET 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 4.1 Entity Framework (EF)
- Learn about the ADO.NET 4.1 Entity Framework Tools (EFT)
- Learn about the ADO.NET 4.1 Entity Data Model (EDM)

First, let's have a global picture of ADO.NET and its components.

3.2 OVERVIEW OF ADO.NET

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 in developing 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 Data Provider classes are located at the different namespaces based on the types of the 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 ADO.NET, let's first take a look at the architecture of ADO.NET.

3.3 THE ARCHITECTURE OF ADO.NET

The ADO.NET 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 3.2 shows a typical architecture of ADO.NET 2.0.

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

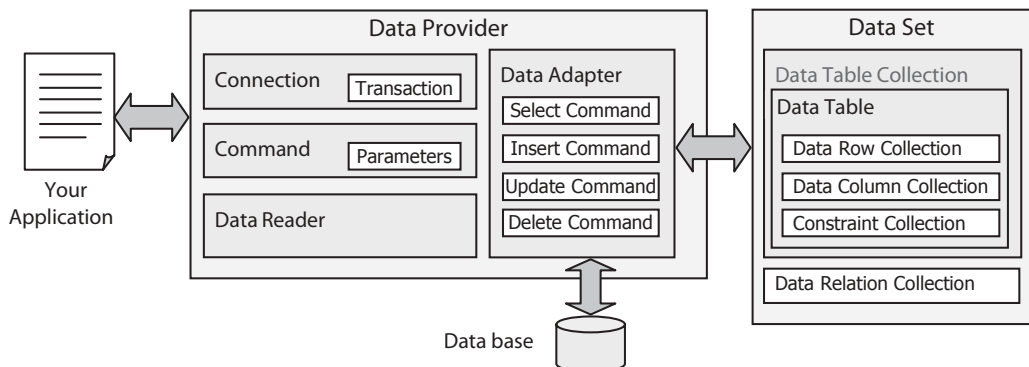


Figure 3.2. A typical architecture of ADO.NET.

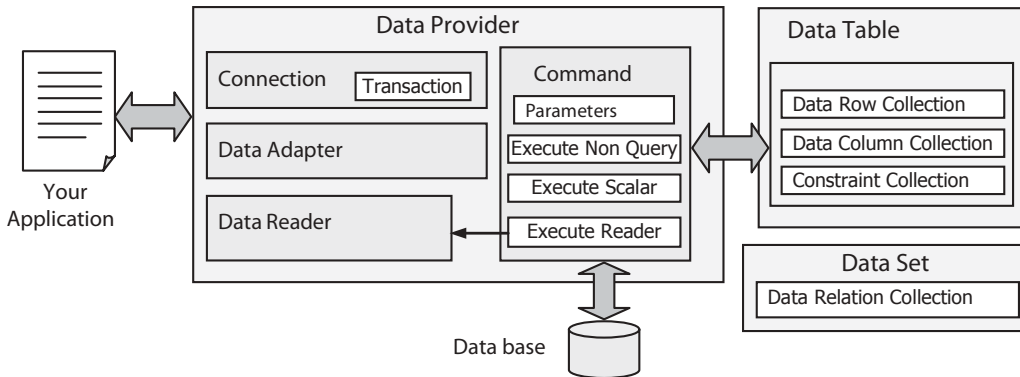


Figure 3.3. Another architecture of ADO.NET.

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 to call the `Command` object with different parameters.

Another ADO.NET 2.0 architecture is shown in Figure 3.3.

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.

Keeping these two ADO.NET architectures in mind, we will have a more detailed discussion for each component of 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 THE COMPONENTS OF ADO.NET

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

3.4.1 The 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 different kinds of databases, the 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, which is the letters enclosed by 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 ADO.NET are extended by the 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 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 lists the most popular namespaces used by different data providers and used by the `DataSet` and the `DataTable`.

Since the different Data Provider is located at the different namespace, as shown in Table 3.1, you must first import the appropriate namespace into your Visual Basic.NET 2005 project, that is, into the 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

Table 3.1. Namespaces for different Data Providers, `DataSet`, and `DataTable`

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

the associated keyword. For example, you must use “imports System.Data.OleDb” to import the namespace of the OLEDB.NET Data Provider if you want to use this Data Provider in your project, and also all classes belong 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 different namespaces and have different prefixes, the classes of these Data Providers have 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 methods or properties with the same name, such as Connection String property, Open() and Close() method, as well as the ExecuteReader() method. This provides the flexibility for the programmers and allows them to use different Data Providers to access the different data source 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 The 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 The OLEDB Data Provider

The System.Data.OleDb namespace holds all classes used by the .NET Framework 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:

Table 3.2. The compatibility between the OLEDB and OLEDB.NET

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

- 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 source 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 The SQL Server Data Provider

This Data Provider provides access to an 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 is located at the System.Data.SqlClient namespace. Although Microsoft provides different Data Providers to access the data in 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.

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 The 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 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 versions. 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 The Connection Class

As shown in Figures 3.2 and 3.3, 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 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 Basic.NET. 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 a quite 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

Table 3.3. The Connection classes and databases

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

4. User ID
5. Password

For different databases, the parameters contained in the connection string may have a 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 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 5.18.1 in Chapter 5.

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 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:

```
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 2008 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

```
Connection = New OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0;" & _
    "Data Source=C:\\database\\CSE_DEPT.accdb;" & _
    "User ID=MyUserID;" & _
    "Password=MyPassWord;")
```

SQL Server Data Provider for SQL Server Database

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

Oracle Data Provider for Oracle Database

```
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 The Open() Method of the 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 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 3.4.

```
Dim strConnectionString As String = " Provider=Microsoft.ACE.OLEDB.12.0;" & _
    "Data Source=C:\\database\\Access\\CSE_DEPT.accdb;"

accConnection = New OleDbConnection(strConnectionString)

Try
    accConnection.Open()
Catch OleDbExceptionErr As OleDbException
    MessageBox.Show(OleDbExceptionErr.Message, "Access Error")
Catch InvalidOperationExceptionErr As InvalidOperationException
    MessageBox.Show(InvalidOperationExceptionErr.Message, "Access Error")
End Try

If accConnection.State <> ConnectionState.Open Then
    MessageBox.Show("Database Connection is Failed")
    Exit Sub
End If
```

Figure 3.4. An example code of the opening a connection.

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 called after a new `OleDbConnection` object is created to open this connection. Two possible typical errors, either an `OleDbException` or an `InvalidOperationException`, could have happened after this `Open()` method is executed. A related message would be displayed if any one of those errors occurred and 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, you can confirm that the connection is successful or not.

3.4.2.2 The Close() Method of the 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 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 that data source, otherwise a possible error may be encountered when you try reopen that connection in the next time as you run your project.

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 The Dispose() Method of the Connection Class

The `Dispose()` method of the `Connection` class is an overloaded method, and it is used to releases 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 the 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.

After the `Close()` and `Dispose()` methods are executed, you can release your reference to the `Connection` instance by setting it to `Nothing`. A part of the sample code is shown in Figure 3.5.

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.

```
' clean up the objects used  
accConnection.Close()  
accConnection.Dispose()  
accConnection = Nothing
```

Figure 3.5. An example code for the cleanup of resources.

3.4.3 The Command and the 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 the 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 the 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 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 3.2 and 3.3 in this chapter. In Figure 3.2, 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 method of the TableAdapter, such as SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand, and is executed 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 3.3, 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 of the most popular properties of the Command class are discussed below.

3.4.3.1 The Properties of the Command Class

The Command class contains more than 10 properties, but only four 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 that is stored in the CommandText property should be executed. In other words, the CommandType property specifies how the CommandText property can be interpreted. In total, 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 the 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 The Constructors and Properties of the Parameter Class

The Parameter class has four popular constructors, which are shown in Figure 3.6 (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 instantiate a new 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

```
Dim sqlParameter As New SqlParameter()
Dim sqlParameter As New SqlParameter(ParamName, objValue)
Dim sqlParameter As New SqlParameter(ParamName, sqlDbType)
Dim sqlParameter As New SqlParameter(ParamName, sqlDbType, intSize)
```

Figure 3.6. Four constructors of the Parameter class.

Table 3.4. The data types and the associated Data Provider

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

Table 3.5. The different parameter mappings

Parameter Mapping	Associated Data Provider
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

ADO.NET Data Provider have different definitions for the data types they provided. The 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 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 Provider.

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 statement 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 a 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. The 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 id, 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 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 that appears 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 id, 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 operator is replaced by the SQL comparator `LIKE` for two parameters. This changing 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 3.7.

Where two `ParameterName` properties are assigned with two dynamic parameters, `@Param1` and `@Param2`, respectively. Both `Param1` and `Param2` are nominal names

```
Dim paramUserName As New SqlParameter
Dim paramPassWord As New SqlParameter

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

Figure 3.7. An example of initializing the property of a `Parameter` object.

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 codes that 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 The Methods of the 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.



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.

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 added 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 added 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, the **Add()** method and the **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( value As SqlParameter ) As SqlParameter
ParameterCollection.Add( parameterName As String, Value As Object )
```

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.

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( parameterName As String, Value As Object )
```

An example of using these two methods to add Parameter objects into a Parameters collection is shown in Figure 3.8.

The top section is used to create and initialize the Parameter objects, which we have discussed in the 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 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 The Constructor of the Command Class

The constructor of the Command class is an overloaded method, and it has multiple protocols. Four popular protocols are listed in Figure 3.9 (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 instantiate a new Command object.

```
Dim paramUserName As New SqlParameter
Dim paramPassWord As 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 3.8. Two methods to add Parameter objects.

```
Dim sqlCommand As New SqlCommand()
Dim sqlCommand As New SqlCommand(connString)
Dim sqlCommand As New SqlCommand(connString, SqlConnection)
Dim sqlCommand As New SqlCommand(connString, SqlConnection, SqlTransaction)
```

Figure 3.9. Three popular protocols of the constructor of the Command class.

```

Dim cmdString1 As String = "SELECT id, user_name, pass_word FROM LogIn "
Dim cmdString2 As String = "WHERE (user_name LIKE @Param1 ) AND (pass_word LIKE @Param2)"
Dim cmdString As String = cmdString1 & cmdString2
Dim paramUserName As New SqlParameter
Dim paramPassWord As New SqlParameter
Dim 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 3.10. An example of creating a SqlCommand object.

The second constructor contains two arguments: the first one is the parameter name that is a string variable, and the second is the value that 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 3.10. 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 a SqlCommand objects are created in the following lines.

Then two SqlParameter objects are initialized with nominal parameters and the associated text box'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 The Methods of the 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 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.

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.

```
Dim cmdString As String = "SELECT id, user_name, pass_word FROM LogIn "
Dim sqlCommand As New SqlCommand

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

Figure 3.11. An example code of running of ExecuteReader method.

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 The ExecuteReader Method The ExecuteReader() method is a data query method, and it can only be used to execute a read-out 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 object 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 following example coding can be used to execute an SQL SELECT statement, which is shown in Figure 3.11.

As shown in Figure 3.11, 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.

```

Dim cmdString As String = "SELECT pass_word FROM LogIn WHERE (user_name = ybai)"
Dim sqlCommand As New SqlCommand
Dim passWord As String

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

```

Figure 3.12. A sample code of using the ExecuteScalar method.

```

Dim cmdString1 As String = "INSERT INTO LogIn (pass_word) VALUES ('reback')"
Dim cmdString2 As String = "DELETE FROM LogIn WHERE (user_name = ybai)"
Dim sqlCommand As New SqlCommand

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

```

Figure 3.13. An example code of using the ExecuteNonQuery method.

3.4.3.6.2 The 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 3.12.

In this sample, the SQL SELECT statement is 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 in Chapter 5 provides an example of using this method to pick up a single value, which is a password, from the LogIn data table from the CSE_DEPT database.

3.4.3.6.3 The ExecuteNonQuery Method As we mentioned, the ExecuteReader method is a read-out 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 3.13 shows a sample of coding using this method to insert to and delete a record from the LogIn data table.

As shown in Figure 3.13, the first SQL statement is to try 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. Similar procedure is performed for the DELETE statement.

Now let's look at the last class in the Data Provider, DataReader.

3.4.4 The 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 assign them to the appropriate DataAdapter properties, such as SelectCommand, and execute 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 a 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 The Constructor of the 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 the most data-driven applications.

3.4.4.2 The Properties of the DataAdapter Class

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

Table 3.7. The 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. The 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. The public methods of the DataAdapter class

Methods	Descriptions
Dispose	Releases the resources used by the DataAdapter.
Fill	Add 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 The Methods of the 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, Dispose, Fill, FillSchema, and Update are the 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 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 The Events of the DataAdapter Class

Two events are available to the DataAdapter class, and these events are listed in Table 3.10.

Table 3.10. The events of the DataAdapter class

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

```
A Dim cmdString As String = "SELECT name, office, title, college FROM Faculty"
Dim sqlCommand As New SqlCommand
Dim sqlDataAdapter As SqlDataAdapter
Dim dataSet As DataSet

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

C     sqlDataAdapter = New SqlDataAdapter(cmdString, sqlConnection)
D     sqlDataAdapter.SelectCommand = sqlCommand
        dataSet = New DataSet()
        dataSet.Clear()
E     Dim intValue As Integer = sqlDataAdapter.Fill(dataSet)
        If intValue = 0 Then
            MessageBox.Show("No valid faculty found!")
        End If

F     dataSet.Dispose()
        sqlDataAdapter.Dispose()
        sqlCommand.Dispose()
        sqlCommand = Nothing
```

Figure 3.14. An example of using the SqlDataAdapter to fill the DataSet.

Before we can complete this section, a coding example 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 3.14 shows an example of using a 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 the SqlCommand class, a new object of the 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 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 codes are released by using the `Dispose` method.

3.4.5 The 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:

```
Dim sqlDataReader As 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 the 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.

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 the 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`.

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 .

Figure 3.15 shows a sample code of the usage of SqlDataReader object to continuously retrieve all records (rows) from the Faculty data table suppose a Connection object sqlConnection has been created.

The functionality of this piece of codes is explained below.

Starting from section **A**, a new SqlCommand and a 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

```

A Dim cmdString As String = "SELECT name, office, title, college FROM Faculty"
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader

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

C     sqlDataReader = sqlCommand.ExecuteReader
    If sqlDataReader.HasRows = True Then
        While FacultyReader.Read()
            For intIndex As Integer = 0 To FacultyReader.FieldCount - 1
                FacultyLabel(intIndex).Text = FacultyReader.Item(intIndex).ToString
            Next intIndex
        End While
    Else
D         MessageBox.Show("No matched faculty found!")
    End If

E     sqlDataReader.Close()
    sqlDataReader = Nothing
    sqlCommand.Dispose()
    sqlCommand = Nothing

```

Figure 3.15. An example code of using the SqlDataReader object.

the HasRows 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 For...Next loop is utilized to sequentially read out all rows one by one using the Read method (refer to Table 3.12). The Item(Int32) property (refer to Table 3.11) and the ToString() method (refer to Table 3.12) are 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...Next 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 staff, 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 unnecessary debug processes as your project runs.

3.4.6 The DataSet Component

The DataSet, which is an in-memory cache of data retrieved from a database, is a major component of 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:

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 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.

- 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, or any OLEDB- 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 Department 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 importantly, 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.

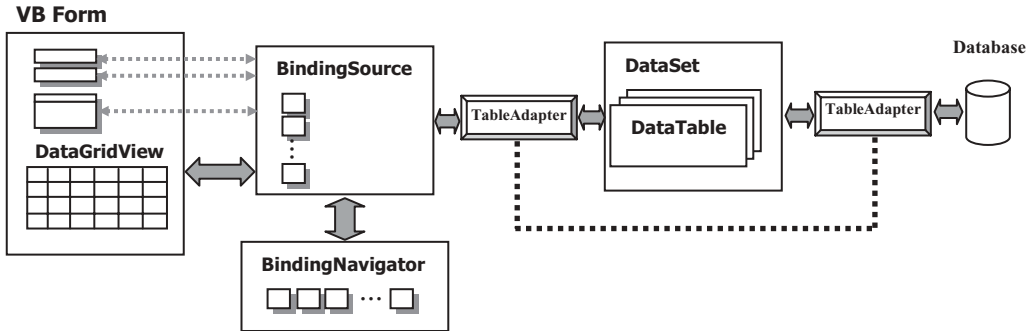


Figure 3.16. A global representation of the DataSet and other data objects.

With the Visual Basic 2010, 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 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 3.16 shows a global relationship between the DataSet object, other data objects, and the Visual Basic 2010 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, you can select to use either kind of DataSet as you like. But the typed DataSet has more support in Visual Studio 2010.

A typed DataSet object provides you with 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 The DataSet Constructor

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

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	DataManager	Gets a custom view of the data contained in the DataSet to allow filtering, searching, and navigating using a custom DataManager
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 .

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.

3.4.6.2 The 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 The 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, it had better 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

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 run time.
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.

finished your data query or data action using the DataSet, you need to release it by executing the Dispose method.

3.4.6.4 The DataSet Events

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

The Disposed event is used to trigger the Dispose event procedure as this event occurs. The Initialized event is used to make a mark to indicate that the DataSet has been initialized 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 you how to create, initialize, and implement a real DataSet object in a data-driven application. A piece of codes shown in Figure 3.17 is used to illustrate these issues, and a SQL Server Data Provider is utilized

```

A Dim cmdString As String = "SELECT name, office, title, college FROM Faculty"
    Dim sqlCommand As New SqlCommand
    Dim sqlDataAdapter As SqlDataAdapter
    Dim sqlDataSet As DataSet
    Dim intValue As Integer

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 Then
        MessageBox.Show("No valid faculty found!")
    End If

E sqlDataSet.Dispose()
    sqlDataAdapter.Dispose()
    sqlCommand.Dispose()
    sqlCommand = Nothing

```

Figure 3.17. An example of using the DataSet.

for this example. We assume 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.

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 the `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.

Then 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` (i.e., 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 codes 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 `Fill` is called, it is opened to retrieve data, and then closed. If the connection is open before `Fill` is called, it still remains open.

The `Fill` operation then adds the rows to destination `DataTable` objects in the `DataSet`, creating the `DataTable` objects if they do not already exist. When creating `DataTable`

objects, the Fill operation normally creates only column name 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 The DataTable Component

`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 your 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, in which each `DataTable` object is 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 3.4.4.4). 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 standalone data table object, and each table can be populated or manipulated by executing either the `ExecuteReader` or `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 are existed 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`

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.

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 `DataRow`. 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 The `DataTable` Constructor

The `DataTable` has four overloaded constructors, and Table 3.18 lists three 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 3.18 shows an example of creating a new `DataTable` and a `DataSet`, and then adding the `DataTable` into the `DataSet` object.

```
Dim FacultyDataSet As DataSet
Dim FacultyTable As DataTable

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

Figure 3.18. An example of adding a DataTable into a DataSet.

Table 3.19. The popular properties of the DataTable class

Properties	Descriptions
Columns	The data type of the Columns property is DataColumn-Collection, 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 existed 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 existed in the DataTable can be retrieved.
TableName	Gets or sets the name of the DataTable.

First, you need to create two instances of the DataSet and the DataTable, respectively. Then you can add this new DataTable instance into the new DataSet object by using the Add method.

3.4.7.2 The 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. 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.

Table 3.20. The 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	Release the resources used by the MarshalByValue-Component.
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.

3.4.7.3 The DataTable Methods

The DataTable class has about 50 different methods with 33 public methods, and Table 3.20 lists some most often used methods.

Among these methods, three of them are important to us: NewRow, ImportRow, and LoadDataRow. Calling 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 ImportRow preserves the existing DataRowState along with other values in the row. Calling LoadDataRow is to find and update a data row from the current data table. This method has two arguments, the Value (As Object) and the Accept Condition (As 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 The DataTable Events

The DataTable class contains 11 public events, and Table 3.21 lists these events.

Table 3.21. The 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
Dim FacultyTable As DataTable = New DataTable("FacultyTable")

B 'Declare DataColumn and DataRow variables
Dim column As DataColumn
Dim row As DataRow

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)

'Create new DataRow objects and add to DataTable.
Dim Index As Integer
E For Index = 1 To 10
    row = FacultyTable.NewRow
    row("FacultyId") = Index
    row("FacultyOffice") = "TC- " & Index
F FacultyTable.Rows.Add(row)
Next Index

```

Figure 3.19. An example of creating a new table and adding data into the table.

The most often used events are ColumnChanged, Initialized, RowChanged, and RowDeleted. By using these events, one can track and monitor the real situations that occurred 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 3.19 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 3.19; 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 contain the `DataColumn` and `DataRow` objects. So next, you need to create `DataColumn` and `DataRow` objects, respectively. Step **B** finished these objects' 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` is added into the `FacultyTable` using the `Add` method of the `Columns` collection class.

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 **D**.

In step **E**, a `For...Next` 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` (refer to Table 3.20). 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.

The completed `FacultyTable` should match the one that is shown in Table 3.22.

3.4.8 ADO.NET Entity Framework 4.1

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

Table 3.22. The 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

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 in our real world today. Although some new components were added into 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 4.1 based on the .NET Framework 4.1 and the addition of LINQ to Visual Studio.NET 2010. The main contributions of ADO.NET 4.1 include that some new components, ADO.NET 4.1 Entity Framework and ADO.NET 4.1 EDM Tools, are added into ADO.NET 4.1. With these new components, the contradiction that existed between the relational model of data used in databases and the object-oriented programming projects can be fully resolved.

The first version of Entity Framework was included with .NET Framework 3.5 Service Pack 1 and Visual Studio 2008 Service Pack 1, released on August 2008. The second version of Entity Framework, named Entity Framework 4.0, was released as part of .NET 4.0 on April 2010, and has addressed many of the criticisms made of version 1. A third version of Entity Framework, version 4.1, was released on April 12, 2011. There were several betas (called Community Technology Previews) of it, the last of which was released in March 2011.

One of the primary goals on ADO.NET 4.1 Entity Framework 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 4.1 abstraction or in terms of persistent objects (ORM) and is built upon the standard ADO.NET 4.1 Provider model that allows access to third-party databases. The Entity Framework introduces a new set of services around the EDM (a medium for defining domain models for an application).

ADO.NET 4.1 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 4.1 Entity Framework 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 set up 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

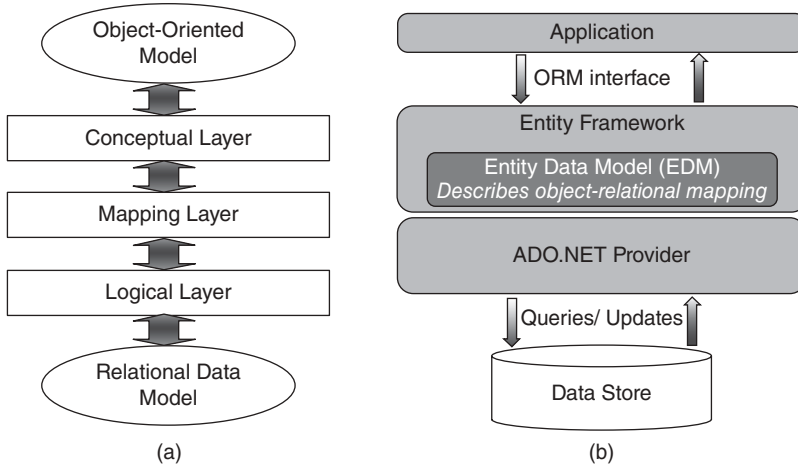


Figure 3.20. The mapping relationship between three layers.

layer. The mapping that is shown in Figure 3.20a allows users to code against the Conceptual layer and map those codes into the Logical layer.

An architecture of implementing Entity Framework 4.1 is shown in Figure 3.20b.

The Entity Framework 4.1 is built on the existing ADO.NET provider model, with existing providers being updated additively to support the new Entity Framework functionality. Because of this, existing applications built on ADO.NET can be carried forward to the Entity Framework 4.1 easily with a programming model that is familiar to ADO.NET developers.

A useful data component is provided by the Conceptual layer to enable users to develop object-oriented codes, and it is called *EntityClient*. In fact, 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. Figure 3.21 shows the Entity Framework architecture for accessing data.

As can be found from Figure 3.21, the Entity Framework includes the *EntityClient* data provider. This provider manages connections, translates entity queries into data source-specific queries, and returns a data reader that the Entity Framework uses to materialize entity data into objects. When object materialization is not required, the *EntityClient* provider can also be used like a standard ADO.NET data provider by enabling applications to execute Entity SQL queries and consume the returned read-only data reader.

3.4.8.1 Advantages of Using the Entity Framework 4.1

In summary, using the Entity Framework 4.1 to write data-oriented applications provides the following benefits:

1. Reduced development time. The framework provides the core data access capabilities so developers can concentrate on application logic.

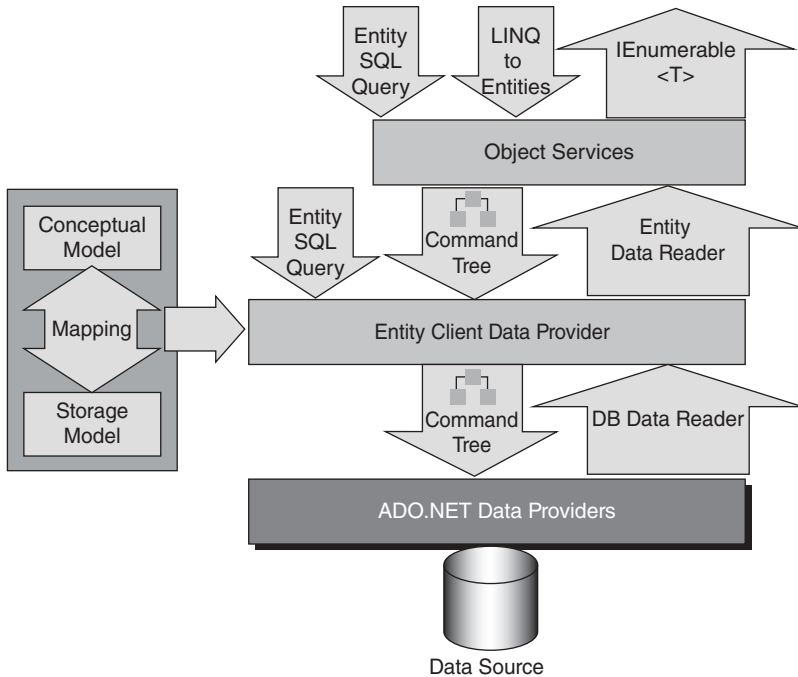


Figure 3.21. Entity Framework architecture.

2. Developers can work in terms of a more application-centric object model, including types with inheritance, complex members, and relationships. In .NET Framework 4, the Entity Framework also supports Persistence Ignorance through Plain Old CLR Objects (POCO) entities.
3. Applications are freed from hard-coded dependencies on a particular data engine or storage schema by supporting a conceptual model that is independent of the physical/storage model.
4. Mappings between the object model and the storage-specific schema can change without changing the application code.
5. LINQ support (called LINQ to Entities) provides IntelliSense and compile-time syntax validation for writing queries against a conceptual model.

The Entity Framework uses the EDM to describe the application-specific object or “conceptual” model against which the developer programs. The EDM builds on the widely known Entity Relationship model to raise the abstraction level above logical database schemas. The EDM was developed with the primary goal of becoming the common data model across a suite of developer and server technologies from Microsoft. Thus, an EDM created for use with the Entity Framework can also be leveraged with WCF Data Services (formerly ADO.NET Data Services), Windows Azure Table Storage, SharePoint 2010, SQL Server Reporting Services, and SQL Server PowerPivot for Excel, with more coming in the future.

The core of ADO.NET 4.1 Entity Framework is its EDM, and the user can access and use this model using the ADO.NET 4.1 EDM Tools that includes the EDM item

template, the EDM wizard, EDM Designer, entity mapping details, and the entity model browser.

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

First, let's take a closer look at the ADO.NET 4.1 EDM.

3.4.8.2 The ADO.NET 4.1 Entity Data Model

The ADO.NET 4.1 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.

EDM design approaches can be divided into three categories:

1. Database First
2. Model First
3. Code First

Let's have a brief discussion for each of these components one by one.

3.4.8.2.1 Database First Entity Framework supports several approaches for creating EDMs. Database First approach was historically the first one. It appeared in Entity Framework v1, and its support was implemented in Visual Studio 2008. This approach considers that an existing database is used, or the new database is created first, and then EDM is generated from this database with EDM Wizard.

All needed model changes in its conceptual (CSDL) and mapping (MSL) part are performed with EDM Designer. If the storage part needs changing, the database must be modified first, and then EDM is updated with Update Model Wizard.

Database First approach is supported in Visual Studio 2008/2010 for MS SQL Server only. However, there are third-party solutions that provide Database First support in Visual Studio for other database servers: DB2, EffiProz, Firebird, Informix, MySQL, Oracle, PostgreSQL, SQLite, Sybase, and VistaDB. Besides, there are third-party tools that extend or completely replace standard EDM Wizard, EDM Designer, and Update Model Wizard.

3.4.8.2.2 Model First A Model First approach was supported in Visual Studio 2010, which was released together with the second Entity Framework version (Entity Framework 4.0). In the Model First approach, the development starts from scratch. At first, the conceptual model is created with EDM Designer, entities and relations are added to the model, but mapping is not created. After this, Generate Database Wizard is used to generate storage (SSDL) and mapping (MSL) parts from the conceptual part of the model and save them to the **edmx** file. Then the wizard generates DDL script for creating the database, which includes tables and foreign keys.

If the model was modified, the Generate Database Wizard should be used again to keep the model and the database consistent. In such case, the generated DDL script

contains DROP statements for tables, corresponding to old SSDL from the **.edmx** file, and CREATE statements for tables, corresponding to new SSDL, generated by the wizard from the conceptual part. In the Model First approach, the developer should not edit storage part or customize mapping, because they will be regenerated each time when Generate Database Wizard is launched.

Model First in Visual Studio 2010 is supported only for MS SQL Server. However, there are third-party solutions that provide support for other databases, such as Oracle, MySQL, and PostgreSQL. Besides, there are third-party tools for complete replacement of EDM Designer and Generate Database Wizard in the context of Model First approach.

3.4.8.2.3 Code First Entity Framework 4.1 Release to Web (RTW) is a new technique that is the first fully supported go-live release of the DbContext API and Code First development workflow.

Code First allows you to define your model using Visual C# or Visual Basic.NET classes; optionally additional configuration can be performed using attributes on your classes and properties or by using a Fluent API. Your model can be used to generate a database schema or to map to an existing database.

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

- The ADO.NET 4.1 EDM item template is available for Visual Basic.NET project type, 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 Visual C# or Visual Basic.NET 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.

3.4.8.2.4 Entity Data Model Item Template The ADO.NET 4.1 EDM item template is the starting point to the EDM tools. The ADO.NET 4.1 EDM item template is available for Visual C# and Visual Basic.NET 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 4.1 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 4.1 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 auto-generated, and is updated when the .edmx file changes, and is compiled as part of the project.

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

3.4.8.2.5 Entity Data Model Wizard The EDM Wizard is used to generate an .edmx file. It also allows you to create a model from an existing database, or to generate an empty model.

The EDM Wizard starts after you add an ADO.NET 4.1 EDM 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 4.1 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 auto-generated 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:** By selecting **Generate from database**, you can generate an .edmx file from an existing database. Then, the EDM Wizard will guide you through selecting a data source, database, and database objects to include in the conceptual model. By selecting **Empty model**, you can add an .edmx file that contains empty a conceptual model, a storage model, and mapping sections to your project. Select this option if you plan to use the Entity Designer to build your conceptual model and later generate a database that supports the model.
- **Choose the Database Connection:** You can choose an existing connection from the drop-down 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.

Beginning with Visual Studio 2010, the **Choose Your Database Objects** dialog box also allows you to perform the following customizations:

1. Apply English-language rules for singulars and plurals to entity, entity set, and navigation property names when the .edmx file is generated.
2. Include foreign key columns as properties on entity types.

Upon closing, the EDM Wizard creates an .edmx file that contains the model information. The .edmx file is used by the Entity Designer, which enables you to view and edit the conceptual model and mappings graphically.

Now let's have a closer look at the real part—ADO.NET 4.1 EDM Designer.

3.4.8.2.6 Entity Data Model Designer The ADO.NET 4.1 EDM Designer provides visual tools for creating and editing an EDM. You can use the Entity Model Designer to visually create and modify entities, associations, mappings, and inheritance relationships. You can also validate an .edmx file using the Entity Model Designer.

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.

The ADO.NET 4.1 EDM Designer is integrated with the Visual Studio.NET 2010 components. You can view and edit information using the Properties window, and errors are reported in the Error List.

Figure 3.22 shows an example of the ADO.NET 4.1 EDM Designer.

Two important functionalities of using the EDM Designer are:

Opening the ADO.NET EDM Designer: The ADO.NET 4.1 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**.

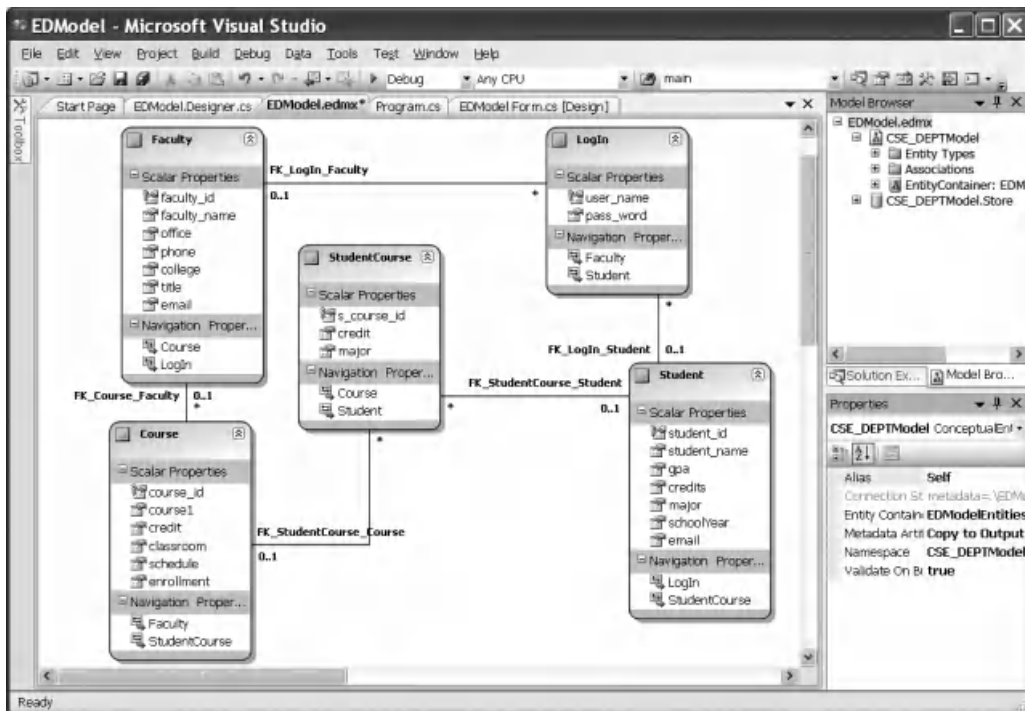


Figure 3.22. An example of ADO.NET 4.1 Entity Data Model Designer.

3.4.8.2.7 Entity Model Browser The Entity Model Browser is a Visual Studio tool window that is integrated with the ADO.NET 4.1 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 an item in the Model Browser makes it active in the **Properties** window and the **Mapping Details** window. You can use these windows to modify the properties and entity mappings.
- Create a function import from a stored procedure.
- Update the storage model when changes are made to the underlying database.
- Delete tables, views, and stored procedures from the storage model.
- Locate an entity type on the design surface. In the **Model Browser**, right-click the entity name in the tree view of the conceptual model and select **Show in Designer**. The visual representation of the model will be adjusted so that the entity type is visible on the design surface.
- Search the tree view of the conceptual and storage models. The search bar at the top of the Model Browser window allows you to search object names for a specified string.

The Entity Model Browser opens when the ADO.NET 4.1 EDM Designer is opened. If the Entity Model Browser is not visible, right-click on the main design surface and select **Model Browser**.

3.4.8.3 Using the ADO.NET 4.1 Entity Data Model Wizard

In this section, we will use a project example to illustrate how to use the EDM Wizard to develop a Database First 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 Basic.NET 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 Database First Visual Basic.NET Windows-based project named EDMModel.

3.4.8.3.1 Create a New Database First Visual Basic.NET Windows-Based Project

Open Visual Studio.NET 2010 and select File/New Project items to create a new project. Perform the following operations to create this new project:

1. Select **Visual Basic** as the project type and **Windows Forms Application** as the Template for this new project.
2. Click on **Other Project Types** and **Visual Studio Solutions** from the Recent Templates window.
3. Enter **EDModel Solution** into the Name box and select any folder as the Location to save this project. Then click on the OK button to create this new project.
4. Right-click on the newly created blank solution from the Solution Explorer window and select **AddNew Project** item.
5. Enter **EDModel Project** into the Name box and click on the OK button.

Now perform the following operations to change the properties of this new project:

1. Change the file object's name from **Form1.vb** to **EDModel Form.vb**.
2. Change the Windows Form object's name from **Form1** to **EDModelForm**.
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 **EDModelForm** window should match the one that is shown in Figure 3.23.

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

3.4.8.3.2 Generate the Entity Data Model Files Before we can continue to generate the EDM files, we must first confirm whether we have installed ADO.NET Entity Framework 4.1 and ADO.NET 4.1 Entity Framework Tools in our computer. To do this confirmation, just right-click the project **EDModel Project** and select **AddNew Item** to

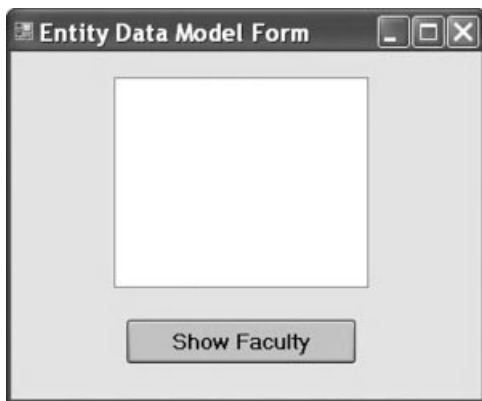


Figure 3.23. The **EDModelForm** window.

open the Add New Item dialog box. If you cannot find the item **ADO.NET Entity Data Model** from the Templates box, this means that you have not installed ADO.NET Entity Framework 4.1 and its Tools. Therefore, let's first download these components and install them in your computer.

One point to be noted is that you should have installed NET Framework 4.0 RTM before you can install the ADO.NET Entity Framework 4.1. Go to the site <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=9cfb2d51-5ff4-4491-b0e5-b386f32c0992&displaylang=en> to download and install NET Framework 4.0.

Let's perform the following operations to complete the download and installation for ADO.NET Entity Framework 4.1 and ADO.NET 4.1 Entity Framework Tools:

1. Open the Microsoft download home page: <http://msdn.microsoft.com/en-us/data/aa937723.aspx>.
2. Select the item **Download Entity Framework 4.1** under the **Learn Entity Framework** title.
3. On the opened download window, click on the **Download** button for the item `EntityFramework41.exe` and click on the **Run** button to begin the downloading and installation process.
4. Follow the screen's instructions to complete this downloading and installation process. Click on the **Finish** button when this process is done.

Now we can continue to the process to generate the EDM Files and use the EDM Wizard.

Perform the following operations to generate our EDM Wizard:

1. Right-click the project **EDModel Project** 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 **ADO.NET Entity Data Model** from the Installed Templates window and click on the **Add** button to open the EDM Wizard.
3. This wizard allows us to choose the model contents and generate a model from our database. Keep the default selection **Generate from database** unchanged and click on the **Next** button.
4. The next wizard enables us to set up a database connection to our target database. Click on the **New Connection** button to generate a new connection to our target SQL Server 2008 sample database `CSE_DEPT.mdf`.
5. On the opened Connection Properties wizard, make sure that the Data Source box contained the **Microsoft SQL Server Database File**. Otherwise, you need to click on the **Change** button to select the correct data source type. Then click on the **Browse** button to find our target database file `CSE_DEPT.mdf`. Regularly, this file should be located at the folder: `C:\Program Files\Microsoft SQL Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA`. Browse to that folder and select our database file `CSE_DEPT.mdf`, and click on the **Open** button.
6. Your finished Connection Properties wizard should match the one that is shown in Figure 3.24.
7. Before we can test this database connection, click on the **Advanced** button to open the Advanced Properties wizard to confirm that we are using the correct data source for this connection. Make sure that the **Data Source** box contains our current target database engine **.SQL2008EXPRESS**. Click on the **OK** button to complete this confirmation.



Figure 3.24. The Connection properties wizard.

8. Click on the **Test Connection** button to test this database connection. A successful connection dialog box should be displayed if this connection is fine. Click on the **OK** button to return to the EDM Wizard, which is shown in Figure 3.25. Click on the **Next** button to continue.
9. Click on the **Yes** button to the next dialog box to enable the database file to be copied into our current project with the modified connection.
10. In the next window, check all components related to our connected sample database since we may need to use some or all of them in our project. Your finished EDM Wizard should match the one that is shown in Figure 3.26. Click on the **Finish** button to complete this database selection and connection process.
11. On the opened Solution Explorer window, change the name of newly created EDM file from **Model1.edmx** to **EDModel.edmx**.
12. Your finished Solution Explorer window is shown in Figure 3.27.

To see this EDM in Designer view, double-click the newly added Entity Data Model **EDModel.edmx**, and the Designer view is shown in Figure 3.28.

Five tables and connections between them are displayed in this view. On each table, two groups of entity properties are displayed, **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-foreign keys) in this database. The connections between each entity (mapped to data table) are called associations.

As you double-click this Entity Data Model **EDModel.edmx**, another tool, **Mapping Details**, is also displayed under this Designer view, which is shown in Figure 3.29.



Figure 3.25. The Entity Data Model Wizard.

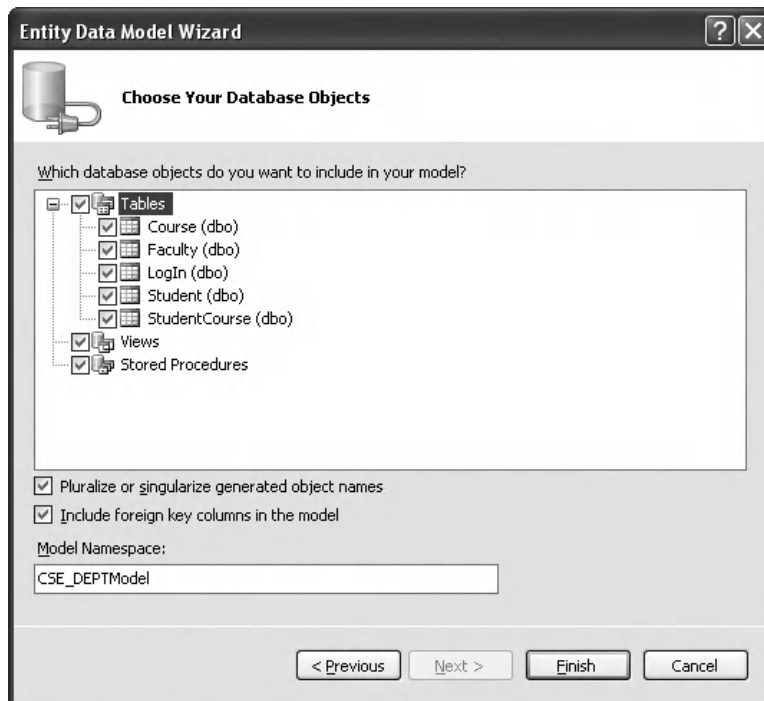


Figure 3.26. The finished Entity Data Model Wizard.

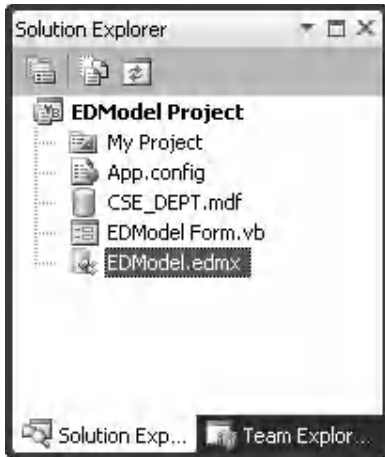


Figure 3.27. The finished Solution Explorer window.

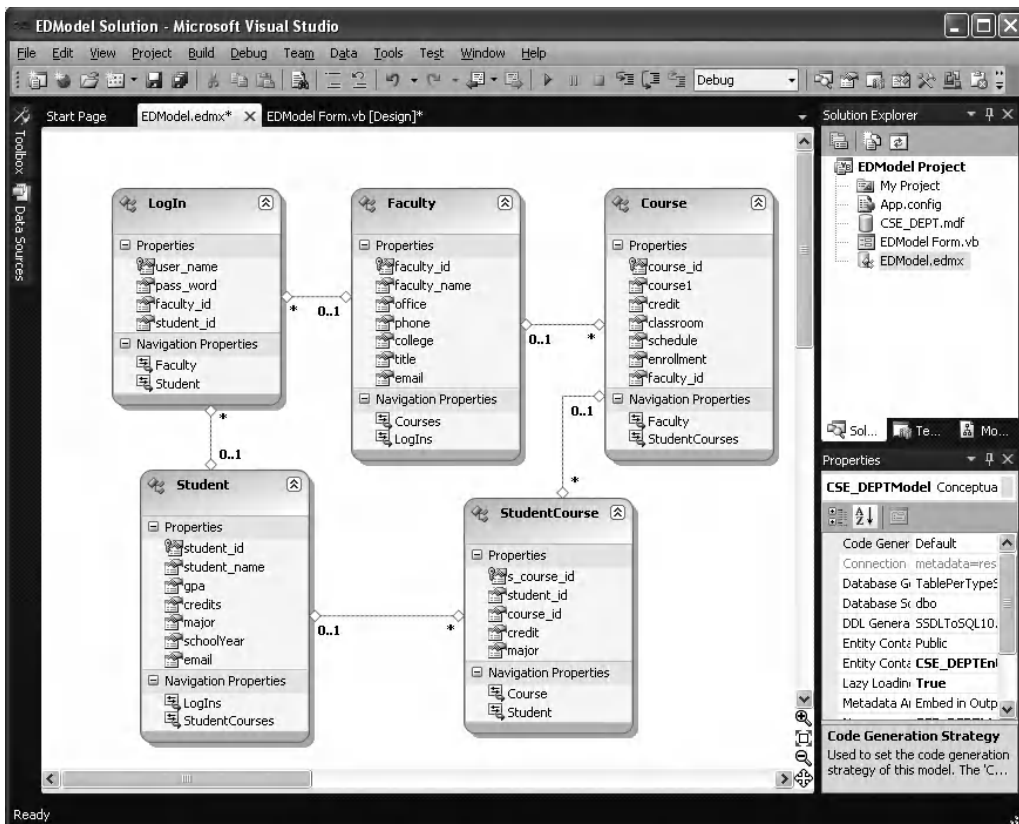


Figure 3.28. The Designer view of the Entity Data Model EDModel.

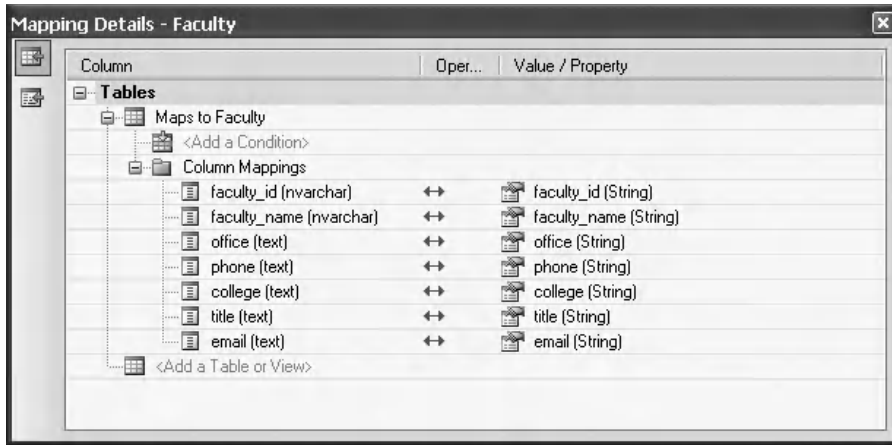


Figure 3.29. An example of Mapping Details—Faculty entity.

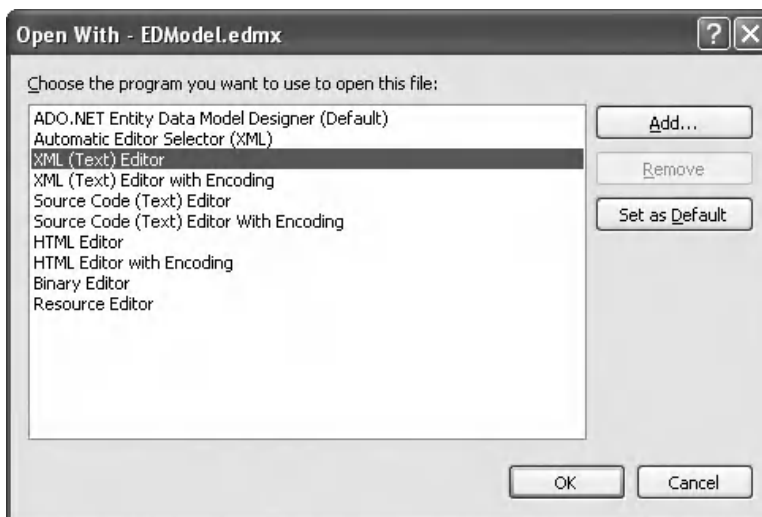


Figure 3.30. The Open With dialog box.

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 EDMModel is also created. To open this file, right-click our newly created Entity Data Model EDMModel.edmx from the Solution Explorer window and select the item **Open With** to open the Open With dialog box, which is shown in Figure 3.30.

Select the item **XML (Text) Editor** and then click the OK to open this XML mapping file. Regularly, the EDM file EDMModel.edmx should be closed before this XML file can be opened. Click on the Yes button to the MessageBox to open this file.

Now if you open the App.Config file, you can find that our connection string, **CSE_DEPTEntities**, which we created using the EDM 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 Basic.NET data-driven application to show readers how to make it work.

3.4.8.3.3 Use the ADO.NET 4.1 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 EDMModelForm 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 EDMModelForm 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 event procedure. Select the form object **EDMModel Form.vb** from the Solution Explorer window and click the View Designer button to open its form window. Double-click the Show Faculty button to open its Click event procedure, and enter the following codes that are shown in Figure 3.31 into this method.

Let's have a closer look at this piece of codes 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.

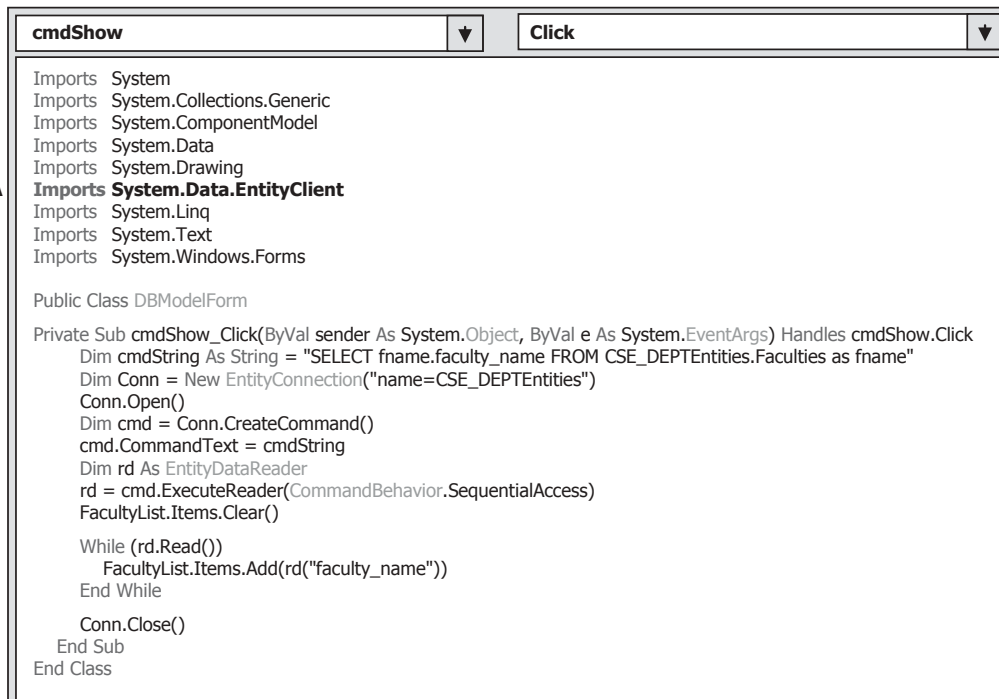


Figure 3.31. The codes for the cmdShow_Click event procedure.

- 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 **Faculties** 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 **Faculties** 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.
- F. A new `EntityDataReader` instance **rd** is created, and this instance works as a data reader to call the `ExecuteReader()` method to retrieve all desired data later.
- G. The `ExecuteReader()` method is called to retrieve back all **faculty_name** and assign them to the `EntityDataReader` object **rd**.
- H. The listbox control **FacultyList** is cleaned up before it can be filled.
- I. A While loop is utilized to pick up all **faculty_name** from the `EntityDataReader` **rd** 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 Basic.NET, a pair of parenthesis is used to indicate each element on 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.
- J. Finally the connection is closed to release the connection object.

Now, let's run the project to test our codes. Click the Start Debugging button to run the project. The `EDModelForm` window is displayed, which is shown in Figure 3.32. Click on the Show Faculty button to connect to our sample database and retrieve all faculty names. The running result is shown in Figure 3.32.

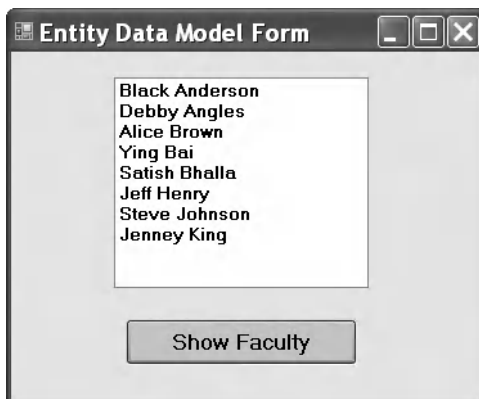


Figure 3.32. The running result of the project `EDModel`.

Click on the Close button that is located at the upper-right corner of this form to stop and close our project.

It can be found from this piece of codes 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 ADO.NET, which includes the architectures, organizations, and components of ADO.NET.

Detailed discussions and descriptions are provided in this chapter to give readers both fundamental and practical ideas and pictures in 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 preferred projects based on the different organizations of ADO.NET.

A history of the development of ADO.NET is first introduced in this chapter, which includes ADO.NET 2.0, ADO.NET 3.5, and ADO.NET 4.1. Different data-related objects are discussed, such as DAO, RDO, ODBC, OLEDB, and ADO. The difference between ADO and ADO.NET is provided in detail.

Fundamentally, 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 the 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 the Data Provider is used for one specific database. But 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 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 a connection between your applications and your data source is the connection string, which has different formats and styles depending on the database you are using. The popular components of the connection string include Provider, Data Source, Database, Use 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.

By finishing this chapter, you should be able to:

- Understand the architecture and organization of ADO.NET
- Understand three components of ADO.NET, 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 a 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

In Chapter 5, we will discuss the data query technique with two methods: using Tools and Wizards provided by Visual Studio.NET 2010, and using the runtime object method to develop simple, but efficient data query applications with three databases: Access, SQL Server, and Oracle. Both methods are introduced in two parts: Part I: Using the tools and wizards provided by Visual Studio.NET 2010 to develop data query project, and Part II: Using the Runtime objects to perform the data query job for three databases.

HOMEWORK

I. True/False Selections

- ____ 1. ADO.NET is composed of four major components: Data Provider, DataSet, DataReader, and DataTable.
- ____ 2. ADO is developed based on OLE and COM technologies.
- ____ 3. ADO.NET is a new version of ADO, and it is based mainly on the Microsoft .NET Framework.
- ____ 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, and it 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

- To populate data from a database to a `DataSet` object, one needs to use the _____.
 - Data Source
 - `DataAdapter` (`TableAdapter`)
 - Runtime object
 - Wizards
- The `Parameters` property of the `Command` class _____.
 - Is a `Parameter` object
 - Contains a collection of `Parameter` objects
 - Contains a `Parameter` object
 - Contains the parameters of the `Command` object
- To add a `Parameter` object to the `Parameters` property of the `Command` object, one needs to use the _____ method that belongs to the _____.
 - `Insert, Command`
 - `Add, Command`
 - `Insert, Parameters collection`
 - `Add, Parameters collection`
- `DataTable` class is a container that holds the _____ and _____ objects.
 - `DataTable, DataRelation`
 - `DataRow, DataColumn`
 - `DataRowCollection, DataColumnCollection`
 - `Row, Column`
- 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`.
 - `Rows, column`
 - `Columns, column`
 - `Row, row`
 - `Rows, row`
- The _____ data provider can be used to execute the data query for _____ data providers.
 - SQL Server, `OleDb` and Oracle
 - `OleDb`, SQL Server and Oracle
 - Oracle, SQL Server and `OleDb`
 - SQL Server, `Odbc` and Oracle

7. To perform a Fill() method to fill a data table, it executes the _____ 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 _____ 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 ADO.NET, and illustrate the functionality of these two architectures using block diagrams.
2. List three basic components of ADO.NET 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. Assuming 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. List three new features used for ADO.NET Entity Framework 4.1 to facility the database developments.

Chapter 4

Introduction to Language Integrated Query (LINQ)

Language Integrated Query (LINQ) is a groundbreaking innovation in Visual Studio 2010 and .NET Framework version 4.0 that bridges the gap between the world of objects and the world of data.

Traditionally, queries against 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 against strongly typed collections of objects by using language keywords and familiar operators.

In Visual Studio.NET, you can write LINQ queries in Visual Basic.NET with SQL Server databases, XML documents, ADO.NET DataSets, and any collection of objects that supports *IEnumerable* or the generic *IEnumerable(Of T)* interface. As we mentioned in Chapter 3, LINQ support for the ADO.NET 4.1 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 should be developed under the .NET Framework 4.0 environment.

Before we can dig deeper on LINQ, we had better have a general and global picture about LINQ. Let's start from the basic introduction about the 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 (SQOs). Most of these methods operate on sequences, where a sequence is an object whose type implements the *IEnumerable(Of T)* interface or the *IQueryable(Of 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 the directive like: `Imports System.Linq` in the namespace declaration section of the code windows.

There are some confused signs and terminologies, such as `IEnumerable`, `IEnumerable(Of T)`, `IQueryable`, and `IQueryable(Of 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(Of T)`, `IQueryable`, and `IQueryable(Of 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# and Visual Basic and the other two interfaces, `IEnumerable(Of T)` and `IQueryable(Of T)`, are used to convert the data type of collections compatible with those in the `System.Collection.Generic` in Visual Basic.NET to either `IEnumerable(Of T)` (LINQ to Objects) or `IQueryable(Of 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(Of T)` or `IEnumerable` interfaces. But a LINQ to SQL query is performed on classes that implement the `IQueryable(Of T)` interface. The relationship between the `IEnumerable(Of T)` and the `IQueryable(Of T)` interfaces is: `IQueryable(Of T)` implements `IEnumerable(Of T)`, therefore, besides the Standard Query Operator (SQO), the LINQ to SQL queries have additional query operators, since it uses the `IQueryable(Of T)` interface.

4.1.1.1 The `IEnumerable` and `IEnumerable(Of T)` Interfaces

The `IEnumerable(Of T)` interface is a key part of LINQ to Objects, and it allows all of early 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(Of 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(Of int)`, you have a sequence or a collection of `ints`.

For nongeneric collections that exist in the old version of Visual Basic, they support the `IEnumerable` interface, but they do not support the `IEnumerable(Of 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(Of 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(Of T)`.

Here, a coding example of using LINQ to Object is shown in Figure 4.1.

The type `IEnumerable(Of int)` plays two important roles in this piece of codes.

1. The query expression has a data source called `intArray` that implements `IEnumerable(Of int)`.
2. The query expression returns an instance of `IEnumerable(Of int)`.

```
// create an integer array
Dim myArray() As Integer = {1, 2, 3, 4, 5}

Dim intArray As IEnumerable(Of Integer) = myArray.Select(Function(i) i)
Dim query = From num In intArray
            Where num >= 3
            Select num
For Each intResult In query
    Console.WriteLine(intResult)
Next
Console.WriteLine("Press any key to exit")
Console.ReadKey()
```

Figure 4.1. A coding example of using LINQ to Object query.

```
// create an integer array
Dim myArray() As Integer = {1, 2, 3, 4, 5}

Dim query As IEnumerable(Of Integer) = From num In myArray
                                       Where num >= 3
                                       Select num

For Each intResult In query
    Console.WriteLine(intResult)
Next
Console.WriteLine("Press any key to exit")
Console.ReadKey()
```

Figure 4.2. A modification of the coding example of using LINQ to Object query.

Every LINQ to Objects query expression, including the one shown above, will begin with a line of this type:

From x In y

In each case, the data source represented by the variable **y** must support the `IEnumerable(Of 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 in a way that is shown in Figure 4.2.

This code makes explicit the type of the variable returned by this query, `IEnumerable(Of int)`. In practice, you will find that most LINQ to Objects queries return `IEnumerable(Of T)`, for different data type **T**.

By finishing these two examples, it should be clear to you that interface `IEnumerable` and `IEnumerable(Of 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(Of T)` as its data source, but it also returns an instance with the same type.

4.1.1.2 The *IQueryable* and *IQueryable(Of T)* Interfaces

As we discussed in the previous section, `IQueryable` and `IQueryable(Of T)` are two interfaces used for LINQ to SQL queries. Similar to `IEnumerable` and `IEnumerable(Of T)`

```

`create a database connection using the DataContext object
Dim cse_dept As CSE_DEPTDataContext

`create local string variables
Dim username As String = String.Empty
Dim password As String = String.Empty

`LINQ query expression
Dim loginfo As IQueryable(Of LogIn) = From lg In cse_dept.LogIns
                                     Where lg.user_name = txtUserName.Text &
                                     lg.pass_word = txtPassWord.Text
                                     Select lg

For Each log In loginfo
    username = log.user_name
    password = log.pass_word
Next

```

Figure 4.3. A coding example of using LINQ to SQL query.

interfaces, in which the Standard Query Operator methods are defined as the static members in the Enumerable class, the Standard Query Operator methods applied for the IQueryable(Of T) interface are defined as static members of the Queryable class. The IQueryable interface is mainly used for the nongeneric collections, and the IQueryable(Of T) is used for generic collections. Another point is that the IQueryable(Of T) interface is inherited from the IEnumerable(Of T) interface from the Queryable class, and the definition of this interface is:

```
interface IQueryable(Of T) : IEnumerable(Of T), Queryable
```

From this inheritance, one can treat an IQueryable(Of T) sequence as an IEnumerable(Of T) sequence.

Figure 4.3 shows an example of using the IQueryable interface to perform a query to the 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(Of T) interface, a Standard Query Operator, is utilized to perform this query. The LogIn works as a type in the IQueryable(Of 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(Of T) and the other that operates on objects of type IQueryable(Of 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 that they operate on. This means that they can be called by using either static method syntax or instance method syntax.

In addition, several standard query operator methods operate on types other than those based on `IEnumerable(Of T)` or `IQueryable(Of T)`. The `Enumerable` type defines two such methods that both operate on objects of type `IEnumerable`. These methods, `Cast(Of TResult)(IEnumerable)` and `OfType(Of TResult)(IEnumerable)`, let you enable a nonparameterized, 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(Of TResult)(IQueryable)` and `OfType(Of 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 methods that operate on in-memory collections, that is, those methods that extend `IEnumerable(Of 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(Of 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(Of T)` object.

Calls to query methods can be chained together in one query, which enables queries to become arbitrarily complex.

According to their 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 most often used Standard Query Operators.

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

```

FacultyDataAdapter.SelectCommand = accCommand
FacultyDataAdapter.Fill(ds, "Faculty")
Dim facultyinfo = From fi In ds.Tables("Faculty").AsEnumerable()
                  Where fi.Field(Of String)("faculty_name").Equals(ComboName.Text)
                  Select fi

For Each fRow in facultyinfo
    'Display selected fRow elements...
Next

```

Figure 4.4. An example code for the operator `AsEnumerable`.

Because of the limitation of the space, we will select some of the most often used Standard Query Operator methods and give a detailed discussion for them one by one.

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.

4.1.3.1 *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(Of T)` to `IEnumerable(Of T)` itself. This means that if an input sequence has a type of `IEnumerable(Of 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 the type of `IEnumerable(Of T)` in order to use the LINQ technique, because all data operations in LINQ use Standard Query Operator methods that can perform complex data queries on an `IEnumerable(Of T)` sequence. A compiling error would be encountered without this operator.

4.1.3.2 *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 `IEnumerable(Of T)` interface, and it can convert any object with an `IEnumerable` type to `IEnumerable(Of T)` type.

An example coding using this operator is shown in Figure 4.5.

4.1.3.3 *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.

```
Dim fruits As New System.Collections.ArrayList()
fruits.Add("apple")
fruits.Add("mango")
Dim query As IEnumerable(Of String) = fruits.Cast(Of String).Select(Function(fruit) fruit)
For Each fruit In query
    Console.WriteLine(fruit)
Next
```

'the running result of this piece of codes is:

apple
mango

Figure 4.5. An example code for the operator Cast.

```
Dim courseinfo = Course.Join(Faculty, Function(ci) ci.faculty_id, _
    Function(fi) fi.faculty_id, _
    Function(ci, fi) New With {.faculty_name = ComboName.Text And course_id = ci.course_id})

For Each cid In courseinfo
    CourseList.Items.Add(cid.course_id)
Next
```

Figure 4.6. An example code for the operator Join.

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 backwards 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 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 dataset 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 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 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 a join operator method shown in Figure 4.6.


```

Dim fruits As New System.Collections.ArrayList(2)
fruits.Add("Mango")
fruits.Add("Orange")
Dim query As IEnumerable(Of String) = fruits.OfType(Of String)()
Console.WriteLine("Elements of type 'string' are:" & vbCrLf)
For Each fruit As String In query
    Console.WriteLine(fruit)
Next

```

'the running result of this piece of codes is:

Elements of type 'string' are:
Mango
Orange

Figure 4.7. An example code for the operator `OfType`.

4.1.3.4 *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 `For Each` in Visual Basic.NET.

An example coding of using this operator is shown in Figure 4.7.

The `OfType(Of TResult)(IEnumerable)` method returns only those elements in the source that can be cast to type `TResult`. To instead receive an exception if an element cannot be cast to type `TResult`, use `Cast(Of TResult)(IEnumerable)`.

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(Of TResult)` extends the type `IEnumerable`. `OfType(Of TResult)` cannot only be applied to collections that are based on the parameterized `IEnumerable(Of T)` type, but collections that are based on the nonparameterized `IEnumerable` type also.

By applying `OfType(Of 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(Of TResult)` would return an object of type `IEnumerable(Of Object)` in Visual Basic, to which the standard query operators can be applied.

4.1.3.5 *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(Of TElement)`. Both `IEnumerable` and `IQueryable` classes contain this operator method.

An example coding of using this operator is shown in Figure 4.8.

```

Sub OrderByEx()
    'Create an array of Pet objects.
    Dim pets() As Pet = {New Pet With {.Name = "Barley", .Age = 8}, _
        New Pet With {.Name = "Boots", .Age = 4}, _
        New Pet With {.Name = "Whiskers", .Age = 1}}

    Dim query As IEnumerable(Of Pet) = pets.OrderBy(Function(pet) pet.Age)
    For Each pt As Pet In query
        Console.WriteLine(pt.Name & " - " & pt.Age)
    Next
End Sub

```

'the running result of this piece of codes is:

```

Whiskers - 1
Boots - 4
Barley - 8

```

Figure 4.8. An example code for the operator OrderBy.

```

Dim squares As IEnumerable(Of Integer) = Enumerable.Range(1, 5).Select(Function(x) x * x)
For Each num As Integer In squares
    Console.WriteLine(num)
Next

```

'the running result of this piece of codes is:

```

1
4
9
16
25

```

Figure 4.9. An example code for the operator Select.

4.1.3.6 *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 `For Each` in Visual Basic.NET.

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 Basic.NET clause translates to an invocation of `Select`.

An example coding of using this operator is shown in Figure 4.9.

4.1.3.7 *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 `For Each` in Visual Basic.NET.

An example coding of using this operator is shown in Figure 4.10.

In query expression syntax, a `Where` in Visual Basic.NET clause translates to an invocation of `Where(Of TSource)IEnumerable(Of TSource), Func(Of TSource, Boolean)`.

4.1.4 Nondeferred Standard Query Operators

Some of the most often used nondeferred Standard Query Operator methods are discussed in this section.

4.1.4.1 *ElementAt (Element Purpose)*

This operator method returns the element at a specified index in a sequence. If the type of source implements `IList(Of 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(Of TSource)` method.

An example coding of using this operator is shown in Figure 4.11.

```
Dim fruits As New List(Of String)(New String() {"apple", "passionfruit", "banana", "mango", _
                                                "orange", "blueberry", "grape", "strawberry"})
Dim query As IEnumerable(Of String) = fruits.Where(Function(fruit) fruit.Length < 6)
For Each fruit in query
    Console.WriteLine(fruit)
Next
```

the running result of this piece of codes is:

```
apple
mango
grape
```

Figure 4.10. An example code for the operator `Where`.

```
`Create a string array
Dim names() As String = _
    {"Hartono, Tommy", "Adams, Terry", "Andersen, Henriette Thaulow", "Hedlund, Magnus", "Ito, Shu"}
Dim name As String = names.ElementAt(2)
Console.WriteLine("The name chosen at position 2 is " & name)
```

the running result of this piece of codes is:

```
Andersen, Henriette Thaulow
```

Figure 4.11. An example code for the operator `ElementAt`.

4.1.4.2 First (Element Purpose)

This operator method returns the first element of an input sequence. The method `First(Of TSource)(IEnumerable(Of 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.

4.1.4.3 Last (Element Purpose)

This operator method returns the last element of a sequence. The method `Last(Of TSource)(IEnumerable(Of TSource))` throws an exception if the source contains no 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.

4.1.4.4 Single (Element Purpose)

This operator method returns a single, specific element of an input sequence of values. The `Single(Of TSource)(IEnumerable(Of TSource))` method throws an exception if the input sequence is empty. To instead return **Nothing** when the input sequence is empty, use `SingleOrDefault`.

An example coding of using this operator is shown in Figure 4.14.

```
'Create a string array
Dim numbers() As Integer = {9, 34, 65, 92, 87, 435, 3, 54, 83, 23, 87, 435, 67, 12, 19}

'Select the first element in the array
Dim first As Integer = numbers.First()

Console.WriteLine(first)

'the running result of this piece of codes is:
9
```

Figure 4.12. An example code for the operator `First`.

```
Dim numbers() As Integer = {9, 34, 65, 92, 87, 435, 3, 54, 83, 23, 87, 67, 12, 19}

Dim last As Integer = numbers.Last()

Console.WriteLine(last)

'the running result of this piece of codes is 19
```

Figure 4.13. An example code for the operator `Last`.

```
Dim fruits() As String = {"orange"}

Dim fruit As String = fruits.Single()

Console.WriteLine(fruit)

'the running result of this piece of codes is: orange
```

Figure 4.14. An example code for the operator `Single`.

4.1.4.5 ToArray (Conversion Purpose)

This operator method converts a collection to an array. This method forces query execution. The `ToArray(Of TSource)(IEnumerable(Of 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.

4.1.4.6 ToList (Conversion Purpose)

This operator method converts a collection to a `List(Of T)`. This method forces query execution. The `ToList(Of TSource)(IEnumerable(Of TSource))` method forces immediate query evaluation and returns a `List(Of 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 Standard Query Operator methods, and they are actual methods to be executed to perform a LINQ query. Next,

```
Module ToArrayClass
Sub Main()
    Dim sArray As String() = {"G", "H", "a", "H", "over", "Jack"}
    Dim names As String() = sArray.OfType(Of String).ToArray()
    For Each name In names
        Console.WriteLine(name)
    Next
End Sub
End Module
'the running result of this piece of codes is:
GHaHoverJack
```

Figure 4.15. An example code for the operator `ToArray`.

```
Dim fruits() As String = {"apple", "banana", "mango", "orange", "blueberry", "grape", "strawberry"}
Dim lengths As List(Of Integer) = fruits.Select(Function(fruit) fruit.Length).ToList()
For Each length As Integer In lengths
    Console.WriteLine(length)
Next
'the running result of this piece of codes is:
5
6
5
6
9
5
10
```

Figure 4.16. An example code for the operator `ToList`.

we will go ahead to 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:

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. Visual Basic.NET 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 Standard Query Operator methods we discussed in the last section, 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(Of T)* or *IQueryable(Of 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 to directly access and 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 the 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 codes are shown in Figure 4.17.

The exact running process of this piece of codes 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 **For Each** loop.

Let's have a closer look at this piece of codes 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*; it implicitly supports the generic *IEnumerable(Of T)* interface. This fact means it can be queried with LINQ. A query is executed in a **For Each** loop, and **For Each** requires *IEnumerable* or *IEnumerable(Of T)*. Types that support *IEnumerable(Of T)* or a derived interface, such as the generic *IQueryable(Of 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 shaped before it is returned. A query is stored in a query variable and initialized with a query expression.

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.

```
Module IntroLINQ
    Sub Main()
        '1. Data Source
        Dim numbers As Integer() = {0, 1, 2, 3, 4, 5, 6}

        '2. Query creation. The numQuery is an IEnumerable(Of Int)
        Dim numQuery = From num In numbers
                        Where (num Mod 2) = 0
                        Select num

        '3. Query execution.
        For Each num In numQuery
            Console.Write("{0,1} ", num)
        Next
        Console.WriteLine(vbNewLine & "Press any key to continue...")
        Console.ReadKey()

    End Sub
End Module

'the running result of this piece of codes is: 0 2 4 6
```

Figure 4.17. An example code for the LINQ query.

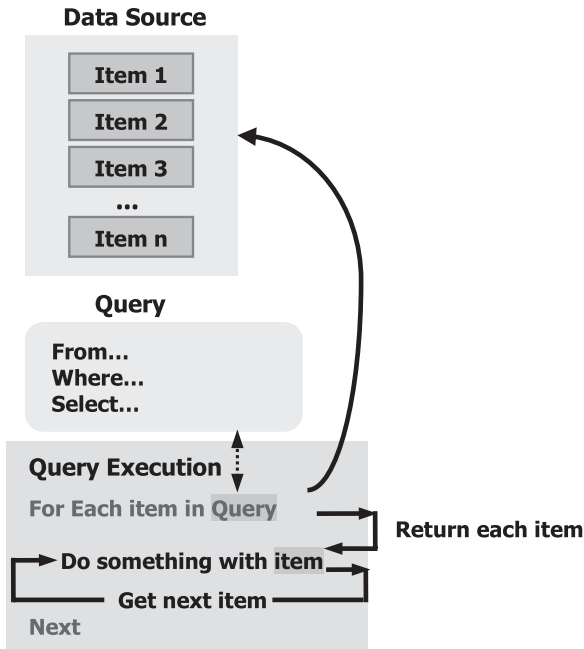


Figure 4.18. The running process of a LINQ query.

```

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. A typical query expression of LINQ query.

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. 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 **For Each** 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.Write()` 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 For Each statement because the query itself must use For Each 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(Of TSource) or ToArray(Of TSource) methods. You can also force execution by putting the For Each 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 THE 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 three components are located at the different namespaces provided by .NET Framework 4.0, which is shown in Table 4.2.

A typical LINQ architecture is shown in Figure 4.20.
Now let's give a brief introduction for each component in LINQ.

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(Of 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

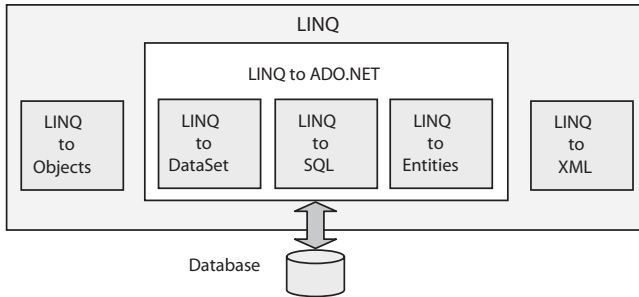


Figure 4.20. A typical LINQ architecture.

4.3.1 Overview of LINQ to Objects

The LINQ to Objects refers to the use of LINQ queries with any `IEnumerable` or `IEnumerable(Of 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 Standard Query Operator methods that are static methods of the static `System.Linq.Enumerable` class that you use to create LINQ to Objects queries. You can use LINQ to query any enumerable collections, such as `List(Of T)`, `Array`, or `Dictionary(Of 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. In 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 **For Each** 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 over 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

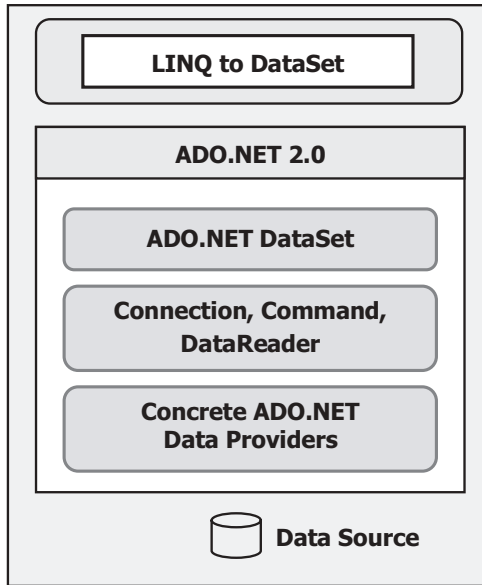


Figure 4.21. The relationship between LINQ to DataSet and ADO.NET 2.0.

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 over 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 can be 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 its all components, including `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 problem.

4.3.3 Overview of LINQ to SQL

LINQ to SQL belongs to LINQ to ADO.NET and it is a sub-component of LINQ to ADO.NET.

LINQ to SQL is a component of .NET Framework version 4.0 that 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.

Two popular LINQ to SQL Tools, **SQLMetal** and **Object Relational Designer**, are widely used in developing applications of using LINQ to SQL. The **SQLMetal** provides a DOS-like template with a black-and-white window. Developers using Visual Studio typically use the **Object Relational Designer**, which provides a graphic user interface (GUI) 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 (EDM) we discussed in Section 3.4.8.1 in Chapter 3, ADO.NET 4.0 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, and the data store.

It can be found that the Entities and EDM released by ADO.NET 4.0 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.

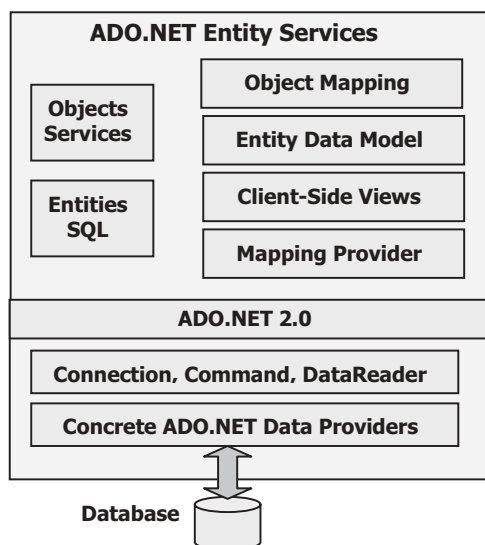


Figure 4.22. Relationship between LINQ to Entities, the Entity Framework, and ADO.NET.

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 4.0 is a conceptual data model 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 Language-Integrated Query (LINQ) Framework. LINQ to XML uses the latest .NET Framework language capabilities and is comparable with an updated, redesigned Document Object Model (DOM) XML programming interface. This interface was previously known as XLang 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 for the LINQ family. Now let’s go a little deeper into those topics to get a more detailed discussion for 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(Of T)` collections. Since any `IEnumerable` collection contains a sequence of objects with a data type that is compatible with `IEnumerable(Of 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(Of T)`, `Array`, or `Dictionary(Of TKey, TValue)`. All of these queries are performed by executing Standard Query Operator methods defined in the `IEnumerable` class. The difference between the `IEnumerable` and `IEnumerable(Of 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 have 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 the 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:

```
Dim query = From s As Student 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(Of TResult)` method. `Cast(Of TResult)` throws an exception if the specified cast cannot be performed. `Cast(Of TResult)` and `OfType(Of TResult)` are the two Standard Query Operator methods we discussed in Section 4.1.3, and these two methods operate on nongeneric IEnumerable types.

Let's illustrate this kind of LINQ query with a Visual Basic.NET example project named **NonGenericLINQ.vb**. Create a new Visual Basic.NET Console project and name it as **NonGenericLINQ**, and enter the codes that are shown in Figure 4.23 into the code window of this new project.

Let's have a closer look at this piece of codes 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 import this namespace in this project before it can be used.
- B.** A new Student class with two properties 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.
- E.** The *For Each* loop is used to pick up all query results one by one and assign it to the iteration variable *student*. 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.

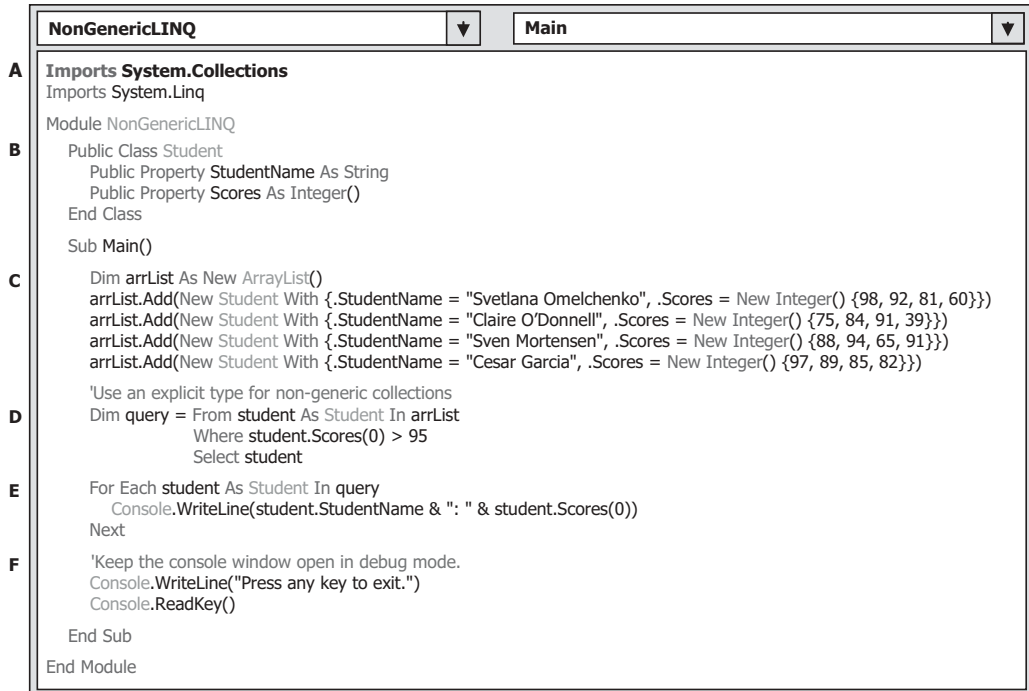


Figure 4.23. The codes for the example project NonGenericLINQ.

Now that you can Build and Run the project, the running result should be:

```

Svetlana Omelchenko: 98
Cesar Garcia: 97
Press any key to exit.

```

A complete Visual Basic.NET Console project named NonGenericLINQ can be found in the folder DBProjects\Chapter 4 that is located at the Wiley ftp site (refer to Fig. 1.2 in 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 semi-structured 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 the Number of Numeric Digits

Because the String class implements the generic IEnumerable(Of 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 Visual Basic.NET Console project and name it as **QueryStringLINQ**, and enter the codes that are shown in Figure 4.24 into the code window of this new project.

```

Module QueryStringLINQ
    Sub Main()
        A   Dim aString As String = "ABCDE99F-J74-12-89A"
        B   Dim stringQuery As IEnumerable(Of Char) = From ch In aString
                                                    Where Char.IsDigit(ch)
                                                    Select ch
        C   For Each c As Char In stringQuery
            Console.Write(c & " ")
        Next
        D   'Call the Count method on the existing query.
            Dim count As Integer = stringQuery.Count()
            Console.WriteLine(System.Environment.NewLine & "Count = " & count)
        E   'Select all characters before the first '-'
            Dim stringQuery2 As IEnumerable(Of Char) = aString.TakeWhile(Function(c) c <> "-")
        F   'Execute the second query
            For Each ch In stringQuery2
                Console.Write(ch)
            Next
        G   Console.WriteLine(System.Environment.NewLine & "Press any key to exit")
            Console.ReadKey()

        End Sub
    End Module
  
```

Figure 4.24. The codes for the example project QueryStringLINQ.

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

- A. 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.
- B. 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 element in this string collection will be filtered, selected, and returned. A `Cast()` operator is used for the returned query collection with an `IEnumerable(Of T)` interface, and `T` is replaced by the real data type `Char` here.
- C. The query is executed by using a For Each loop, and `c` is an iteration variable. The queried digits are displayed by using the `Console.WriteLine()` method.
- D. 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.
- E. 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.
- F. The second query is executed, and the result is displayed using the `Console.Write()` method.
- G. 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
Press any key to exit
```

A complete Visual Basic.NET Console project named `QueryStringLINQ` can be found in the folder `DBProjects\Chapter 4` that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

Our first LINQ and Strings 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 the 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.

First, let's create a new Visual Basic.NET Console project named `SortLinesLINQ` and save this project into an appropriate location in your computer. Then we need to create a sample text file `scores.csv` that will be used for our new project created above.

Open the NotePad editor and enter the codes that are 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 on the `File!Save As` menu item from the NotePad editor to open the Save As dialog box. Browse to the folder in which our new Visual Basic.NET Console project

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. The content of the sample text file `scores.csv`.

SortLinesLINQ Solution is located. Enter “**scores.csv**” into the File name box and click on 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 Visual Basic.NET Console project **SortLinesLINQ**.

Open the code window of our new Visual Basic.NET Console project **SortLinesLINQ** and enter the codes that are shown in Figure 4.26 into this code window.

Let’s have a closer look at this piece of codes to see how it works.

- A.** 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(Of 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.
- B.** 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 criterion by selecting any other column if you like.
- C.** The query is built and executed by calling a function `RunQuery()` with two arguments: the data source *scores* and the filtering criteria *sortField*. The queried results are displayed by executing the method `Console.WriteLine()`.
- D.** The purpose of these two coding lines is to allow users to run this project in a Debugging mode.
- E.** The body of the function `RunQuery()` starts from here. One point to be noticed is that the accessing mode for this function is `Private`, which means that all other event procedures defined in this console application can call and use this function.
- F.** 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 `Order By` operator.
- G.** 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.

```

Module SortLinesLINQ
    Sub Main()
        'Create an IEnumerable data source
        A Dim scores As String() = System.IO.File.ReadAllLines(".././../scores.csv")

        'Change this to any value from 0 to 4
        B Dim sortField As Integer = 1
        Console.WriteLine("Sorted highest to lowest by field " & sortField)

        'The query is executed here.
        C For Each str As String In RunQuery(scores, sortField)
            Console.WriteLine(str)
        Next

        'Keep console window open in debug mode.
        D Console.WriteLine("Press any key to exit.")
        Console.ReadKey()

    End Sub

    E Private Function RunQuery(ByVal source As IEnumerable(Of String), ByVal num As Integer) As IEnumerable(Of String)
        F Dim scoreQuery = From line In source
            Let fields = line.Split(New Char() {","})
            Order By fields(num) Descending
            Select line

        G Return scoreQuery
    End Function
End Module

```

Figure 4.26. The codes for the example project SortLinesLINQ.

```

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
Press any key to exit.

```

Figure 4.27. The running result of the project SortLinesLINQ.

A complete Visual Basic.NET Console project named **SortLinesLINQ** can be found in the folder **DBProjects\Chapter 4** that is located at the Wiley ftp site (refer to Fig. 1.2 in 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 system are read-only. They are not 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 are existed 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 the limitation of the 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 the Contents of Files in a Folder

This example shows how to query over 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.

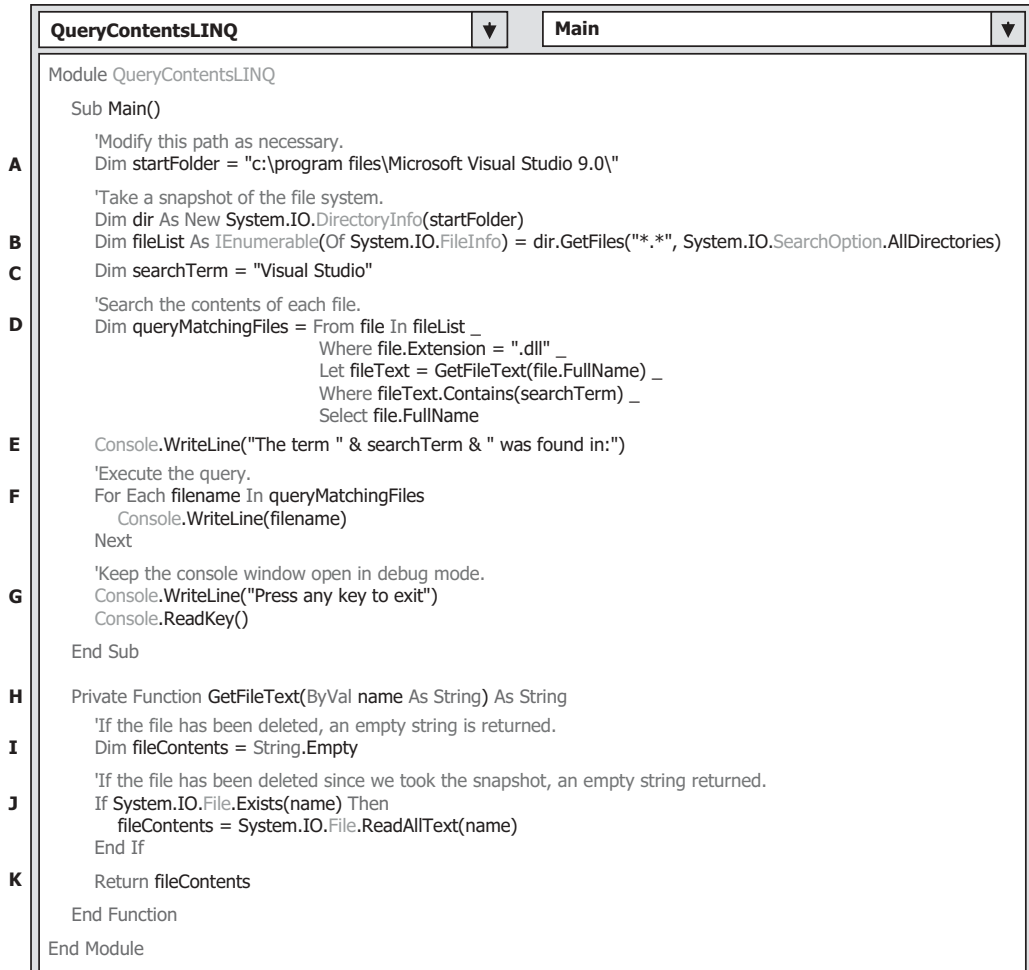


Figure 4.28. The codes for the example project QueryContentsLINQ.

Create a new Visual Basic.NET Console project named **QueryContentsLINQ**, and then open the code window of this new project and enter the codes that are shown in Figure 4.28 into the code window of this project.

Let's have a closer look at this piece of codes 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 2010, in which all files of the Visual Studio.NET 2010 are installed. You can modify this path if you installed your Visual Studio.NET 2010 at a different folder in your computer.
- B.** An **IEnumerable(Of 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 criterion *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 `.dll` 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 function `GetFileText()` starts from here. The point is that this method must be defined as a private function to indicate that this function can be called by all other procedures defined in this console application.
- 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 the 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.

Now you can build and run the project by clicking the *Debug|Start Debugging* menu item. All files that have an extension of `.dll`, 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 from the keyboard to exit this project.

A complete Visual Basic.NET Console project named `QueryContentsLINQ` can be found in the folder `DBProjects\Chapter 4` that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1). 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 4.0 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 will 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 Visual Basic.NET console project and name it as `QueryReflectionLINQ`. Open the code window of this new project and enter the codes that are shown in Figure 4.29 into this window.

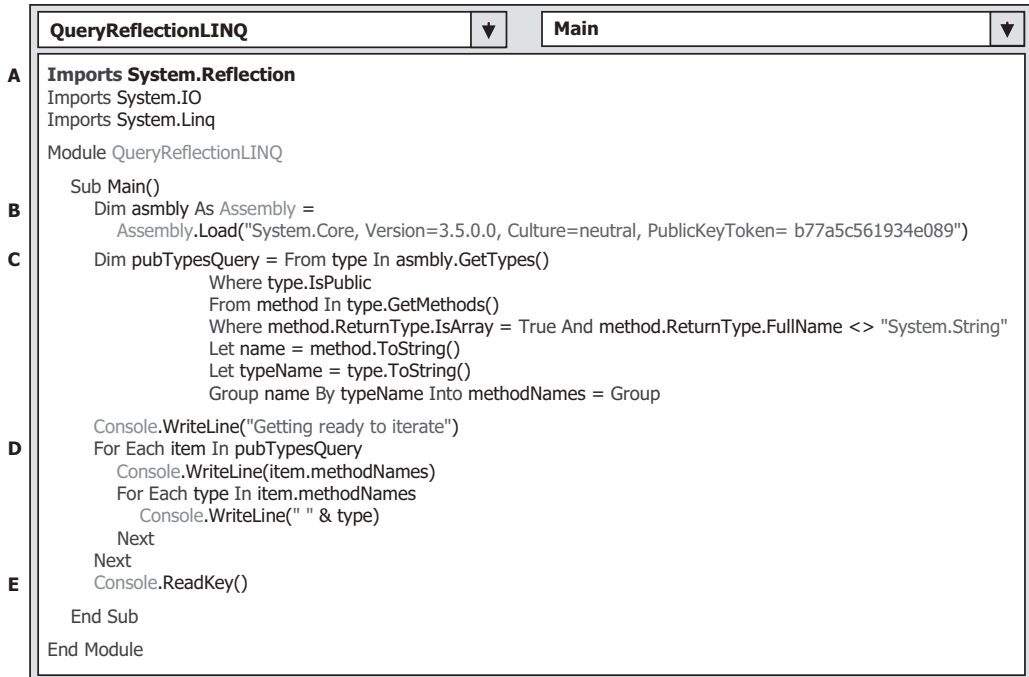


Figure 4.29. The codes for the example project QueryReflectionLINQ.

Let's have a closer look at this piece of codes 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 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 two criteria: (1) the returning type of the method is array, and (2) the returning type of those methods should not be `System.String`. Also, the queried methods' names are converted to string.
- D.** Two `For Each` 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 this coding line 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 Visual Basic.NET Console project named `QueryReflectionLINQ` can be found in the folder `DBProjects\Chapter 4` that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

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 that ADO.NET is built on. 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 process to understand it.

Among all LINQ to DataSet query operations, the following three are the 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 have a little deeper understanding about the LINQ to DataSet, or the operations to the DataSet objects.

4.5.1 Operations to DataSet Objects

Data sources that implement the `IEnumerable(Of T)` generic interface can be queried through LINQ using the Standard Query Operator (SQO) methods. Using `AsEnumerable` SQO to query a `DataTable` returns an object that implements the generic `IEnumerable(Of 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 over in a `For Each` 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 to. 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 to work with. You can also use the `ToList(Of TSource)` or `ToArray(Of 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 Language-Integrated Query (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 to 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 Standard Query Operator 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 query 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 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**, **Order By**, **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 Standard Query Operator methods, such as **From**, **Where**, **Order By**, **Join**, **Let**, and **Select**, as the queries are compiled. Refer to Table 4.1 in this chapter 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 **For Each** loop or a direct call to its method `IEnumerator.MoveNext`.

The following code example 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 contain any query result. This query is composed of four clauses: **From**, **Where**, **Order By**, and **Select**. Both the first and the last clause are required, and the others are optional. The query is casted to a type of `IEnumerable(Of Int32)` by using an `IEnumerable(Of T)` interface. The `testScore` is an iteration variable that is scanned through the **For Each** loop to get and display each queried data when this query is executed. When the **For Each** 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(Of T)` interface, and this makes sure that the type of returned collection is `IEnumerable(Of 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 `scoreQuery` 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 that is shown in Figure 4.31 by using the implicit typing of query variable.

Here, the implicit type is used to replace the explicit type `IEnumerable(Of T)` for the query variable, and it can be converted to the `IEnumerable(Of Int32)` automatically as this piece of codes is compiled.

```
Module QueryExpression
    Dim scores As Integer() = {90, 71, 82, 93, 75, 82}

    Sub main()
        'Query Expression.
        Dim scoreQuery As IEnumerable(Of Int32) = From score In scores
                                                Where score > 80
                                                Order By score Descending
                                                Select score

        'Execute the query to produce the results
        For Each testScore In scoreQuery
            Console.WriteLine("{0, 1} ", testScore)
        Next
        Console.WriteLine(vbNewLine & "Press any key to continue...")
        Console.ReadKey()
    End Sub
End Module

'Outputs: 93 90 82 82
```

Figure 4.30. The example codes for the query expression syntax.

```

Module QueryExpression
    Dim scores As Integer() = {90, 71, 82, 93, 75, 82}

    Sub main()
        'Query Expression.
        Dim scoreQuery = From score In scores
                          Where score > 80
                          Order By score Descending
                          Select score

        'Execute the query to produce the results
        For Each testScore In scoreQuery
            Console.WriteLine("{0, 1} ", testScore)
        Next
        Console.WriteLine(vbNewLine & "Press any key to continue...")
        Console.ReadKey()
    End Sub
End Module

'Outputs: 93 90 82 82

```

Figure 4.31. The example codes for the query expression in implicit typing of query variable.

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. 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 Standard Query Operators (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 have 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 in 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 a quite few of examples using the method syntax in Sections 4.1.3 and 4.1.4. Refer to those sections to get a clear picture in 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 formats in using the query syntax and the method syntax for a given data source.

Create a new Visual Basic console project **QueryMethodSyntax**. Open the code window of this new project and enter the codes that are shown in Figure 4.32 into this code window.

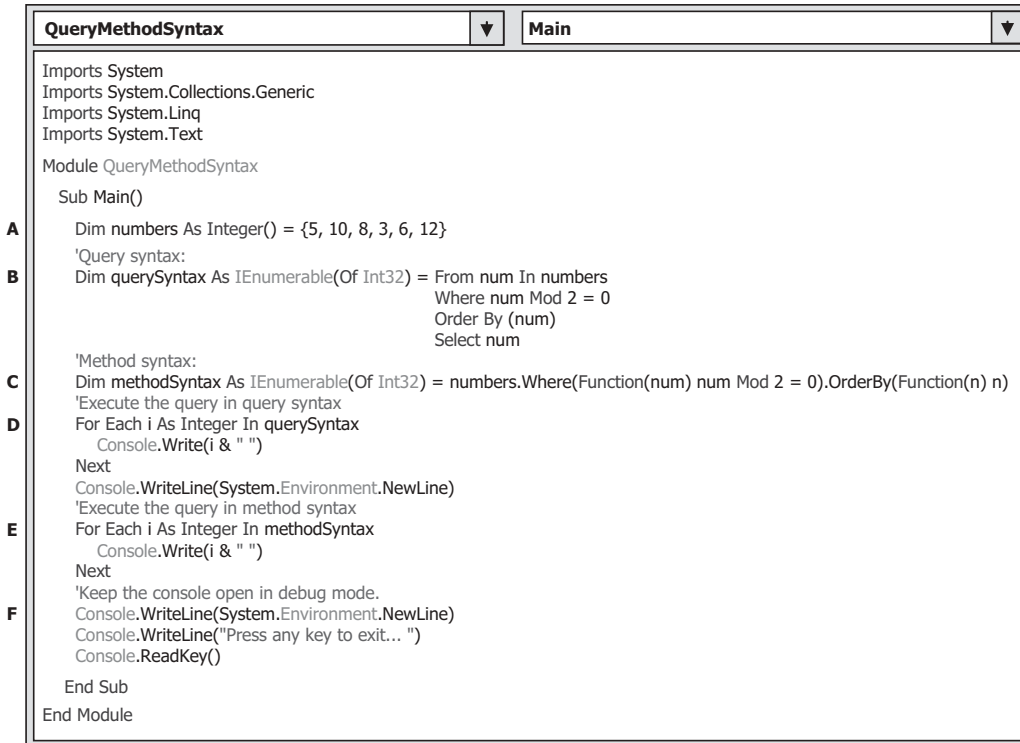


Figure 4.32. The codes for the example project QueryMethodSyntax.

Let's have a closer look at this piece of codes 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(Of Int32)`.
- C.** The second query that uses a method syntax is created and initialized with the Standard Query Operator methods `Where()` and `Order By()`.
- D.** The first query is executed using a `For Each` 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 codes. 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 Visual Basic.NET Console project **QueryMethodSyntax** can be found in the folder **DBProjects\Chapter 4** that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

Besides the general and special properties of query expression discussed above, the following points are also important to understand 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 **For Each** loop.
5. At compile time, query expressions are converted to Standard Query Operator method calls according to the rules set forth in the Visual Basic 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 that the query is applied to. **IEnumerable(Of T)** queries are compiled to delegates. **IQueryable** and **IQueryable(Of 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

LINQ queries work on data sources that implement the **IEnumerable(Of 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 that is 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 in querying a single **DataTable** using the LINQ to **DataSet**. Create a new Visual Basic.NET console project **DataSetSingleTableLINQ**. On the opened project, change the File Name property to **DataSetSingleTableLINQ.vb**. Open the code window of this new project and enter the codes that are shown in Figure 4.33.

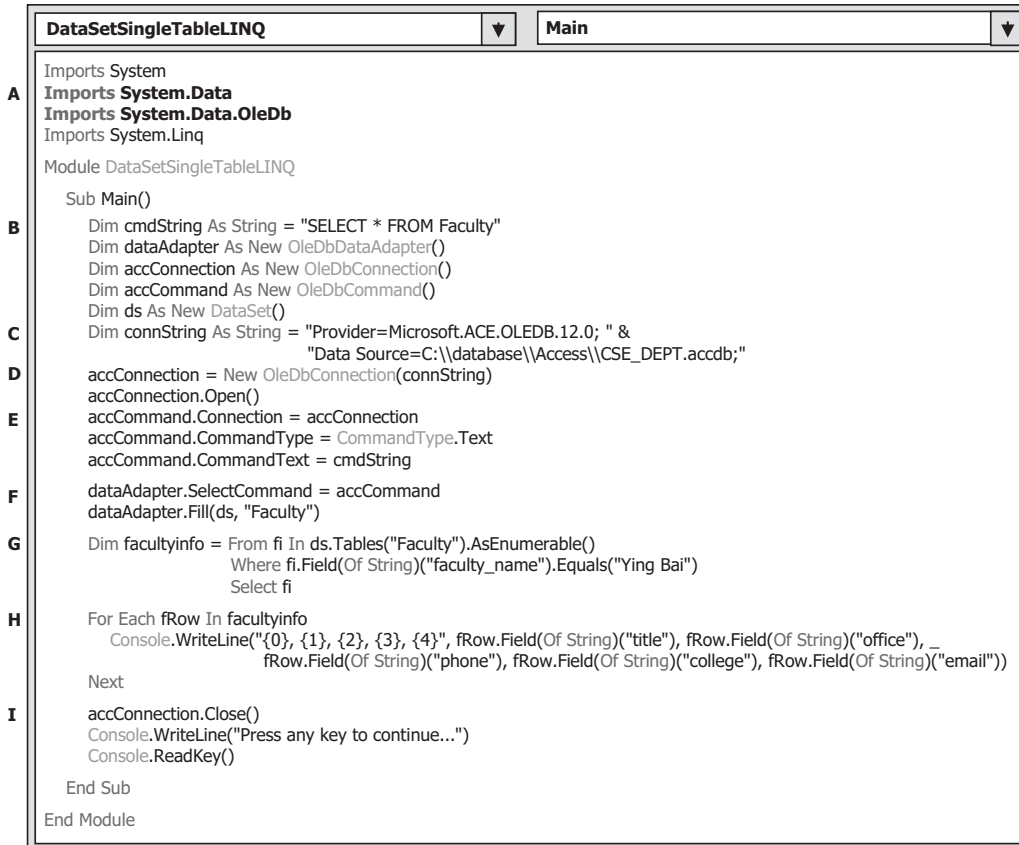


Figure 4.33. The codes for the example project DataSetSingleTableLINQ.

Let's have a closer look at this piece of codes 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 that was developed in Microsoft Access 2007. You need to modify this string based on the real location where 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 trying 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(Of 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 Standard Query Operator methods that will be executed to perform and complete this query.
- H.** The **For Each** loop then enumerates the enumerable object returned by **selecting** and yielding the query results. Because query is an `Enumerable` type, which implements `IEnumerable(Of T)`, the evaluation of the query is deferred until the query variable is iterated over using the **For Each** 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 Debugging`. Related information for the selected faculty will be retrieved and displayed in the console window.

A complete Visual Basic.NET Console project `DataSetSingleTableLINQ` can be found in the folder `DBProjects\Chapter 4` that is located at the Wiley ftp site (refer to Fig. 1.2 in 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 over 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 multi-table query using LINQ to DataSet. The functionality of this project is:

1. Populate a DataSet instance; 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 Visual Basic.NET console project and name it DataSetCrossTableLINQ. On the opened project, change the File Name property to DataSetCrossTableLINQ.vb. Open the code window and enter the codes that are shown in Figure 4.34 into this window.

Let's have a closer look at this piece of codes 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` that was 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.
- 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. A LINQ to DataSet query is created with a Join clause. The data type of the query variable `courseinfo` is an 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(Of T)` type. Two identical fields, `faculty_id`, i.e., 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 Standard Query Operator methods that will be executed to perform and complete this query.

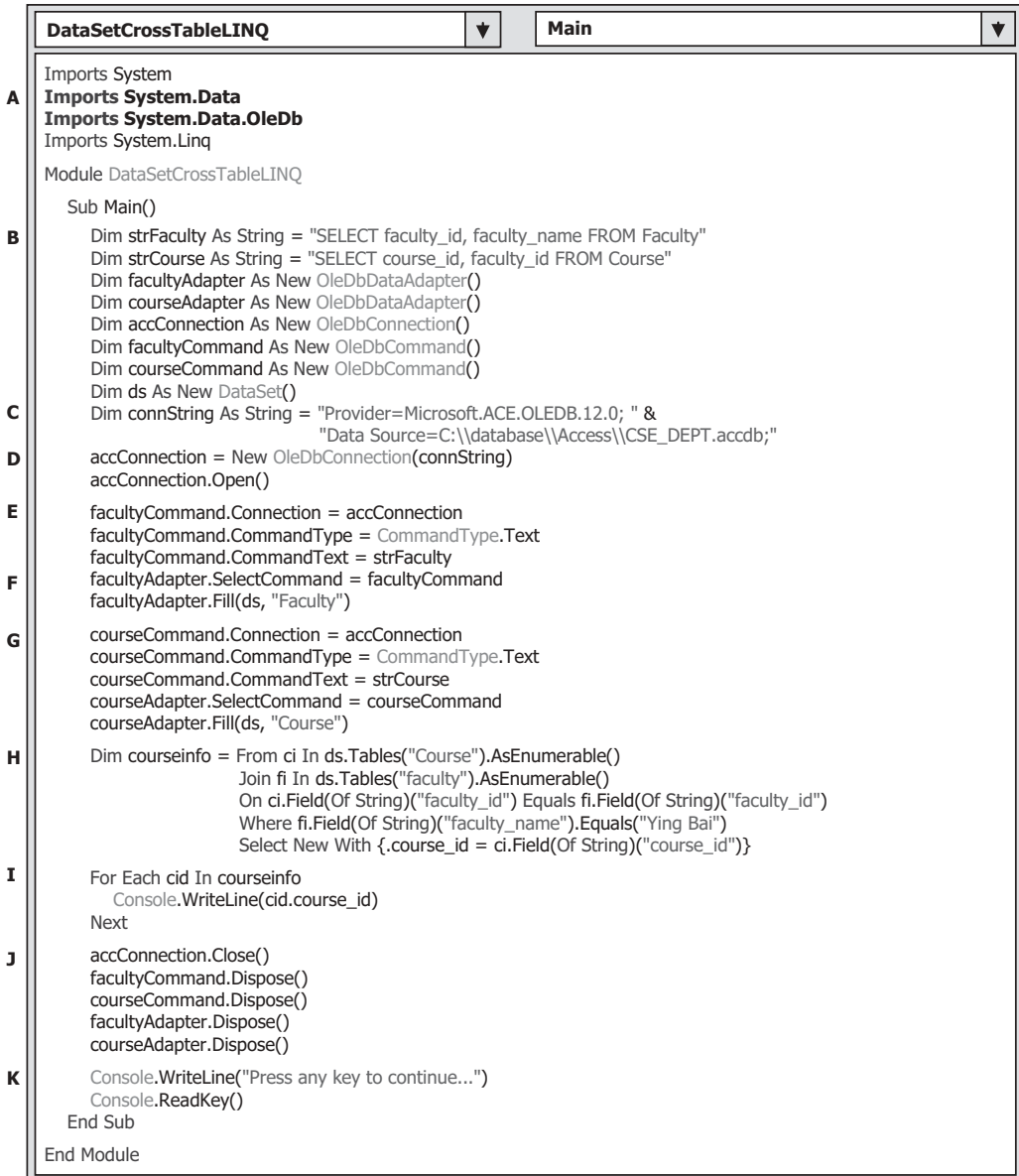


Figure 4.34. The codes for the example project DataSetCrossTableLINQ.

- I.** The **For Each** loop then enumerates the enumerable object returned by **selecting** and yielding the query results. Because query is an `Enumerable` type, which implements `IEnumerable(Of T)`, the evaluation of the query is deferred until the query variable is iterated over using the **For Each** 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 **For Each** loop is done.

- J.** Finally, the connection to our sample database is closed by calling the `Close()` method, and all data components used in this project are released.
- K.** These two coding lines are used to enable this console project to be run in the Debugging mode.

Now you can build and run this project. One point to be noticed 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.accdb` in a different folder.

Click the `Debug|Start 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 Visual Basic.NET Console project `DataSetCrossTableLINQ` can be found in the folder `DBProjects\Chapter 4` that is located at the Wiley ftp site (refer to Fig. 1.2 in 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 derives 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 2010.

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 2010.

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 Visual Basic.NET console project `TypedDataSetLINQ` and change the File Name property to `TypedDataSetLINQ.vb`.

Let's first create our typed DataSet. On the opened new project, right-click our new project `TypedDataSetLINQ` 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.

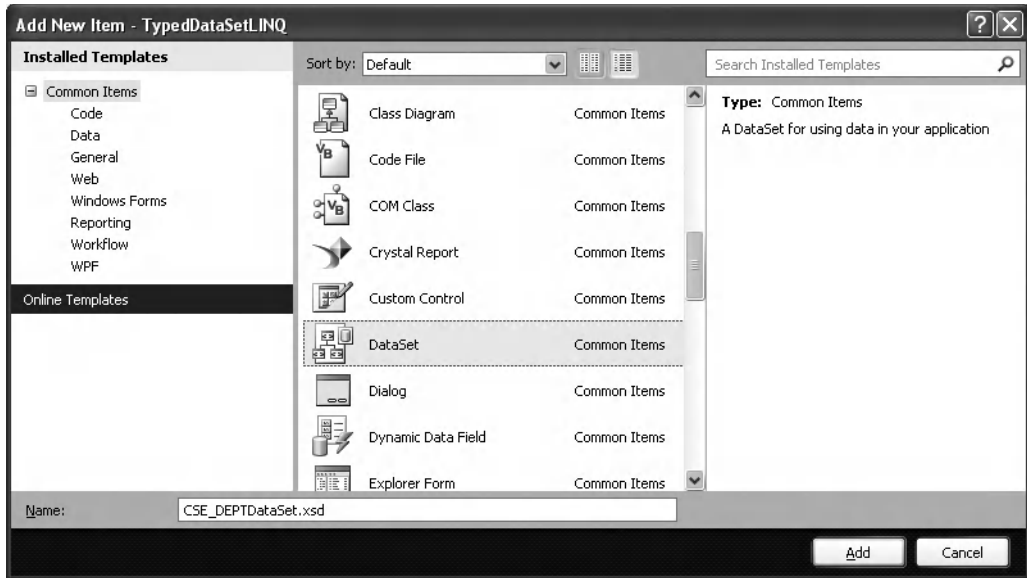


Figure 4.35. The opened Add New Item dialog box.

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 that is 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 the first folder **Data Connections** if you have not connected any data source. Then click the **Add Connection** item from the popup 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 on 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.

Now you can find that a new data connection folder has been added into the Server Explorer window with our sample database `CSE_DEPT.accdb`. If you expand the **Tables** folder under this data source, you can find all five tables, which is shown in Figure 4.36b, in our sample database.

Open the DataSet Designer by double-clicking on the item `CSE_DEPTDataSet.xsd` from the Solution Explorer window if it is not opened, and then drag the **Faculty** and the **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

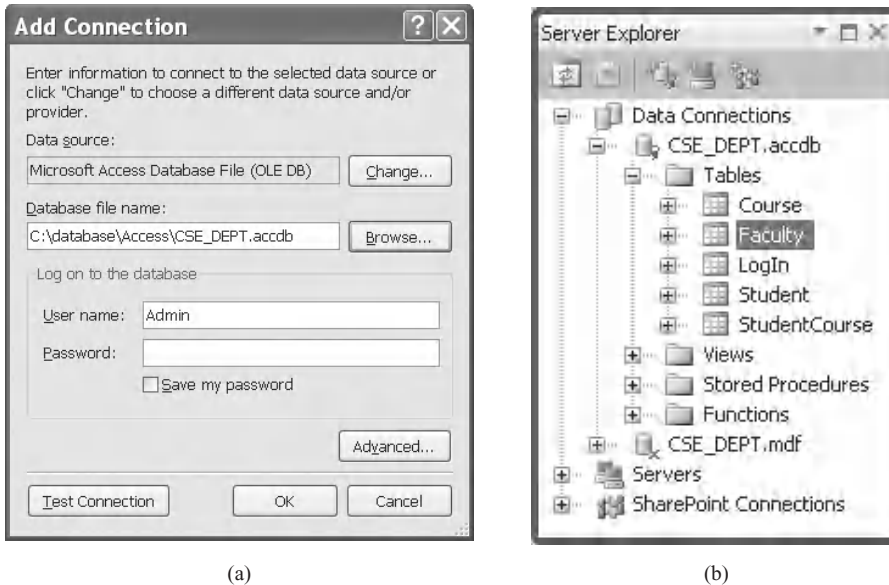


Figure 4.36. The Add Connection dialog and the Server Explorer window.

of them. 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 our new project **TypedDataSetLINQ.vb** from the Solution Explorer window to open the code window of this project. Enter the codes that are shown in Figure 4.37 into this window.

Let's have a closer look at this piece of codes 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.
- D.** The new instance of DataSet is populated with data using the **Fill()** method. 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, 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 worrying about the **Field(Of T)** setup with the real table name.

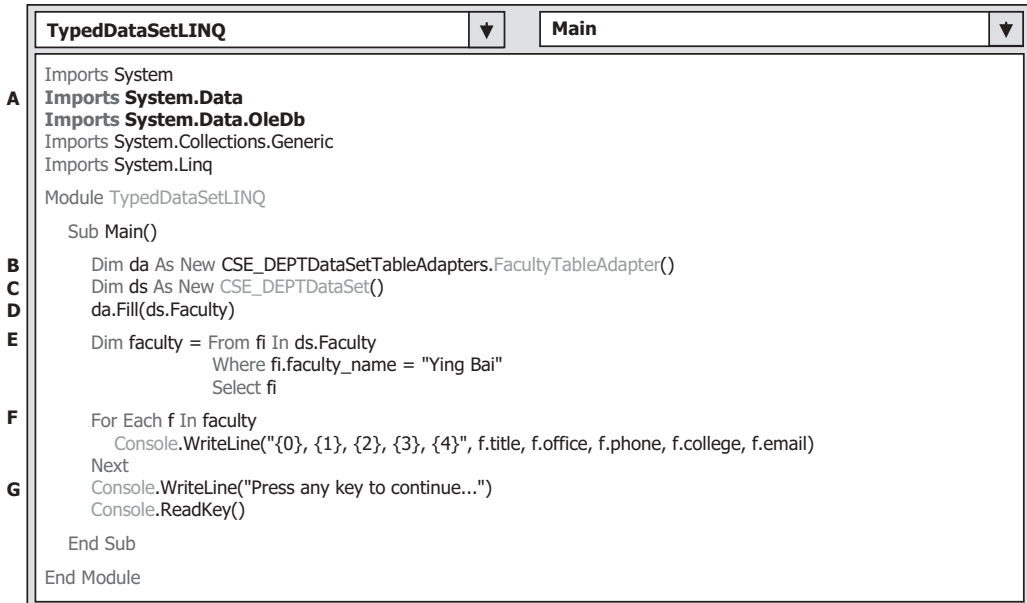


Figure 4.37. The codes for the example project TypedDataSetLINQ.

F. The For Each 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 in Section 4.5.1.3, 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.

G. These two coding lines enable this console project to be run in the Debugging mode.

Now you can build and run the project. Click on the Debug|Start Debugging item to run the project, and you can find all pieces of information related to the selected faculty are retrieved and displayed in this console window. Our project is successful!

A complete Visual Basic.NET Console project TypedDataSetLINQ can be found in the folder DBProjects\Chapter 4 that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

4.5.2 Operations to DataRow Objects Using the Extension Methods

The LINQ to DataSet functionality is exposed primarily through the extension methods in the DataRowExtensions and DataTableExtensions classes. In Visual Basic.NET, 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`. In total, 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 Visual Basic.NET console project to illustrate how to use the `Field()` method to retrieve some 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 2010 and create a new Visual Basic.NET console project `DataRowFieldLINQ`. Open the code window of this new project and enter the codes that are shown in Figure 4.38 into this window.

Let's have a closer look at this piece of codes 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 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` that was developed in Microsoft Access 2007. You need to modify this string based on the real location in which you saved your database.
- D. The `Connection` object `accConnection` is initialized with the connection string, and a connection is executed by calling the `Open()` method. A `Try . . . Catch` block should be used regularly for this connection operation to catch up any possible exception. Here, we skip it, since we are trying 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 (`Of DataRow`).
- 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 the six prototypes if you like to

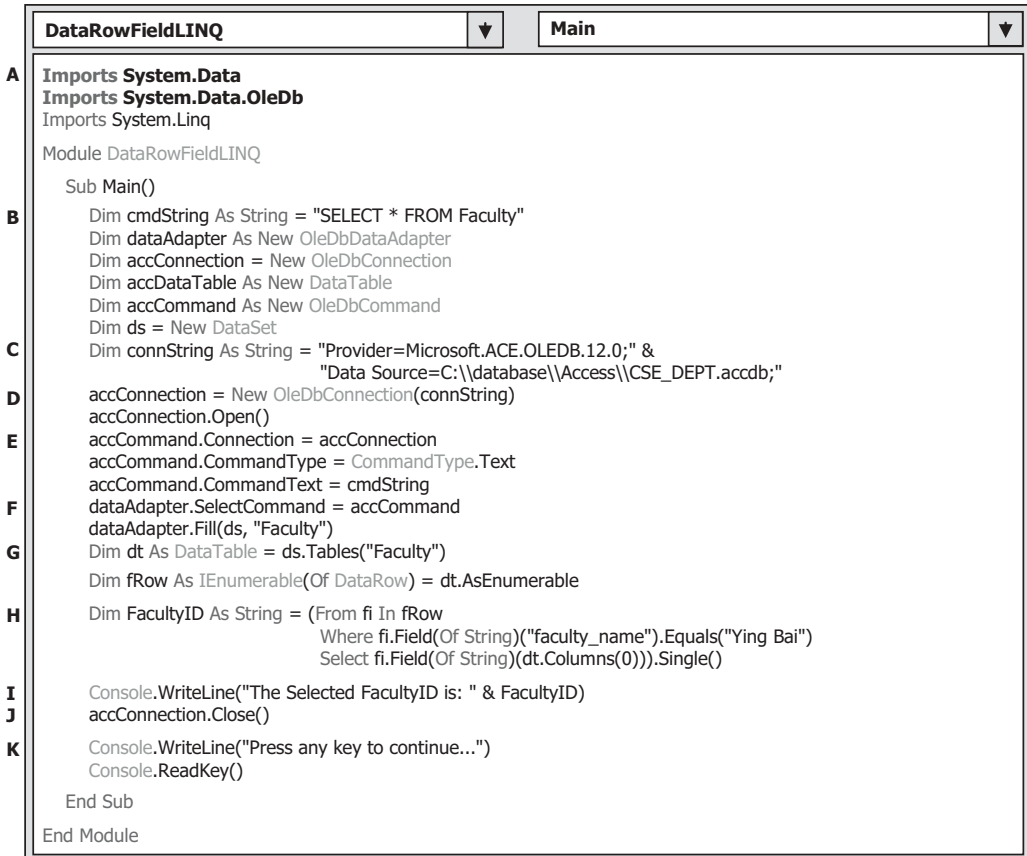


Figure 4.38. The codes for the example project DataRowFieldLINQ.

replace this one. The Standard Query Operator 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.
- K.** These two coding lines enable this console project to be run in the Debugging mode.

Now you can build and run this project to test the functionality of querying a single column from a `DataRow` object. Click the `Debug!Start Debugging` menu item to run the project. The desired `faculty_id` will be obtained and displayed in this console window.

A complete Visual Basic.NET Console project `DataRowFieldLINQ` can be found in the folder `DBProjects\Chapter 4` that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

Before we can finished 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 2010 and create a new Visual Basic.NET Console project and name it as `DataRowSetFieldLINQ`. Change the File Name property to

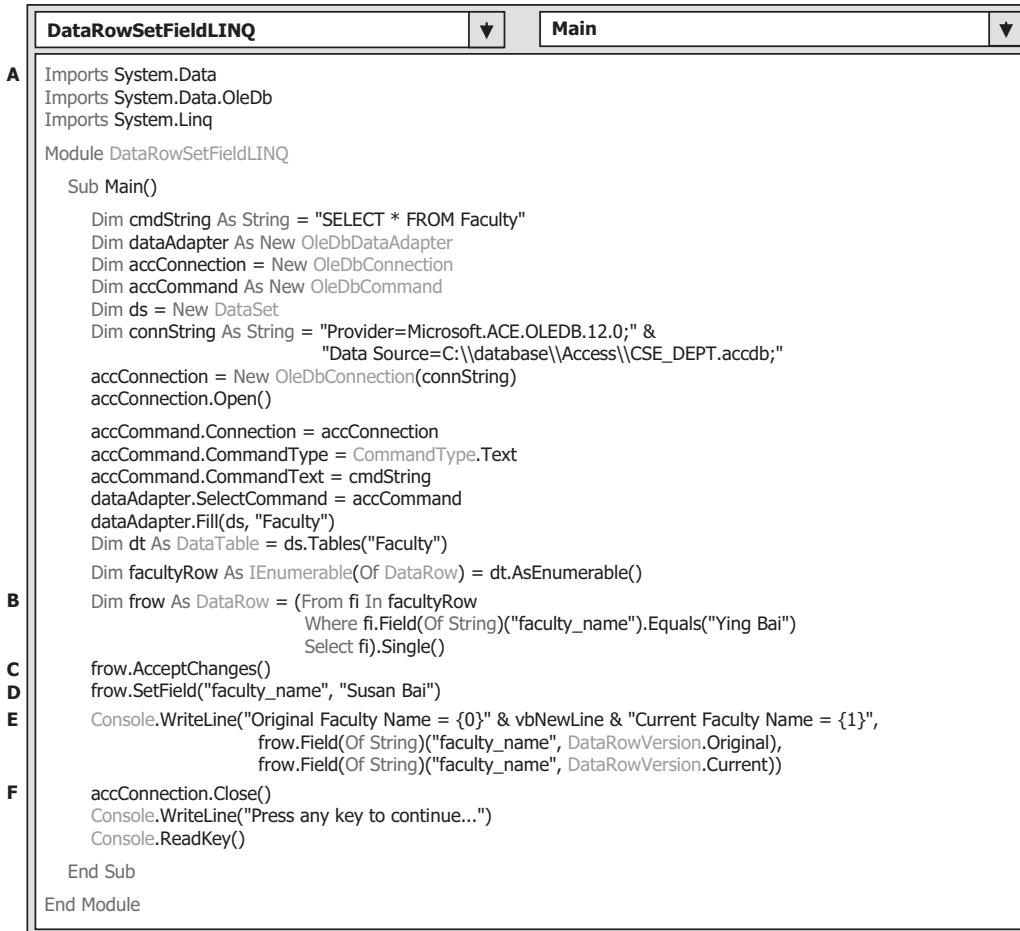


Figure 4.39. The codes for the example project DataRowSetFieldLINQ.

DataRowSetFieldLINQ.vb. Open the code window of this project and enter the codes that are 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 codes 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 other two prototypes if you like.
- E. The `Console.WriteLine()` method is executed to display both 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 the `Debug|Start 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 Visual Basic.NET Console project `DataRowSetFieldLINQ` can be found in the folder `DBProjects\Chapter 4` that is located at the Wiley ftp site (refer to Fig. 1.2 in 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 Visual Basic, 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(Of T)` types. To use the standard query operators, first bring them into scope with an `Imports System.Linq` directive. Then any type that implements `IEnumerable(Of 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(Of T)` type, such as `List(Of 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 give a brief discussion about the first method in this section.

The functionality of the extension method `AsEnumerable()` is to convert and return a sequence of type `IEnumerable(Of 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 about 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(Of T) interface. Since the IQueryable(Of T) interface is inherited from the IEnumerable(Of T) with additional components, therefore, besides the Standard Query Operator (SQO), the LINQ to SQL queries have additional query operators, since it uses the IQueryable(Of 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 Standard Query Operators (SQOs) 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 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. Only after this mapping relationship has been setup, 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 is the guy who will work 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 provide 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.

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 command into a black-and-white window. The Object Relational Designer provides a 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.

We will process this section with 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 discuss how to create entity classes and add the DataContext object to connect to our sample SQL Server database CSE_DEPT.mdf using a real project **SQLSelectRTOBJECTLINQ**, which is a blank project with five form windows and located at the folder DBProjects\Chapter 5 in the Wiley ftp site (refer to Fig. 1.2 in Chapter 1), 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 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

Open the blank project **SQLSelectRTOBJECTLINQ** and 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. Add a reference to our project
2. 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 click the OK button to add this reference to our project.

Now open the code window of the LogIn form and add the **Imports System.Data.Linq** to the namespace declaration section in this code window.

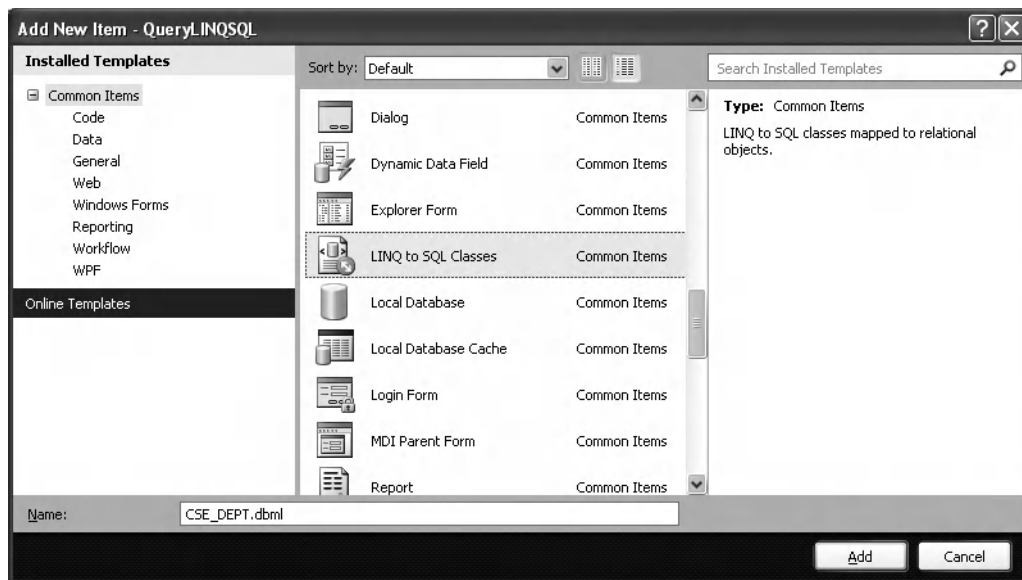


Figure 4.40. The Add New item dialog.

Next, we need to create entity classes to setup 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 our project `SQLSelectRTObjctLINQ` from the Solution Explorer window, and select the item `AddNew Item` from the popup menu to open the Add New Item dialog. On the opened dialog, select the item `LINQ to SQL Classes` by clicking 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.

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, and it gives you flexibility or controllability on entity classes. You can use any meaningful name for this DBML file, but regularly, 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.vb`, are attached under that folder. The first file is exactly the designer that is shown as a blank window in Figure 4.41, and the second file is auto-generated 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.

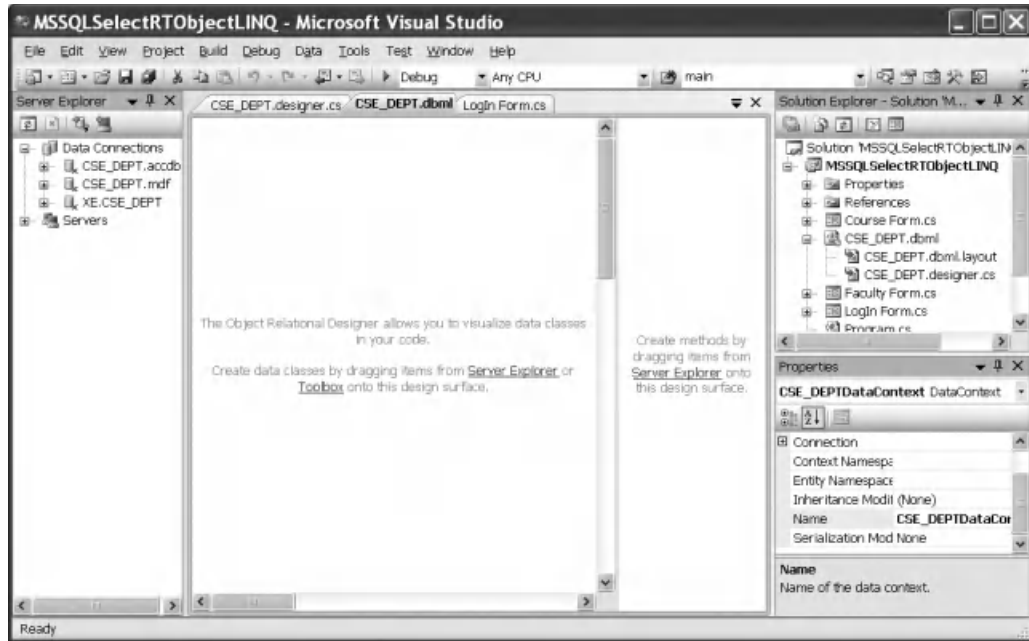


Figure 4.41. The opened Object Relational Designer.

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 from 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. The following three points should be noted when you perform this database connection:

1. Change the **User Instance** property to **False** if you reinstall the Microsoft SQL Server 2008 after removing an old version of Microsoft SQL Server database by using the Advanced setup function (click the **Advanced** button). Otherwise, you do not need to do this modification.
2. Confirm that a SQL Server 2008 Express database file is being used for this connection by clicking on the **Advanced** button, and then going to the **Data Source** item to check this.



Figure 4.42. The Add Connection dialog box.

3. You can find our sample database `CSE_DEPT.mdf` in the folder `Database\SQLServer` that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1). You can copy this database file and save it to any folder at your computer.

Click on the OK button to close this dialog. Your finished Add Connection dialog box should match the 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 under the **Tables** node and you can find all of our 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 to the Design window. Click on the **Yes** button for the first message box to copy the database file to our project, and click on the **No** to the second message box to avoid the duplication of our database. 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.vb` 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 designer should match the one that is shown in Figure 4.43. The arrow between tables is an association that 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.

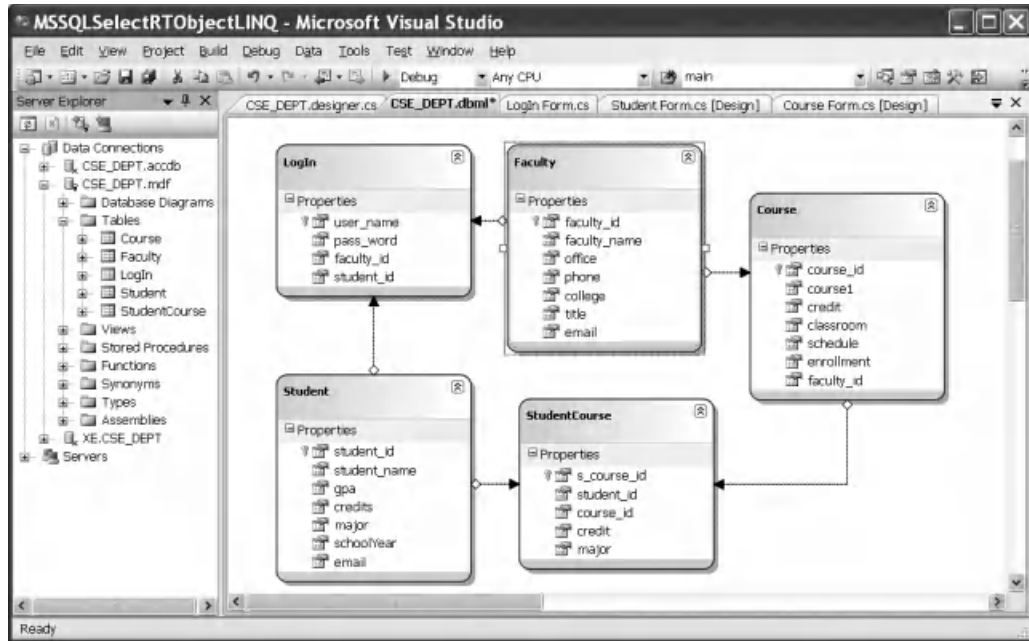


Figure 4.43. The finished Designer.

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 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 the last section to do LINQ to SQL actions against our sample database. Because the blank project `SQLSelectRTOObjectLINQ` will be used in Chapter 5, we had better create another Visual Basic.NET 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 Visual Basic.NET console project `QueryLINQSQL`. On the opened new project, change the File Name property to `QueryLINQSQL.vb`. Then perform the following operations to prepare our LINQ to SQL queries:

1. Add the `System.Data.Linq` reference to this new project by right-clicking on our new project `QueryLINQSQL` from the Solution Explorer window, and click on the Add

Reference item from the popup menu to open the Add Reference wizard. Make sure that the .NET tab is selected, scroll down the list, 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 the file `QueryLINQSQL.vb`:
 - Imports System.Data.Linq
 - Imports System.Data.Linq.Mapping
3. Follow 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 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 on the mapping file **CSE_DEPT.dbml** from the Solution Explorer window and select **View Code** item to create a Visual Basic code file for our database, **CSE_DEPT.vb**.

Now open the code window and enter the codes shown in Figure 4.44 into this window.

Let's take a closer look at this piece of codes 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.
- 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 or data action.
- D. The users' answer is received by running a system method `Console.ReadLine()`.
- E. A `Select Case` structure is used to identify the user's selection and direct the program to the associated method to perform the desired query or action.
- 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.

Now we have finished all prerequisite jobs to make a desired LINQ to SQL query, let's start our coding from the first query, data selection.

4.6.2.1 Data Selection Query

Create a new subroutine `LINQSelect()` under the `Main()` subroutine in this console project and enter the codes that are shown in Figure 4.45 into this subroutine.

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

- A. The newly created `DataContext` object is passed into this subroutine since we need to use this object to access our sample database to perform the selection query. The accessing


```

Imports System.Linq
Imports System.Collections.Generic
A Imports System.Data.Linq
Imports System.Data.Linq.Mapping

Module QueryLINQSQL

    Sub Main()
B        Dim cse_dept As New CSE_DEPTDataContext()
C        Console.WriteLine("Make your selection: " & vbNewLine)
        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" & vbNewLine)

D        Dim input As String = Console.ReadLine()
E        Select Case (input)
            Case "1"
                LINQSelect(cse_dept)
            Case "2"
                LINQInsert(cse_dept)
            Case "3"
                LINQUpdate(cse_dept)
            Case "4"
                LINQDelete(cse_dept)
            Case "5"
                Exit Sub
            Case Else
                Console.WriteLine("Invalid input value ")
        End Select
F        Console.WriteLine(vbNewLine & "Press Enter key to continue ....")
        Console.ReadLine()

    End Sub

End Module

```

Figure 4.44. The codes for the console Main method.

```

A Private Sub LINQSelect(ByRef db As CSE_DEPTDataContext)
B        Dim faculty = From fi In db.Faculties
        Where fi.faculty_id = "B78880"
        Select fi
C        For Each f In faculty
            Console.WriteLine("{0}, {1}, {2}, {3}, {4}, {5}", f.faculty_name, f.title, f.office, f.phone, f.college, f.email)
        Next

    End Sub

```

Figure 4.45. The codes for the LINQSelect subroutine.

mode of this subroutine should be Private if it will be called by any other procedures in the current Main subroutine in this console project.

- 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 **For Each** 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. The option 2 is similar to the data updating query; therefore, we concentrate on the first option in this section only.

Create a new subroutine **LINQInsert()** in this project, and enter the codes that are shown in Figure 4.46 into this method.

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

- A.** The new created **DataContext** object is passed into this subroutine since we need to use this object to access our sample database to perform the data insertion action.
- B.** A new **Faculty** entity object is created, and it is equivalent to a **DataRow** object.
- C.** The newly created **Faculty** entity object is initialized with all new columns' values.

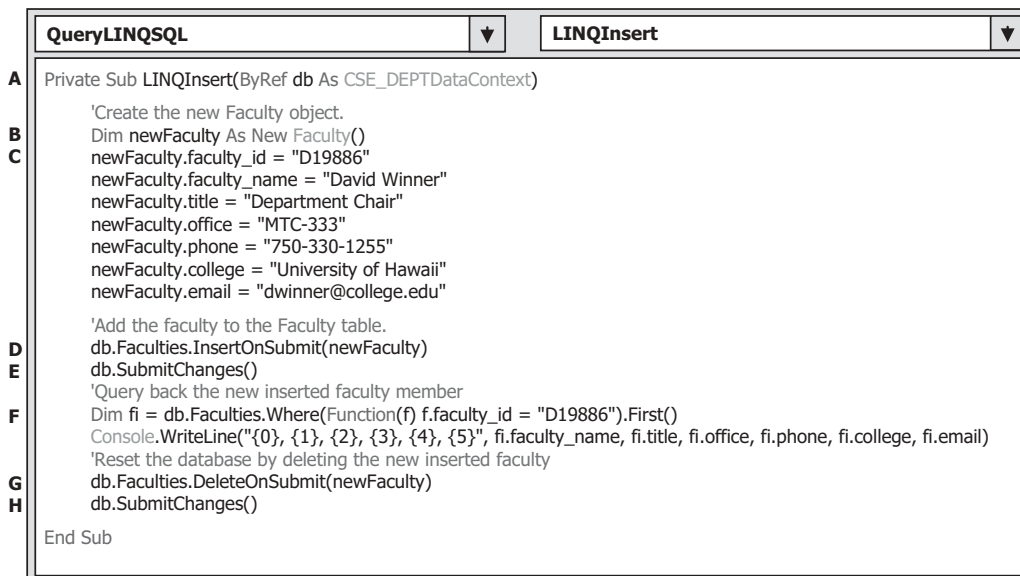


Figure 4.46. The codes for the LINQInsert subroutine.

- D. The `InsertOnSubmit()` method is executed to add this new object into the **Faculty** entity object (Faculty data table). One point to be noticed is that this new record can be 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 codes are used to retrieve and display the newly inserted record to confirm our data insertion operation. Two Standard Query Operator methods, `Where()` and `First()`, are used in this query.
- G. To make our database clean, we need to delete this newly inserted record from our database after this insertion is successful.
- H. The newly 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

To perform a data updating operation, the following operation sequence should be followed regularly:

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. Update or delete those records that are foreign keys but are 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 key from the parent table, such as `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 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 codes show how to update the `faculty_name` column from the `Faculty` table.

Create a new subroutine `LINQUpdate()` in this project and enter the codes that are shown in Figure 4.47 into this subroutine.

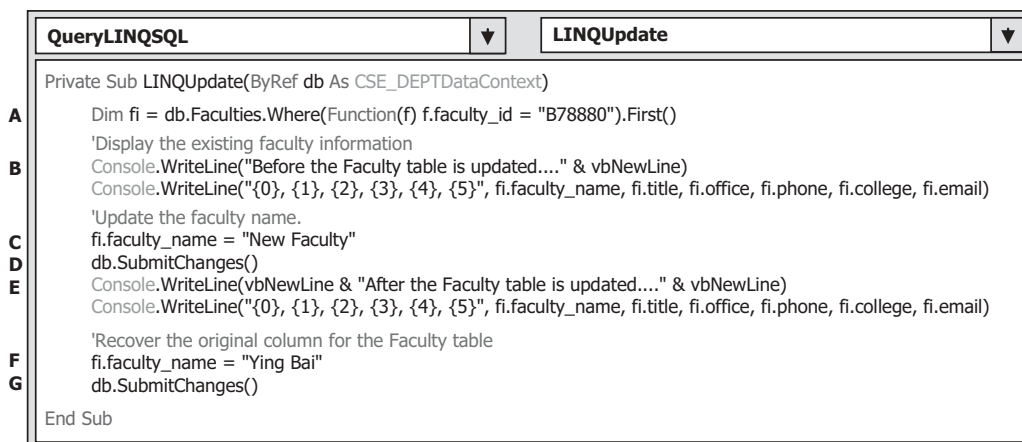


Figure 4.47. The codes for the LINQUpdate subroutine.

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

- A.** A selection query is executed using the Standard Query Operator 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.
- 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 in Chapter 2, 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 is to allow all related records in the child tables to be cascade 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(Of T)` of which it is a member with the `Table(Of T)` object's `DeleteOnSubmit()` method.

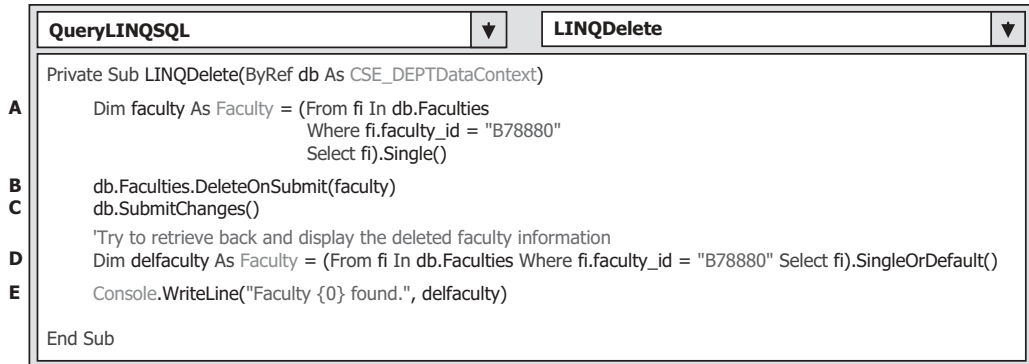


Figure 4.48. The codes for the LINQDelete subroutine.

The example codes in Figure 4.48 show 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 subroutine **LINQDelete()** in this project, and enter the codes shown in Figure 4.48 into this subroutine.

Let's have a closer look at this piece of codes 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 **Standard Query Operator** method **Single()** is used to retrieve only a single row.
- B.** The queried row **faculty** is placed into the deleting pool with the **DeleteOnSubmit()** 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 **Standard Query Operator** 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()**.

Now you can build and run this project to test all LINQ to SQL queries developed in this project. A complete Visual Basic.NET Console project **QueryLINQSQL** can be found in the folder **DBProjects\Chapter 4** that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

One point to be noticed is 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 recover these three tables by adding those deleted records back to those tables as soon as you finished this deletion operation in this project.

As we mentioned, if you performed a deleting action from this project, you need to recover those deleted records for the three tables. Now let's first open those three tables to check what and how many records have been deleted after 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:\Chapter 4\QueryLINQSQL\QueryLINQSQL\bin\Debug. If you did not integrate this database with your project, the database file may 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 our database. Then expand the database to locate all data tables by expanding the **Tables** folder. Right-click a data table and select the **Edit Top 200 Rows** 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, 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 4.3–4.5 into the associated table in Microsoft SQL Server Management Studio Express. The recovery order is:

1. Recover deleted data in the primary table **Faculty** first
2. Recover deleted data in the child tables, such as **LogIn** and **Course**

Close the Microsoft SQL Server Management Studio Express when this recovery job is done.

Table 4.3. Deleted record from the Faculty table

faculty_id	faculty_name	office	phone	college	title	email
B78880	Ying Bai	MT-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

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 in this book, 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 coding developments for those projects.

4.7 LINQ TO ENTITIES

As we mentioned in the introduction to 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 4.0 Entity Framework (ADO.NET 4.0 EF) and ADO.NET 4.0 Entity Framework Tools (ADO.NET 4.0 EFT). ADO.NET 4.0 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 4.0 EF and ADO.NET 4.0 EFT, developers need to understand the EDM that is a core of ADO.NET 4.0 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 on Microsoft .NET Framework 4.0. 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 EDM and the application.

LINQ to Entities queries use the Object Services infrastructure. The `ObjectContext` class is the primary class for interacting with an EDM 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 The 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 CLR objects that are instances of entity types. Object Services supports both LINQ and Entity SQL queries against types defined in an 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 TheObjectContext 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 that persist 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 The 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 that is 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 Visual Basic 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 have a little detailed discussion for each of these steps.

4.7.4.1 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.

4.7.4.2 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 method-based query syntax with real example codes in Sections 4.5.1.1 and 4.5.1.2 in this chapter. Refer to those sections to get a clearer picture for these two syntaxes.

4.7.4.3 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 `Order By`) and expressions. LINQ Standard Query Operators are not defined by a class, but rather are static methods on 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 hierarchy, which are then placed in a command tree. The command tree is used by

```
Dim FacultyInfo As IQueryable(Of String) = From fi In Faculties
                                           Where fi.faculty_id = "B78880"
                                           Select fi.faculty_name
```

Figure 4.49. An example of expression used in LINQ to Entities.

```
Dim faculties As ObjectQuery(Of Faculty) = cse_dept.Faculty
Dim FacultyNames As IQueryable(Of String) = From f In faculties
                                           Select f.faculty_name

Console.WriteLine("Faculty Names:")
For Each fName In FacultyNames
    Console.WriteLine(fName)
Next
```

Figure 4.50. An example of executing the query.

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.

4.7.4.4 *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 over, 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 display the faculty names. Iterating over the query variable in the **For Each** 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.

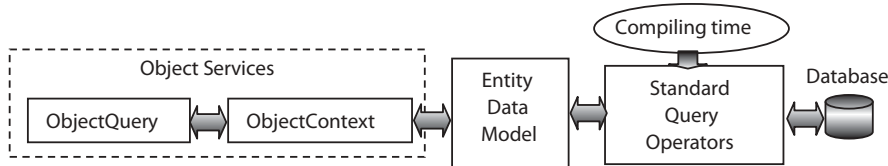


Figure 4.51. A simplified structure of LINQ to Entities.

4.7.4.5 Materialize the Query

Materialization is the process of returning query results back to the client as CLR types. In LINQ to Entities, query results of 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 EDM.
- CLR types supported by the EDM.
- 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 have provided a very detailed discussion about the ADO.NET Entity Framework 4.1 and ADO.NET 4.0 EDM, including the EDM Wizard, EDM Designer, and Entity Model Browser with a real example project **EDModle**. Review that section to get more details for the implementation of LINQ to Entities. A complete example project **EDModel** that uses LINQ to Entities query can be found in the folder **DBProjects\Chapter 5** located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

4.8 LINQ TO XML

LINQ to XML was developed with LINQ 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 LINQ 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 with a modernized, redesigned Document Object Model (DOM) 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 that is shown in Figure 4.52.

The following important points should be noticed 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/

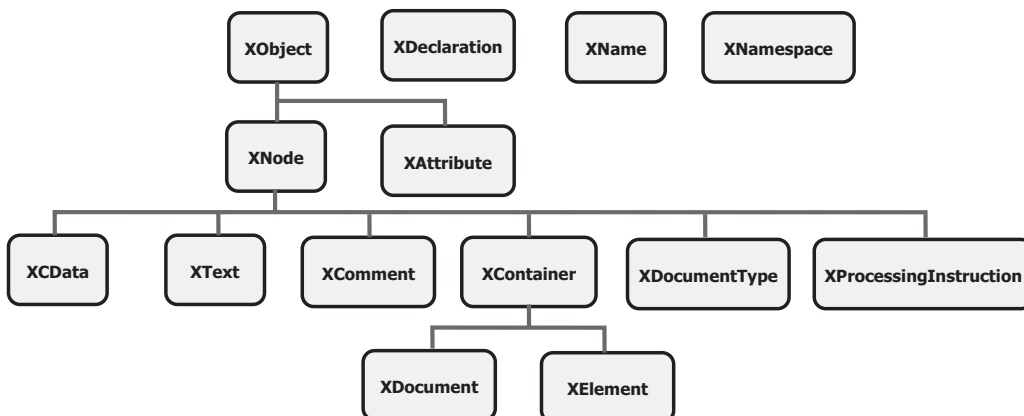


Figure 4.52. The LINQ to XML class hierarchy.

value pairs associated with an XElement. XDocuments are created only if necessary, such as to hold a 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 semi-hidden implementation detail except when exposing text 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 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 illustrate 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.

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)
```

```
Dim faculties As XElement = 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. A sample XML file created using LINQ to XML.

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`, added as a child element.
- An `XAttribute`, 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 (“Patrick Tones”) is passed into the `faculty_name` `XElement` constructor. This could have been a variable (e.g., `new XElement(“faculty_name”, facultyName)`), it could have been a different type besides string (for example, `new XElement(“quantity”, 55)`), and it could have been the result of a function call like one shown in Figure 4.54.

It could also have even been an `IEnumerable(Of 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`.

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.

```

.....
    Dim qty As XElement = New XElement("quantity", GetQuantity())
.....
Public Function GetQuantity() As Integer
    Return 55
End Function

```

Figure 4.54. A sample functional construction.

```

Class Person
    Public faculty_name As String
    Public PhoneNumbers As String()
End Class

Sub Main()
    Dim persons As Person() = {New Person With {.faculty_name = "Patrick Tones", .PhoneNumbers = {"750-555-0144",
"750-555-0145"}}, New Person With {.faculty_name = "Gretchen Rivas", .PhoneNumbers =
{"750-555-0163"}}}

    Dim faculties As XElement = 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)
End Sub

```

Figure 4.55. A sample query using LINQ to XML.

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 the 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 that is 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.

Let's take a look a little deeper 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

```

A Dim mobilePhone As XElement = New XElement("phone", "750-555-0168")
    faculty.Add(mobilePhone)

B     Dim mobilePhone As XElement = New XElement("phone", "750-555-0168")
    Dim firstPhone As XElement = faculty.Element("phone")
    firstPhone.AddAfterSelf(mobilePhone)

C     Dim mobilePhone As XElement = 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 codes of 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 codes of using LINQ to XML to update XML.

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 codes that are 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 codes that are shown in section **A** in Figure 4.57.

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.

A	<code>faculty.Element("phone").Remove()</code>
B	<code>faculty.Elements("phone").Remove()</code>
C	<code>faculties.Element("faculty").Element("office").RemoveNodes()</code>
D	<code>faculty.SetElement("phone", null)</code>

Figure 4.58. Some sample codes of using LINQ to XML to delete XML.

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 that is shown in section **D** in Figure 4.57. Attributes have a symmetric method called `SetAttribute()`, which has 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 following code that is 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 that is 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 that is 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.

4.8.3.1 Add XML Attributes

Adding an XAttribute is very similar to adding a simple XElement. In the sample XML that is 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.

```
Dim Faculty = <faculties>
  <faculty>
    <faculty_name>Patrick Tones</faculty_name>
    <phone type="home">750-555-0144</phone>
    <phone type="work">750-555-0145</phone>
  </faculty>
```

Figure 4.59. A sample XML attributes.

```
Dim faculty As XElement = 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. A sample code to create an XAttribut.

```
A For Each p In faculty.Elements("phone")
    If p.Attribute("type") = "home" Then
        Console.WriteLine("Home phone is: " & p.ToString)
    End If
Next

B faculty.Elements("phone").First().Attribute("type").Remove()

C faculty.Elements("phone").First().SetAttributeValue("type", DBNull.Value)
```

Figure 4.61. A sample code to get and delete an XAttribut.

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 that are 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 codes that are shown in section **A** in Figure 4.61.

Notice that how the Attribute(XName) works similarly to the Element(XName) method. Also, notice that there are some differences between the Attribute() and the SetAttributeValue() methods.

4.8.3.3 Delete XML Attributes

If you want to delete an attribute, you can use the `Remove()` or `SetAttributeValue(XName, Object.Value)` 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 that is shown in section **B** in Figure 4.61.

Alternatively, you can use the `SetAttributeValue()` method with a `DBNull.Value` 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 deep 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 LINQ. 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(Of T)`. Standard query operators show up as extension methods on any object that implements `IEnumerable(Of 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 are query expressions for 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 LINQ, 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

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 in this chapter.

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 codes that are 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 codes that are shown in Figure 4.63. Notice that we used

```
Dim faculties = <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. A sample code to create an `XElement`.

```
Dim f As New XElement("Faculties",
    From c In faculties.Elements("faculty")
    Select New Object() _
    {
        New XComment("faculty"),
        New XElement("faculty_name", c.Element("faculty_name")), c.Elements("phone"),
        New XElement("office", c.Element("office"))
    })
```

Figure 4.63. A sample code to perform the query to an `XElement`.

an array initializer to create the sequence of children that will be placed directly under the `faculty` 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 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(Of XElement)`). For example, to get the child elements for every faculty in the faculty list, you could do the following:

```
For Each fi As XElement In faculties.Elements("Faculties").Elements()
    Console.WriteLine(fi)
Next
```

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(Of XObject)` containing the child elements in the single `XElement` `faculties`. The second `Elements()` method is defined as an extension method on `IEnumerable(Of 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:

```
For Each pi As XElement In faculties.Elements("Faculties").Elements("phone")
    Console.WriteLine(pi)
Next
```

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 in 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(Of 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(Of XElement)`.

Another example shown in section **C** in Figure 4.64 retrieving the students that have a `gpa` that is greater than the average `gpa`.

```

A Dim query = From fi In faculties.Elements("faculty")
           Where fi.Element("office") = "MTC-3.*"
           Order By f.Element("faculty_name")
           Select fi.Element("faculty_name")

B Dim query = From fi In faculties.Elements("faculty"), p In fi.Elements("phone")
           Where fi.Element("faculty_id") = "B.*" & p.Value.StartsWith("750")
           Order By fi.Element("faculty_name")
           Select fi

C Dim query = From s In students.Elements("student"), average In students.Elements("student").
           Average(Function(x As Integer) x.Element("gpa"))
           Where (s.Element("gpa") > average)
           Select s

```

Figure 4.64. A sample code to perform the query using query expressions with XML.

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. A point to be noticed is 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: Imports `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:

```

For Each phone In faculties.XPathSelectElements("//phone")
    Console.WriteLine(phone)
Next

```

Likewise, XSLT is enabled by referencing the `System.Xml.Xsl` namespace by typing: Imports `System.Xml.Xsl` in the namespace declaration section on the code window of each project. That allows you to create an `XPathNavigator` using the `XDocumentCreateNavigator()` 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 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.

4.8.4.5.1 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.

4.8.4.5.2 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 members 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.

```
Dim 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(faculties)
```

Figure 4.65. A 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. A sample XML.

```
For Each fi In facultyUpdates.Elements("facultyUpdate")
    Dim faculty As Faculty = db.Faculties.
    First(Function(f) f.faculty_id = fi.Element("faculty_id"))
    faculty.Phone = fi.Element("phone")
Next

db.SubmitChanges()
```

Figure 4.67. A piece of sample codes to read and update database.

At this point, we have finished the discussion about the LINQ to XML. Next, we will have a closer look at the Visual Basic.NET language enhancement for LINQ.

4.9 VISUAL BASIC.NET LANGUAGE ENHANCEMENT FOR LINQ

Visual Basic.NET introduces several language extensions to support the creation and use of higher order, functional-style class libraries. The extensions enable construction of compositional APIs that have equal expressive power of query languages in domains, such as relational databases and XML.

Starting from Visual Basic.NET 2008, significant enhancements have been added into Visual Basic.NET, and these enhancements are mainly developed to support LINQ. LINQ is a series of language extensions that supports data querying in a type-safe way; it was released with the 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 Visual Basic.NET. The main components of these improvements include:

- Lambda expressions
- Extension methods
- Implicitly typed local variables
- Query expressions

Let's have a detailed discussion for 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. 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 fact, a lambda expression is a function or subroutine without a name that can be used wherever a delegate is valid. Lambda expressions can be functions or subroutines and can be single-line or multiline. You create lambda expressions by using the **Function** or **Sub** keyword, just as you create a standard function or subroutine. However, lambda expressions are included in a statement.

You can pass values from the current scope to a lambda expression. Unlike named functions, a lambda expression can be defined and executed at the same time. 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 in Visual Basic.NET can be expressed as a function or a subroutine declaration, followed by an expression that can be considered as the function or subroutine body. For more complicated lambda expressions, a statement block

can be followed and embedded. A simple example of lambda expression used in Visual Basic.NET looks like:

```
Dim var = Fucntion(Argument list...) function body or expression
```

where **var** on the left side of the function is the returned running result of the function. The **Argument list** contains all inputs to the function. The function body is a simple expression in most situations. This lambda expression can be read as *input Argument and output var*.

For more complicated lambda expressions, a statement block should be adopted.

The syntax of a lambda expression resembles that of a standard function or subroutine. The differences are:

- A lambda expression does not have a name.
- Lambda expressions cannot have modifiers, such as **Overloads** or **Overrides**.
- Single-line lambda functions do not use an **As** clause to designate the return type. Instead, the type is inferred from the value that the body of the lambda expression evaluates to. For example, if the body of the lambda expression is `f.office = "MTC-332"`, its return type is **Boolean**.
- In multiline lambda functions, you can either specify a return type by using an **As** clause, or omit the **As** clause so that the return type is inferred. When the **As** clause is omitted for a multi-line lambda function, the return type is inferred to be the dominant type from all the **Return** statements in the multi-line lambda function. The dominant type is a unique type that all other types supplied to the **Return** statement can widen to. If this unique type cannot be determined, the dominant type is the unique type that all other types supplied to the **Return** statement can narrow to. If neither of these unique types can be determined, the dominant type is **Object**. For example, if the expressions supplied to the **Return** statement contain values of type **Integer**, **Long**, and **Double**, the resulting type is **Double**. Both **Integer** and **Long** widen to **Double** and only **Double**. Therefore, **Double** is the dominant type.
- The body of a single-line function must be an expression that returns a value, not a statement. There is no **Return** statement for single-line functions. The value returned by the single-line function is the value of the expression in the body of the function.
- The body of a single-line subroutine must be a single-line statement.
- Single-line functions and subroutines do not include an **End Function** or **End Sub** statement.
- You can specify the data type of a lambda expression parameter by using the **As** keyword, or the data type of the parameter can be inferred. Either all parameters must have specified data types or all must be inferred.
- Optional and Paramarray parameters are not permitted.
- Generic parameters are not permitted.

Another example of using lambda expressions is shown in Figure 4.68.

For the first two lambda expressions, both are expressed by using **Function**, followed by the function body. The difference is that the first is a single-line expression, but the second is a multi-line expression with the **Return** and **End Function** statements involved.

The second two lambda expressions are expressed by using two subroutines, with one in a single-line and another one is multi-line expressions.

```

Dim increment1 = Function(x) x + 1
Dim increment2 = Function(x)
    Return x + 2
End Function

Dim writeline1 = Sub(x) Console.WriteLine(x)
Dim writeline2 = Sub(x)
    Console.WriteLine(x)
End Sub

```

Figure 4.68. A piece of sample codes for lambda expressions.

In some situations, the lambda expressions are combined with LINQ to simplify the query operations. One example is:

```

Dim faculty As Enumerable = IEnumerable(Of Faculty).Where(faculties, Function(f)
f.faculty_name = "Ying Bai")

```

Here, the Standard Query Operator 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(Of 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 keyword 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 IEnumerable(Of Faculty).Where(faculties, (Function(Faculty f) {id = faculty_
id Return f.faculty_id = id})

```

Here, the `IEnumerable(Of Faculty)` class is used to allow us to access and use the static method `Where()` since all Standard Query Operator 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 possible? The answer is maybe. To get that question answered correctly, let’s go to the next topic.

4.9.2 Extension Methods

Extension methods enable developers to add custom functionality to data types that are already defined without creating a new derived type. Extension methods make it possible to write a method that can be called as if it were an instance method of the existing type.

Regularly, static methods can only be accessed and used by classes in which those static methods are defined. For example, all Standard Query Operator 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.

When building and using extension methods, the following points should be noted:

1. An extension method can be only a `Sub` procedure or a `Function` procedure. You cannot define an extension property, field, or event. All extension methods must be marked with the extension attribute `<Extension()>` from the `System.Runtime.CompilerServices` namespace.
2. The first parameter in an extension method definition specifies which data type the method extends. When the method is run, the first parameter is bound to the instance of the data type that invokes the method.
3. Extension methods can be declared only within modules. Typically, the module in which an extension method is defined is not the same module as the one in which it is called. Instead, the module that contains the extension method is imported, if it needs to be, to bring it into scope. After the module that contains the extension method is in scope, the method can be called as if it were an ordinary instance method.
4. When an in-scope instance method has a signature that is compatible with the arguments of a calling statement, the instance method is chosen in preference to any extension method. The instance method has precedence even if the extension method is a better match.

Let's use an example to illustrate these important points and properties. Figure 4.69 shows a piece of codes that defines an instance and an extension method in the module `Conversion`.

In this example, both methods have the same name but different signatures. Let's have a closer look at this piece of codes to see how it works.

- A. The namespace `System.Runtime.CompilerServices` is imported first since we need to use some extension attributes defined in that namespace.
- B. First, a class `ExampleClass` is created with an instance method `ConvertToUpper()`. The first argument of this instance method is an integer. To call and execute this instance method, one must first create a new instance based on the class `ExampleClass`, and then

```

A Imports System.Runtime.CompilerServices
Module Conversion
B   Class ExampleClass
        'Define an instance method named ConvertToUpper.
        Public Function ConvertToUpper(ByVal m As Integer, ByVal aString As String) As String
            Console.WriteLine(vbNewLine & "Instance Method is called ")
            Return aString.ToUpper
        End Function
    End Class
C   <Extension()>
        Function ConvertToUpper(ByVal ec As ExampleClass, ByVal n As Long, ByVal aString As String) As String
            Console.WriteLine(vbNewLine & "Extension Method is called ")
            Return aString.ToUpper
        End Function
End Module

```

Figure 4.69. An example of defining class and instance method.

```

A Imports VB_Extensions.Conversion
    Module VBExtensions
        Sub Main()
B         Dim exClass As New ExampleClass
            Dim input As String = "Hello"
            Dim index_ext As Long = 5
            Dim index_ins As Integer = 1

            'The following statement calls the extension method.
C         Console.WriteLine(exClass.ConvertToUpper(index_ext, input))

            'The following statement calls the instance method.
D         Console.WriteLine(exClass.ConvertToUpper(index_ins, input))

            Console.WriteLine(vbNewLine & "Press any key to exit ")
            Console.ReadKey()
        End Sub
    End Module

```

Figure 4.70. An example of calling class and instance method.

call that method. The second argument is a string to be returned with the uppercase when this instance method is done.

- C.** The extension method is declared with the different signature. The type of the first argument of this class method is Long, and the second argument is also a string to be returned as the uppercase when this class method is executed. To call and execute this class method, one can directly call it with the class name prefixed in front of this method.

Figure 4.70 shows a piece of codes to illustrate how to distinguish these two methods when calling them with different signatures.

As we mentioned, to call the extension methods defined in a module, a different module should be created; here, the module **VBExtensions** is used for this purpose.

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

- A.** Since the extension method is defined in another module, **VB_Extensions.Conversion** (**VB_Extensions** is the name of a VB project in which the **Conversion** module is located), we need to import that module first to enable the codes developed in the current module **VBExtensions** to recognize it.
- B.** An instance of the class **ExampleClass**, **exClass**, is created, since we need to call the instance method **ConvertToUpper()** defined in that class in another module named **VB_Extensions.Conversion**. Also, some local variables are declared here.
- C.** First, we try to call the extension method with the type of the first argument as a Long.
- D.** Then we try to call the instance method with the first argument as an integer.

The running result is shown in Figure 4.71.

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 an existing static method, just redefine that existing static method with the **<Extension()>** keyword. For example, to make the class

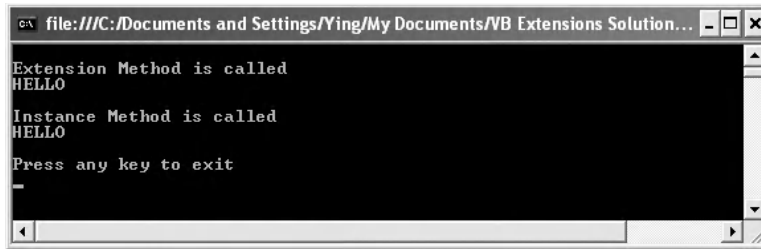


Figure 4.71. The running result of the execution of the extension method.

```
Imports System.Runtime.CompilerServices
Module Conversion
    Class ExampleClass
        'Define an instance method named ConvertToUpper.
        Public Function ConvertToUpper(ByVal m As Integer, ByVal aString As String) As String
            Console.WriteLine(vbNewLine & "Instance Method is called ")
            Return aString.ToUpper
        End Function
    End Class

    <Extension()>
    Function ConvertToUpper(ByVal ec As ExampleClass, ByVal n As Long, ByVal aString As String) As String
        Console.WriteLine(vbNewLine & "Extension Method is called ")
        Return aString.ToUpper
    End Function

A <Extension()>
B Function ToUpper(ByVal ec As ExampleClass, ByVal aString As String) As String
    Console.WriteLine(vbNewLine & "Extension ToUpper() Method is called ")
    Return aString.ToUpper
End Function
End Module
```

Figure 4.72. Declare the class method ToUpper() to extension method.

method ToUpper() an extension method, redefine that method with the <Extension()> keyword in a module, as shown by the codes that have been highlighted in bold in Figure 4.72.

Now the class method ToUpper() has been converted to an extension method and can be accessed by any instance of the class ExampleClass.

A complete project, VB Extensions, can be found in the folder DBProjects\Chapter 4 that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

4.9.3 Implicitly Typed Local Variables

In LINQ query, there is another language feature known as implicitly typed local variables that instructs the compiler to infer the type of a local variable. Instead of explicitly specifying a type when you declare and initialize a variable, you can now enable the compiler to infer and assign the type. This is referred to as local type inference. Local type inference works only when you are defining a local variable inside a method body, with Option Infer set to On. On is the default for new projects in LINQ.

As you know, with the addition of anonymous types to Visual Basic.NET, a new problem becomes a main concern, which is that if a variable being instantiated is an unnamed type, as in an anonymous type, of what type variable would you assign it to? LINQ queries belong to strongly typed queries with two popular types: `IEnumerable(Of T)` and `IQueryable(Of T)`, as we discussed at the beginning of this chapter. Figure 4.73 shows an example of this kind of variable with an anonymous type.

A compiling error will be encountered when this piece of codes is compiled since the data type of the variable `faculty` is not indicated. In Visual Basic.NET language enhancement for LINQ, a new terminology, implicitly typed local variable, is developed to solve this kind of anonymous type problem. Refereeing to Figure 4.74, the codes written in Figure 4.73 can be rewritten as shown in the figure.

This time there would be no error if you compile this piece of codes since the keyword *Dim* 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 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,

```
Module ImpLocal
    Class faculty
        Public faculty_id As String
        Public faculty_name As String
    End Class
    Sub Main()
        faculty = New With {.faculty_id = "B78880", .faculty_name = "Ying Bai"}
        Console.WriteLine("faculty information {0}, {1}", faculty.faculty_id & ". " & faculty.faculty_name)
    End Sub
End Module
```

Figure 4.73. Declare an anonymous type variable.

```
Module ImpLocal
    Class faculty
        Public faculty_id As String
        Public faculty_name As String
    End Class
    Sub Main()
        Dim faculty = New With {.faculty_id = "B78880", .faculty_name = "Ying Bai"}
        Console.WriteLine("faculty information {0}, {1}", faculty.faculty_id & ". " & faculty.faculty_name)
    End Sub
End Module
```

Figure 4.74. Declare an anonymous type variable using implicitly typed local variable.

```
Imports VB_Extensions.ImpLocal
Module ObjInitializer
    Sub Main()
        Dim fi = New faculty With {.faculty_id = "B78880",
                                   .faculty_name = "Ying Bai"}
    End Sub
End Module
```

Figure 4.75. An example of using the object initializer.

or other blocks because the type cannot be explicitly stated, and *Dim* is not legal for fields or parameter types.

Another important terminology applied in Visual Basic.NET language enhancement for LINQ is the object initializers. Object initializers are used in query expressions when you have to create an anonymous type to hold the results of a query. They also can be used to initialize objects of named types outside of queries. By using an object initializer, you can initialize an object in a single line without explicitly calling a constructor.

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 4.75.

In this example, there is no constructor of *faculty* that takes a faculty id and name; however, there are two properties, *faculty_id* and *faculty_name*, which can be set once an instance *fi* is created. Object initializers allow creating 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, and LINQ to XML, a valid query expression is needed. The query expressions implemented in Visual Basic.NET have a syntax that is closer to SQL statements and are composed of some clauses. Regularly, a query expression can be expressed in a declarative syntax similar to that of SQL or XQuery. At compile time, query syntax is converted into method calls to a LINQ provider's implementation of the standard query operator extension methods. Applications control which standard query operators are in scope by specifying the appropriate namespace with an *Imports* statement.

One of the most popular query expressions is the *For Each* statement. As this *For Each* is executed, the compiler converts it into a loop with calls to methods such as *GetEnumerator()* and *MoveNext()*. The main advantage of using the **For Each** 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 4.76.

Generally, a query expression is composed of two blocks. The top block in Figure 4.76 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.

```

Dim 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]

For Each range_variable In query_variable
    'pick up or retrieve back each element from the range_variable....
Next

```

Figure 4.76. A typical syntax of query expression.

```

Sub Main()
    Dim faculty As IEnumerable(Of Faculty) = From f In Faculty
                                              Let f.college <> "U.*"
                                              Where f.title = "Professor"
                                              Order By f.faculty_name Ascending
                                              Select f.phone, f.email

    'Execute the query to produce the results
    For Each fi In faculty
        Console.WriteLine("{0}, {1}, {2}, {3}", fi.faculty_name, fi.title, fi.phone, fi.email)
    Next
End Sub

```

Figure 4.77. A real example of query expression.

Referring to syntax represented in Figure 4.76, the following components should be included in a query expression:

- A query variable must be defined first in either explicitly (`IEnumerable(Of T)`) or implicitly (`Dim`) type
- 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**, **Order By**, **Join**, **Let**, and even additional **From** clauses

In all LINQ queries (including LINQ to DataSet), all of clauses will be converted to the associated Standard Query Operator methods, such as `From()`, `Where()`, `OrderBy()`, `Join()`, `Let()`, and `Select()`, as the queries are compiled. Refer to Table 4.1 in this chapter 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 **For Each** 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 for this topic.

Before we can finish this chapter, a real query example implemented in our project is shown in Figure 4.77.

In fact, the **Let** clause is not necessary in this query block, and it can be combined with the **Where** clause. Generally, the **Let** clause is used to perform some non-Boolean operations, but the **Where** clause is used to perform Boolean operations.

So far, we have provided a detailed discussion about LINQ queries in Visual Basic.NET with the most popular techniques and implementations. All sample projects involved in this chapter have been debugged and tested, and can be used directly in real applications.

4.10 CHAPTER SUMMARY

LINQ, which is built on .NET Frameworks 3.5, is a new technology released with Visual Studio.NET 2008 by Microsoft in 2008. LINQ is designed to 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 at the first part in this chapter. Some popular interfaces widely used in LINQ, such as *IEnumerable*, *IEnumerable(Of T)*, *IQueryable* and *IQueryable(Of 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 the LINQ that is implemented for different data sources is provided based on a 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. Visual Basic.NET 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 techniques applied for different data, but also practical and real feeling about the program codes developed to realize the desired functionalities.

Fifteen real projects are provided in this chapter to help readers to understand and follow up 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 Visual Basic.NET 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 Visual Basic.NET language enhancement for LINQ to perform all different queries to data sources

Starting from the next chapter, we will concentrate on the database programming with Visual Basic.NET using the 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(Of T), IQueryable and IQueryable(Of T).
- ____ 3. IEnumerable interface is used to convert the data type of the data source to IEnumerable(Of T) that 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 Visual Basic.NET.
- ____ 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(Of 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 method

- c. Variable methods
- d. Extension methods
- 4. One of the SQO methods, AsEnumerable() operator method, is used to convert the data type of the input object from _____ to _____
 - a. IQueryable(Of T), IEnumerable(Of T)
 - b. IEnumerable(Of T), IEnumerable(Of T)
 - c. Any, IEnumerable(Of T)
 - d. All of them
- 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(Of T)
 - b. IEnumerable, IEnumerable(Of T)
 - c. Deferred SQO, nondeferred 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(Of T)
 - c. SQLMetal, Object Relational Designer
 - d. IQueryable, IQueryable(Of 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(Of T), IEnumerable(Of T), IQueryable(Of T)
 - b. IEnumerable(Of T), IQueryable(Of T), IEnumerable(Of T)
 - c. IQueryable(Of T), IEnumerable(Of T), IQueryable(Of T)
 - d. IQueryable(Of T), IQueryable(Of T), IEnumerable(Of 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 4.0 Entity Framework, ADO.NET 4.0 Entity Framework Tools
 - c. IEnumerable(Of T), IQueryable(Of T)
 - d. Entity Data Model, Entity Data Model Designer
- 10. To access and implement ADO.NET 4.0 EF and ADO.NET 4.0 EFT, developers need to understand the _____ that is a core of ADO.NET 4.0 EF
 - a. SQLMetal
 - b. Object Relational Designer

- c. Generic collections
 - d. Entity Data Model
11. Lambda expressions 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(Of T), IQueryable(Of 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(Of 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, Order By
 - c. From, Select, Group
 - d. query variable, range variable, For Each 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 provide 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 For Each statement.
3. Explain the definitions and functionalities of the Standard Query Operator methods.
4. Explain the relationship between LINQ query expressions and Standard Query Operator methods
5. Explain the definitions and functionalities of IEnumerable, IEnumerable(Of T), IQueryable, and IQueryable(Of T) interfaces.
6. Explain the components and procedure used to perform LINQ to SQL queries.

```

Module Example4_78
    Sub Main()
        Dim fruits As New List(Of String)(New String() {"apple", "passionfruit", "banana", "mango", _
                                                         "orange", "blueberry", "grape", "strawberry"})

        Dim query = From fruit In fruits
                     Where fruit.Length < 6
                     Select fruit

        For Each f In query
            Console.WriteLine(f)
        Next

    End Sub
End Module

```

Figure 4.78. A LINQ to Object query.

7. A query used for LINQ to Objects, which is represented by a query syntax, is shown in Figure 4.78. Try to convert this query expression to a method's 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 Basic.NET

Starting from Visual Studio 2005, Visual Basic.NET added some new components and wizards to simplify the data access, inserting, and updating functionalities for database development and applications. Compared with Visual Studio 2005, Visual Studio 2008 added more new components to simplify the data accessing, inserting, and updating functionalities. Quite a number of new features, such as Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF), and Language Integrated Query (LINQ), had been added into Visual Studio.NET 2008.

The transition from the Visual Studio.NET 2008 to the Visual Studio.NET 2010 is more about extending the language to cope with new Windows 7 features than a radical of the language itself. Visual Basic.NET 2010 describes the new features in the Visual Basic language and Code Editor. The features include implicit line continuation, auto-implemented properties, collection initializers, and more. One of the significant differences between the Visual Studio.NET 2008 and Visual Studio.NET 2010 is that the former is built based on the .NET Framework 3.5, and the latter is built based on .NET Framework 4.0.

Starting from Visual Studio.NET 2005, Microsoft provides a quite few design tools and wizards to help users to 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

The Toolbox window in Visual Studio.NET 2010 contains data components that enable you to quickly and easily build simple database applications without needing to touch very complicated coding issues. Combining these data components with wizards, which are located in the Data Source wizard and related to ADO.NET, one can easily develop binding relationships between the data source and controls on the Visual Basic windows form object, furthermore one can build simple Visual Basic project to navigate, scan, retrieve and manipulate data stored in the data source with a few of lines of codes.

This chapter is divided to two parts. Part I provides a detailed description and discussion on how to use Visual Studio.NET 2010 tools and wizards to build simple but efficient database applications without touching complicated coding. In Part II, a deeper digging

in how to develop advanced database applications by using runtime objects is presented. More complicated coding technology is provided in this part. Some real examples are provided in detail with these two parts to enable readers to have a clear picture about 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 TableAdapter 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 query
- Build and execute simple dynamic data query commands to retrieve desired data

To successfully complete this chapter, you need to understand topics such as Fundamentals of Databases, which is introduced in Chapter 2, ADO.NET, which is discussed in Chapter 3, and LINQ to SQL, which is discussed in Chapter 4. Also, a sample database developed in Chapter 2 will be used through this Chapter.

PART I DATA QUERY WITH VISUAL STUDIO.NET DESIGN TOOLS AND WIZARDS

Before we can start the next section, a preview of a completed sample database application is necessary, and this preview can give readers a feeling about how a database application works and what it can do. The database used for this project is Access.

5.1 A COMPLETED SAMPLE DATABASE APPLICATION EXAMPLE

This sample application is composed of five forms, named **Login**, **Selection**, **Faculty**, **Student**, and **Course** forms. This example 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, student, 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**, 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 table located in the **CSE_DEPT** database. As the project runs, a data query will be executed via a

Table 5.1. Relationship between the Form and Data Table

VB Form	Tables in Sample Database
LogIn	LogIn
Faculty	Faculty
Course	Course
Student	Student, StudentCourse

**Figure 5.1.** The LogIn form.

dynamic SQL statement that is built during the configuration of each TableAdapter in the Data Source wizard. The retrieved data will be reflected on the associated controls that have been bound to those data fields.

Go to the folder DBProjects\Chapter 5 located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1), and find an executable file **SampleWizards Project.exe** that is located under the project folder **SampleWizards Solution\SampleWizards Project**. Double click on this file to run it.

As the project runs, a login form will be displayed to ask users to enter the username and password, which is shown in Figure 5.1. Enter **jhenry** and **test** as user name 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 user to continue to select and check the desired information related to faculty, course, or student, which is shown in Figure 5.2.

Select the default information—**Faculty Information** by clicking the **OK** button, and the **Faculty** form appears as shown in Figure 5.3.

All faculty names in the CSE department are listed in a **comboBox** control on the form. Select the desired faculty name from the **comboBox** control by clicking on the drop-down arrow, and click on the desired faculty name. To query all information for this



Figure 5.2. The Selection form.



Figure 5.3. The Faculty form.

faculty, click on the **Select** button to execute a prebuilt dynamic SQL statement. All information related to the selected faculty will be fetched from the faculty table in our sample database and reflected on five label controls in the Faculty form, as shown in Figure 5.3. A 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 the **Select** button to retrieve courses taught by this faculty, which will be displayed in the Course ListBox, as shown in Figure 5.4.

An interesting thing is that when you select the specified course by clicking on it from the Course list, all information related to that 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.

Figure 5.4. The Course form.

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 discuss how to design and build this demo project step by step by using SQL Server 2008. It is very easy to develop a similar project as this one 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 DESIGN TOOLS AND WIZARDS

When developing and building Windows application that needs to interface to database, a powerful and simple way is to use the design tools and wizards provided by Visual Studio.NET. By using this technique, the length of coding process can be significantly reduced, and the developing procedures can also be greatly simplified. Now, let's first take a look at those components resided in the Toolbox window.

5.2.1 Data Components in the Toolbox Window

Each database-related Windows application contains three components that can be used to develop a database application by using the data controls in the Toolbox: **DataSet**, **BindingSource**, and **TableAdapter**. Two other useful components are the **DataGridView** and the **BindingNavigator**. All of these components are located in the Toolbox window, as shown in Figure 5.5.

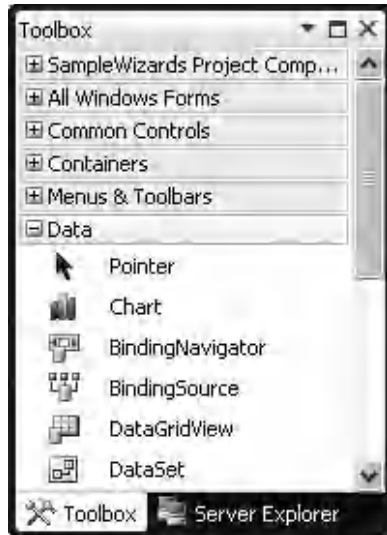


Figure 5.5. The Data components in Toolbox window.

5.2.1.1 The 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() event procedure 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, 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 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, 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.

With the Visual Basic.NET 2010, 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 in a manner more directly tied to the DataSet rather than having to deal with an XML Schema (XSD).

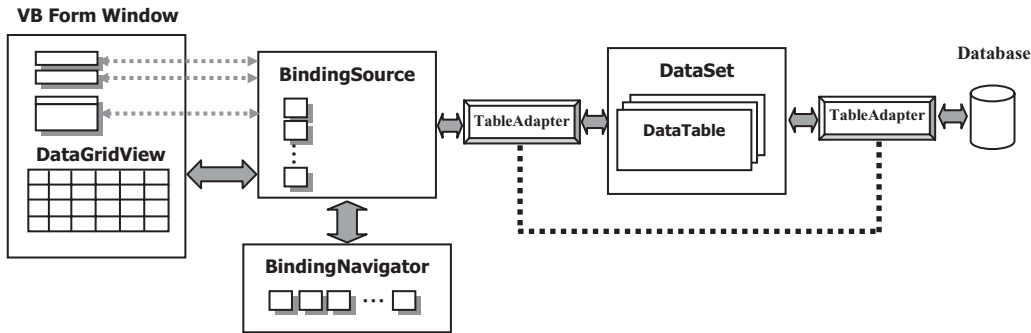


Figure 5.6. The relationship between data components.

In summary, 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 project and a database using ADO.NET, the *DataTables* in the *DataSet* can be populated with data of 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 for this issue.

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 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 in how to use the DataGridView control to bind and display data in Visual Basic.NET 2010 will be provided in Section 5.5 in this chapter.

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 to data 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.

The **BindingSource** control works as a bridge to connect the data bound controls on your Visual Basic forms with your data source (**DataSet**). The **BindingSource** control can also be considered as a container object, and it 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 Basic form and data in the **DataSet**, that is, you set up a connection between your controls on the Visual Basic 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 the **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 Basic.NET 2010, 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 users to scan and browse all records stored in the data source (**DataSet**) one by one in a 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 exist 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 component did. When the user clicks either the **Previous** or the **Next** button on the **BindingNavigator** UI, a request is sent to the **BindingSource** for the previ-

ous 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 Basic form need to be filled or updated.

Prior to Visual Basic.NET 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 had to call the **Update** method of each Data Adapter.

The **TableAdapter** belongs to designer-generated component, and you cannot find this component from the Toolbox window. The function of a **TableAdapter** is to connect 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 Data Source Configuration Wizard, when you build your applications.

The **TableAdapter** is similar to **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.NET 2010 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 Source Window

Two Integrated Development Environment (IDE) features included in the Visual Studio.NET, the **Data Sources Window** and the **Data Source Configuration Wizard**, are used to assist you to set 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

onto Visual Basic 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 Sources

The first time you create a new data-driven project in Visual Basic.NET 2010 environment, no data source that has been connected to your project, and, therefore, the Data Source window is a blank window with no data source in there. For example, you can create a new Visual Basic.NET 2010 Windows application by selecting **File/New Project** menu items and selecting the **DataSource** as the project name. After this new project is created and opened, you can find the Data Sources window by clicking the **Data** menu item from the menu bar, which is shown in Figure 5.7.

To open the Data Sources window, click **Data>Show Data Sources** item. Because you have no previous database connected to this new project, the opened Data Sources window is a blank one. To add a new data source or database to this new project, you can click on the item **Add New Data Source** from the **Data** menu.

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 your desired database to be connected with your new project.

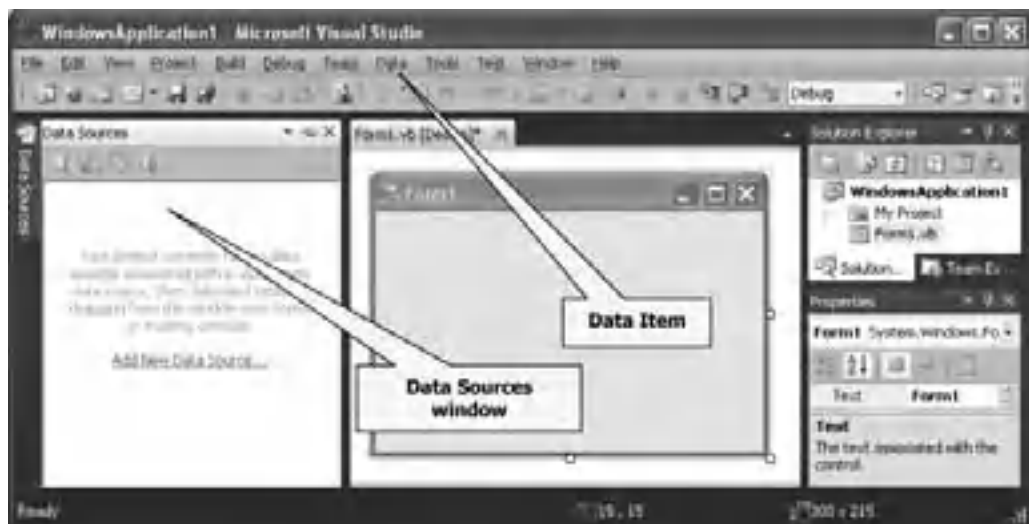


Figure 5.7. The Data Sources window.



Figure 5.8. The Data Source Configuration Wizard.

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 your desired data source or database that will be connected to your new project. The Data Source Configuration Wizard supports four types of data sources.

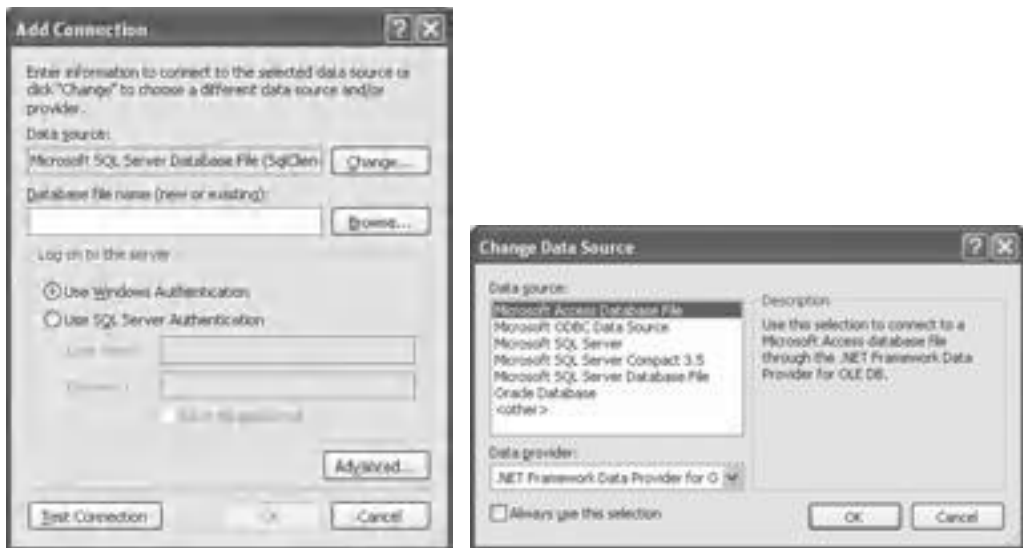
1. The first option, **Database**, allows you to select a data source for a database server on your local computer or on a remote server. The examples for this kind of data sources are SQL Server 2008 Express, Microsoft Data Engine (MSDE) 2000, or SQL Server 2008. This option also allows you to choose either an .mdf SQL Server database file or a Microsoft Access .accdb file. The difference between an SQL Server database and an 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.
2. The second option, **Web Service**, enable you to select a data source that is located at a Web service.
3. The third option, **Object**, allows you to bind your user interface to one of your own database classes.
4. The **SharePoint** allows users to connect to a SharePoint site and choose the SharePoint objects for the applications.

The next step in the Data Source Configuration Wizard allows you to either select an existing data connection or create a new connection for your data source, which is shown in Figure 5.9.

The first time you run this wizard, there is no preexisting 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 wizard is displayed, which is shown in Figure 5.10a.



Figure 5.9. Choose a Database Model in the Data Source Configuration Wizard.



(a)

(b)

Figure 5.10. The Add Connection and Change Data Source Wizards.

You can select different types of data source by clicking on the **Change** button. The Change Data Source wizard is displayed as you do that, which is shown in Figure 5.10b. Six (6) popular data sources can be chosen based on your application;

1. Microsoft Access Database File
2. Microsoft ODBC Data Source

3. Microsoft SQL Server
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 an SQL Server 2008 Express data source. For example, if you want to connect your new project with a Microsoft Access database named `CSE_DEPT.accdb`. You need to select the default Data Source as Microsoft Access Database File, and click the Browse button to locate and select that file. 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.

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 Basic.NET 2010 project. You can save this connection string for your further use if you want to use the same connection again for your application later.

When you click on 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 the data source into your project is that you can combine your project with the data source to make a complete application. In this way, you are free from worrying about any connection problem between your project and your data source, and they are of 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.

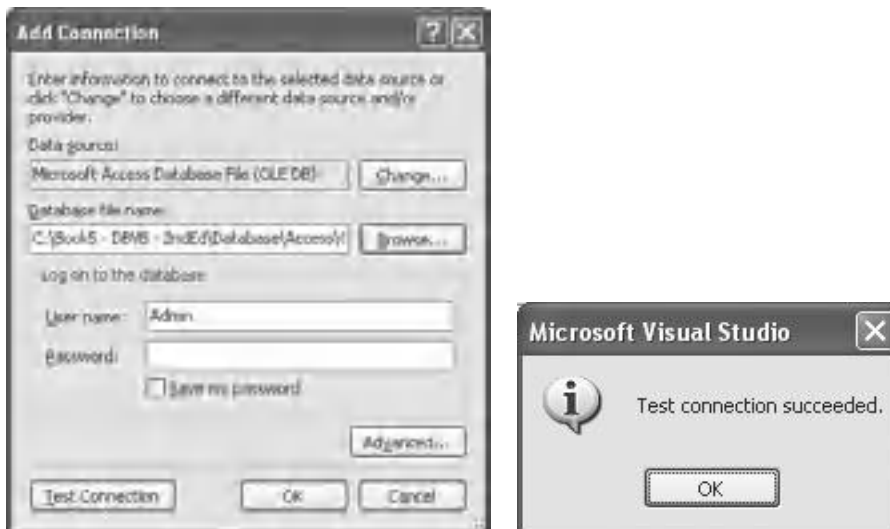


Figure 5.11. The Add Connection Wizard and Testing messagebox.

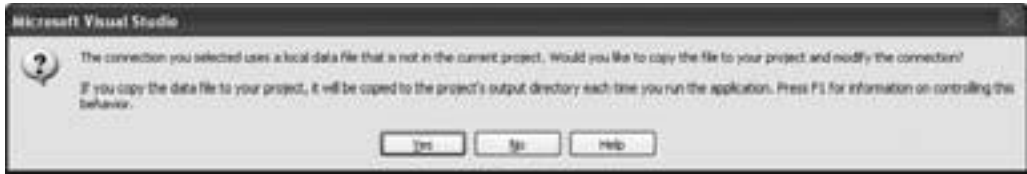


Figure 5.12. A message to ask you to save the data source.



Figure 5.13. Select database objects in the configuration wizard.

The next configuration step, which is shown in Figure 5.13, is to allow you to select the database objects for this data source. Although you can select any number of tables, 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.

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`, which is located at the DataSet name box shown in Figure 5.13. 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`), and you can use `Preview Data` to view data in each table in the DataSet. The wizard will build your `SELECT` statements for you automatically.

An important issue is that as you finish this Data Source Configuration and close this Wizard, the connection you set between your application and your database is closed. You need to use data query, data manipulation methods, or the `Fill()` method to reopen this connection if you want to perform any data action between your application and your database later.

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.

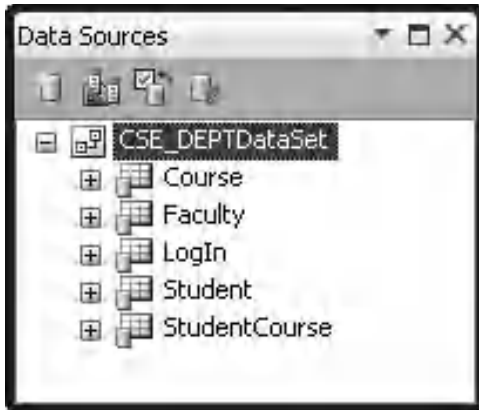


Figure 5.14. The added data source.

The data source added into your project is exactly 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 or tables as a tree view, and each table is connected to this tree via a node. If you click on the plus node “+” prefixed in each table, all columns of the selected table will be displayed.

Even after the data source is added into your project, the story has not been 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, make 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, DataColumns, 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 that you want to modify. 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).

In addition to multiple editing abilities mentioned above, you can perform the following popular data operations using the DataSet Designer:

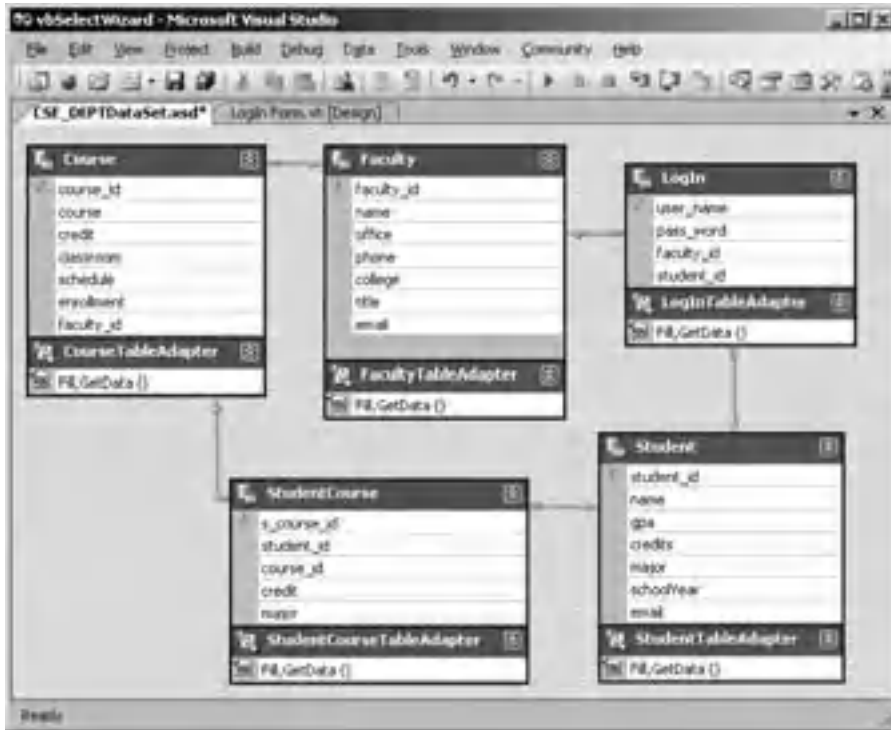


Figure 5.15. A sample DataSet Designer.

- **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.

Based on the above discussions, it can be seen that the DataSet Designer is a powerful tool to help users to design and manipulate the data source or DataSet, even the data source has been added into your project. But it has one more important function, which is to allow users to add any missing table to your project. In some cases, if you have forgotten to add a data table, or you add the wrong table (according to my experience, this has happened a lot for students who selected the wrong data source), you need to use this function to add that missed table, or first delete the wrong table and add the correct one.

To perform adding a missed table, just right-click a blank area of the designer surface and choose **AddDataTable**. You can also use this functionality to add a TableAdapter, Query, or a Relation to this DataSet. A more detailed exploration of DataSet Designer will be provided in Section 5.6.

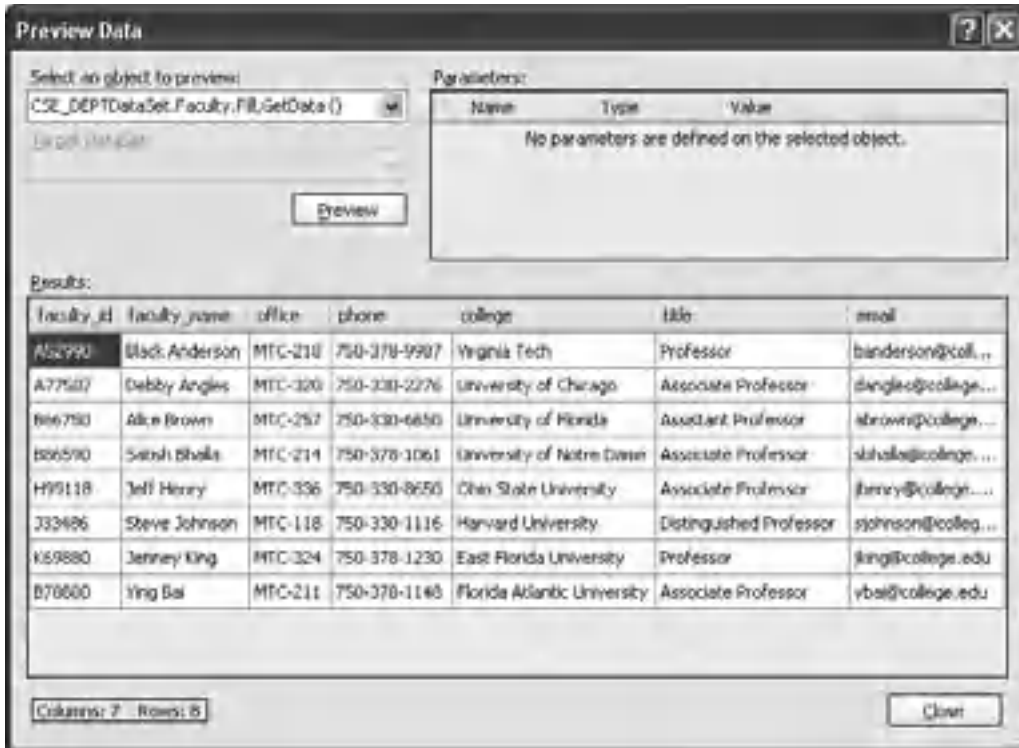


Figure 5.16. An example of the Preview Data for Faculty Table.

5.3 QUERY DATA FROM SQL SERVER DATABASE USING DESIGN TOOLS AND WIZARDS

So far, we have introduced and discussed most design tools located at both Visual Studio. NET 2010 Toolbox and Data Source window. In the following sections, 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 Basic.NET 2010 Windows-based project named SelectWizard, which means that we want to build a project with design tools and wizards provided by the Toolbox window and Data Source window.

First, let's take care of all graphic user interfaces (GUIs) in this project.

5.3.1 Application User Interface

We made a similar demo for this sample data-driven application in Section 5.1. This project is composed of five forms, named **Login**, **Selection**, **Faculty**, **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,

Table 5.2. Relationship between each form and data table

VB Form	Tables in Sample Database
LogIn	LogIn
Faculty	Faculty
Course	Course
Student	Student, StudentCourse

including faculty, courses taught by selected faculty, student, and courses taken by the associated students.

Each form, except the Selection form, is associated with one or two data tables in our sample database CSE_DEPT, which was developed in Chapter 2. The relationship between each form and tables is shown in Table 5.2.

Controls on each form are bound to the associated fields in certain data table 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 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 2008 R2 Express database, since it is compatible with SQL Server 2008 database, and more important, it is free and can be easily downloaded from the Microsoft Knowledge Base site. Refer to Appendix A to get details in how to download and install SQL Server R2 2008 Express database. You can use other databases, such as Microsoft Access or Oracle, for this 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.

All of these five forms are available from the folder VB Forms located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1). You can directly copy those forms and paste them into your project if you want to save time. However, here we want to provide a detailed discussion about how to build those forms.

Let's begin to develop this sample project with five forms.

5.3.1.1 The LogIn Form

First, let's create a new Visual Basic.NET 2010 Windows-based project with the name SelectWizard. You can first create a new solution named SelectWizard Solution and add this new project into that solution, or you can directly create this new project.

Open Visual Studio 2010 and go to the File/New Project item to open a New Project window. First create a new solution SelectWizard Solution, then right-click on the newly created solution and select the Add/New Project item. Make sure that the Windows Forms Application icon is selected from the Installed Templates window, enter SelectWizard into the Name textbox as the project's name, and click on the OK button to continue.

As the new project is created and the default windows form is opened, which is shown in Figure 5.17, perform the following modifications to this opened form:

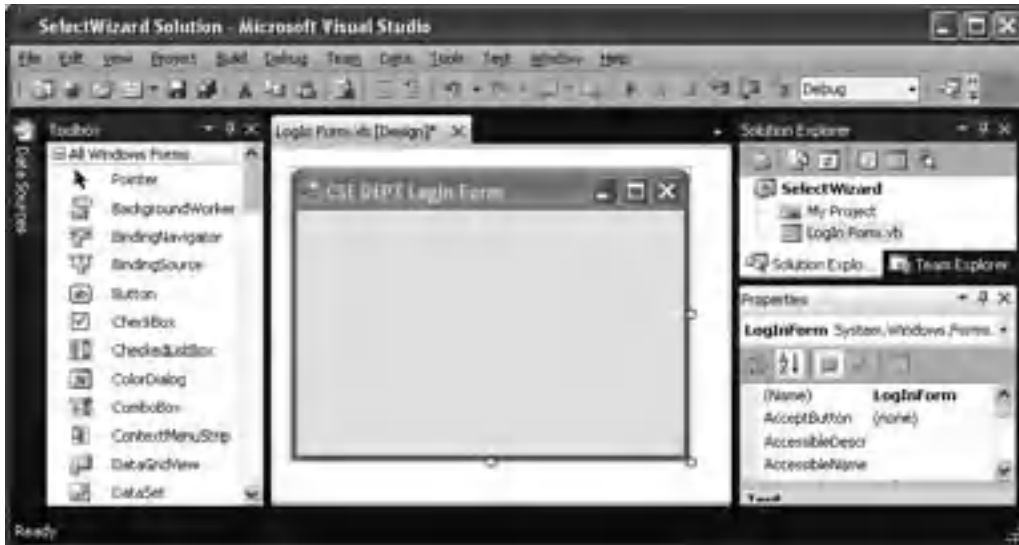


Figure 5.17. Create a new project window.

Table 5.3. Controls for 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

- Change the File Name from Form1.vb to LogIn Form.vb.
- 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.

Add the following controls shown in Table 5.3 into the LogIn form.

The finished LogIn form should match the one that is shown in Figure 5.18.

You should select the LogIn button, cmdLogIn, as the default button by choosing this button from the AcceptButton property of the form window. Also, you need to select the CenterScreen from the StartPosition property of the form.

5.3.1.2 The 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.



Figure 5.18. The LogIn form.

Table 5.4. Objects for 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	Selection Form		

Go to the **Project\Add Windows Form** menu item to add a new form with a file name of **Selection Form.vb**. The following objects need to be added into this form (Table 5.4).

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** from the **StartPosition** property of the form.

The completed Selection form should match the one that is shown in Figure 5.19.

5.3.1.3 The Faculty Form

The Faculty form contains controls that are related to faculty information stored in the Faculty table in our sample database **CSE_DEPT**, which is built in Chapter 2.

Go to **Project\Add Windows Form** menu item to add a new form with a file name of **Faculty Form.vb**. The finished Faculty form should match the one that is shown in Figure 5.20.

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** from the **StartPosition** property of the form.

The following objects need to be added into this form (Table 5.5).

In this chapter, we only use the **Select** button to make a data query to the data source. Other buttons will be used for the following chapters.

5.3.1.4 The Course Form

This form is used to interface to the Course table in your data source to retrieve course information associated with a specific faculty member selected by the user. Recall that in



Figure 5.19. The completed Selection form.



Figure 5.20. The finished Faculty form.

Chapter 2, we developed a sample database CSE_DEPT, and the Course table is one of five tables built in 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 the **Project!Add Windows Form** menu item to add a new form with a file name of `Course Form.vb`.

Add the objects shown in Table 5.6 into this Course form window.

The finished Course form should match the one that is shown in Figure 5.21.

Table 5.5. Objects on the Faculty form

Type	Name	Text	TabIndex	DropDownStyle
Label	Label1	Faculty Image	0	
TextBox	txtImage		1	
PictureBox	PhotoBox			
GroupBox	FacultyBox	Faculty Name & Query Method	2	
Label	Label2	Faculty Name	2.0	
ComboBox	ComboName		2.1	DropDownList
Label	Label3	Query Method	2.2	
ComboBox	ComboMethod		2.3	DropDownList
GroupBox	FacultyInfoBox	Faculty Information	3	
Label	Label4	Faculty ID	3.0	
TextBox	txtID		3.1	
Label	Label5	Name	3.2	
TextBox	txtName		3.3	
Label	Label6	Title	3.4	
TextBox	txtTitle		3.5	
Label	Label7	Office	3.6	
TextBox	txtOffice		3.7	
Label	Label8	Phone	3.8	
TextBox	txtPhone		3.9	
Label	Label9	College	3.10	
TextBox	txtCollege		3.11	
Label	Label10	Email	3.12	
TextBox	txtEmail		3.13	
Button	cmdSelect	Select	4	
Button	cmdInsert	Insert	5	
Button	cmdUpdate	Update	6	
Button	cmdDelete	Delete	7	
Button	cmdBack	Back	8	
Form	FacultyForm	CSE DEPT Faculty Form		

Table 5.6. Objects on the Course form

Type	Name	Text	TabIndex	DropDownStyle
GroupBox	NameBox	Faculty Name & Query 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	CourseIDLabel	Course ID	2.0	
TextBox	txtID		2.1	
Label	CourseLabel	Course	2.2	
TextBox	txtCourse		2.3	
Label	ScheduleLabel	Schedule	2.4	
TextBox	txtSchedule		2.5	
Label	ClassRoomLabel	Classroom	2.6	
TextBox	txtClassRoom		2.7	
Label	CreditsLabel	Credits	2.8	
TextBox	txtCredits		2.9	
Label	EnrollLabel	Enrollment	2.10	
TextBox	txtEnroll		2.11	
Button	cmdSelect	Select	3	
Button	cmdInsert	Insert	4	
Button	cmdUpdate	Update	5	
Button	cmdDelete	Delete	6	
Button	cmdBack	Back	7	
Form	CourseForm	CSE DEPT Course Form		

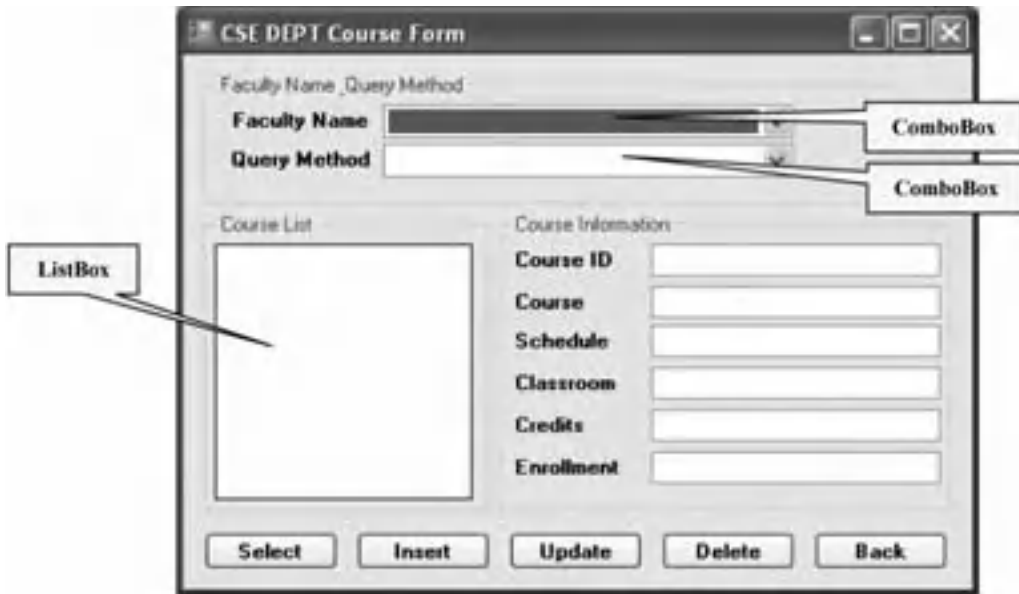


Figure 5.21. The completed Course form.

In this chapter, we only use the **Select** button to make a data query to the data source. The **Insert** and other buttons will be used in Chapters 6 and 7 for other data actions, such as data insertion, data updating, and deleting operations against our sample database CSE_DEPT.

5.3.1.5 The 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 one is the StudentCourse table. This is a typical example of using two data tables for one GUI (form).

Go to the **Project/Add Windows Form** menu item to add a new window form with a file name of **Student Form.vb**. Add the objects shown in Table 5.7 into this Student form window. Your finished Student form is shown in Figure 5.22.

Make sure that you set up the following properties for controls and objects:

- Make the **Select** button as the default button by selecting this button from the **AcceptButton** property of the form
- Select the **CenterScreen** from 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 bind and display the student's information. All courses, which are represented by the **course_id**, taken by the student, are reflected and displayed in a **ListBox** control, **CourseList**.

Table 5.7. Objects for the Student form

Type	Name	Text	TabIndex	DropDownStyle
PictureBox	PhotoBox			
GroupBox	StudentNameBox	Student Name & 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	Label3	Student ID	2.0	
TextBox	txtID		2.1	
Label	Label4	Student Name	2.2	
TextBox	txtName		2.3	
Label	Label5	School Year	2.4	
TextBox	txtSchoolYear		2.5	
Label	Label6	GPA	2.6	
TextBox	txtGPA		2.7	
Label	Label7	Major	2.8	
TextBox	txtStatus		2.9	
Label	Label8	Credits	2.10	
TextBox	txtCredits		2.11	
Label	Label9	Email	2.12	
TextBox	txtEmail		2.13	
Button	cmdSelect	Select	3	
Button	cmdInsert	Insert	4	
Button	cmdBack	Back	5	
Form	StudentForm	CSE DEPT Student Form		

Figure 5.22. The completed Student form.

5.4 ADD AND UTILIZE VISUAL STUDIO WIZARDS AND DESIGN TOOLS

After the GUIs are completed, next, we need to add a data source to this new project and set up a connection between our project and our sample database CSE_DEPT. In Section 5.2.2, we have provided a detailed discussion about how to add a new data source and how to configure a new added data source in a data-driven application. Now, we will illustrate these steps with a real Visual Basic.NET 2010 project, **SelectWizard**, we created in this section.

5.4.1 Add and Configure a New Data Source

Open the project **SelectWizard** and select the **LogIn** form window.

Go to **DataShow Data Sources** menu item to open the Data Source window. Currently, this window is a blank one since we have not added any data source to this project. Click on the link **Add New Data Source** to add a new data source to our project.

Perform the following operations to set up this data source connection:

1. On the opened Data Source Configuration Wizard, keep the default selection **Database** and click on the **Next** button.
2. On the next wizard, keep the default **DataSet** selection unchanged and click on the **Next** button to open the next wizard, which is shown in Figure 5.23a.
3. Click on the **New Connection** button to open the Add Connection dialog. This dialog allows us to select the database type and database name, which are determined by the actual database. We want to use an SQL Server 2008 Express database for this project, so first we need to change the default database type from the Microsoft Access Database File

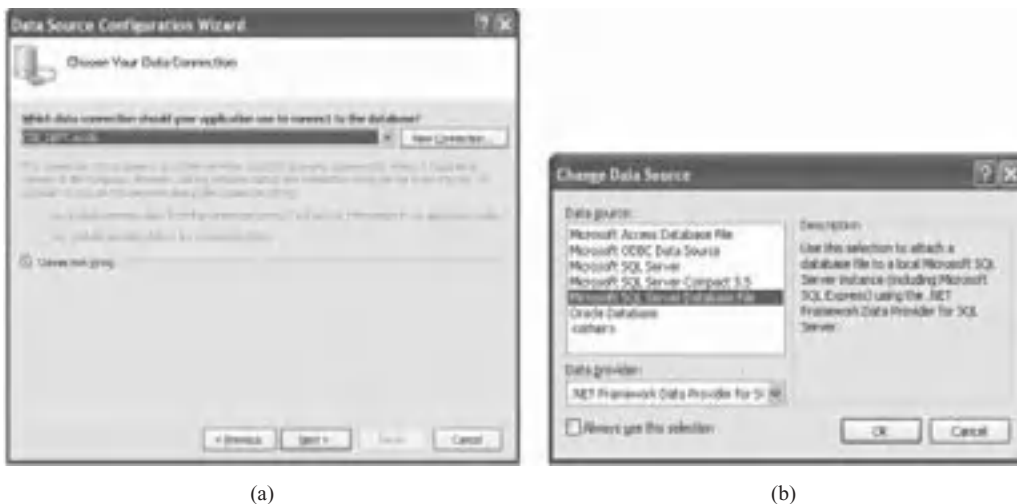


Figure 5.23. The Change Data Source wizard.

to the Microsoft SQL Server Database File by clicking on the **Change** button that is located next to the Data source box.

4. On the opened Change Data Source dialog, select the **Microsoft SQL Server Database File**, which is shown in Figure 5.23b.
5. Click on the **OK** button to select this database type.
6. Click on the **Browse** button to locate our desired database file. You should have developed your database using Microsoft SQL Server 2008 Express in Chapter 2. This database file is located at: `C:\Program Files\Microsoft SQL Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA` in your computer. In this example, the database name is `CSE_DEPT.mdf` with five data tables. You can find this database file from the folder `Database\SQLServer` from the Wiley ftp site (refer to Fig. 1.2 in Chapter 1). Select this database file and click on the **Open** button to add it into your project. A completed Add Connection dialog is shown in Figure 5.24a.
7. Before we can continue, we need to confirm that we are connecting to our target database. To confirm this, click on the **Advanced** button to open the Advanced Properties wizard. Go to the **Data Source** item to make sure that our target database `.\SQL2008EXPRESS` is selected in there. Otherwise, you need to change this to select our desired database.
8. Now you can test this Add Connection by clicking the **Test Connection** button. For the logon security, we use the default Windows Authentication mode. You can use the SQL-specific Username and Password if you like by selecting the checkbox: **Use SQL Server Authentication**.
9. Click the **OK** button to return to the Data Source Configuration Wizard, which is shown in Figure 5.24b. Click the **Next** button to go to the next wizard.



(a)



(b)

Figure 5.24. The Add Connection and Data Source Configuration Wizards.



Figure 5.25. Select the database objects.

10. A message box will pop up to ask if you like to add this data source into your project. As we discussed in Section 5.2.2.2, click on the Yes button to save the data source into your project.
11. The next window shows a message to ask if you like to save this connection string for future use; select the check box to save it and click on the Next button to continue.
12. The next step allows you to select different database objects. Generally, all data tables are necessary to be selected because we need to use those data to perform data operations between your Visual Basic.NET 2010 project and those tables in the connected database. The View object provides users with a view of tables, and it allows users to open and scan all data using the Preview Data functionality. Stored Procedures are used to combine a sequence of queries to form 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 be no damage to our project if we select all of them. So just check all checkboxes to select all of them, as shown in Figure 5.25.

You may already find that a new DataSet with the name of `CSE_DEPTDataSet` has been created, and it is located in the DataSet name box. Click on the Finish button to complete this configuration.

After you finish this Data Source Configuration, a new instance of the DataSet with a name of `CSE_DEPTDataSet` is added into your project, which is shown in Figure 5.26. 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.

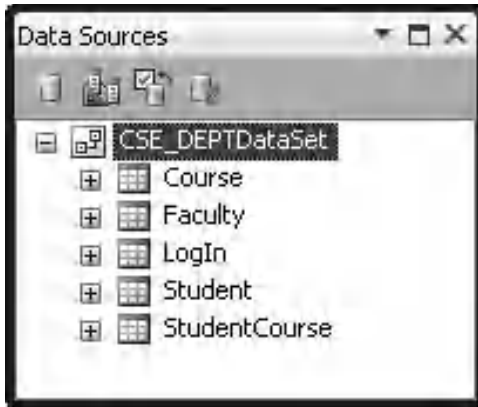


Figure 5.26. The new data source CSE_DEPTDataSet.

5.5 QUERY AND DISPLAY DATA USING THE DATAGRIDVIEW CONTROL

Now we have added a data source into your Visual Basic.NET 2010 project. Before we can develop this data-driven project, we want to show a popular but important functionality provided by the Toolbox window: DataGridView. As we discussed this issue in Section 5.2.1.2, the DataGridView is 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 in your Visual Basic form windows.

To use this tool, you can add a new blank form to the project SelectWizard, and name this new form as **Grid Form**. Go to **Project!Add Windows Form** to open the Add New Item dialog, enter **Grid Form.vb** into the Name box, and click on the Add button.

Select the newly added form **Grid_Form**, and open the Data Source window by clicking on the Data menu item from the menu bar. You can view data of 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 the Entire Table

To view the full Faculty table, click the Faculty table from the Data Source window, click on the drop-down arrow, and select the DataGridView item. Then drag the Faculty table to the Grid Form window, which is shown in Figure 5.27.

As soon as you drag the Faculty table to the Grid Form, a set of graphical components is created and added into your form automatically, which include the browsing arrows, Addition, Delete, and Save buttons. This set of components helps you to view data from

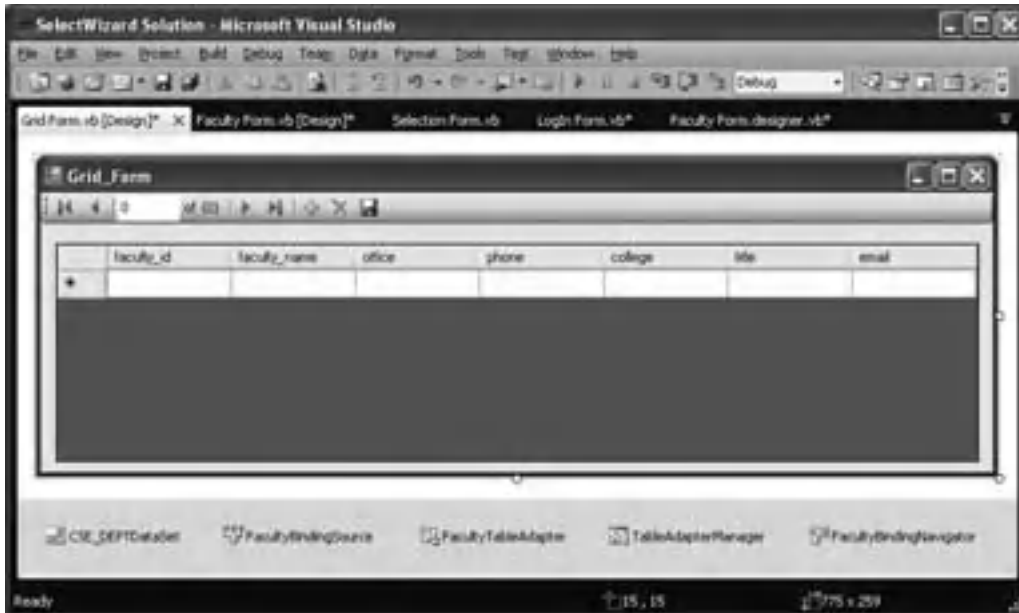


Figure 5.27. The DataGridView tool.

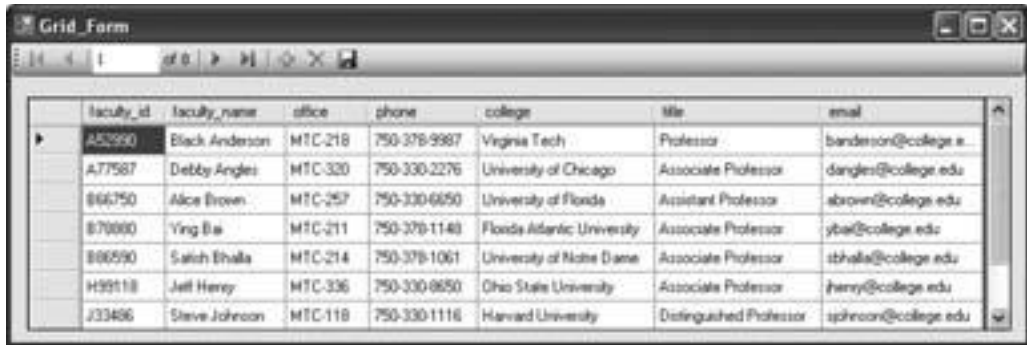
the selected table. To make a full table view, make sure to set two properties of the DataGridView, `AutoSizeColumnsMode` and `AutoSizeRowsMode`, to **AllCells**. In this way, you can display all data on this grid view tool.

Now you can run your project by clicking on the Start button. But wait a moment! One more thing before running the project is to check whether you have selected your Grid Form as the startup object from the project property menu item. To do that, go to **Project!SelectWizard Properties**; on the opened window, select **Grid_Form** from the **Startup form box**. Now run the project and you can find that the entire Faculty table is shown in this grid view tool, as shown in Figure 5.28.

By using this grid view tool, you can not only view data from the Faculty table, but also you can add new data into and delete data from the table by clicking the Add (+) or Delete (x) button to do that. Just type the new data in the new line after you click the Add button if you want to add new data, or move to the data you want to delete by clicking the browsing arrow on the top of the form window and then click the Delete button. One thing you need to know is that these modifications only take effect on data in your data tables in the DataSet; it has nothing to do with data in your database yet.

When you drag the Faculty table from the data source window to the Grid Form, what happened behind this dragging? Let's take a little deeper look at this issue.

First, you may already find that three components, `FacultyBindingSource`, `FacultyTableAdapter`, and `FacultyBindingNavigator`, have been added into this form as you perform this dragging. As we mentioned in Sections 5.2.1.1–5.2.1.5, those components are objects or instances that are created based on their associated classes, such as the `BindingSource`, `BindingNavigator`, and `TableAdapter` as you drag the Faculty table into your form window.



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
B70900	Ying Bai	MTC-211	750-378-1140	Florida Atlantic University	Associate Professor	ybai@college.edu
B86590	Satish Bhatta	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhatta@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

Figure 5.28. The entire table view for the Faculty table.

Second, let's look at the situation that occurred to the program codes, which are related to those objects and created automatically by the system when those new objects or instances are created as your dragging occurs. Open the Solution Explorer window and select the Grid Form.vb, then click on the View Code button to open the code window. Browse to the Grid_Form_Load() event procedure and you can find that a line of code is in there:

```
Me.FacultyTableAdapter.Fill(Me.CSE_DEPTDataSet.Faculty)
```

It looks like that the Fill() method, which belongs to the FacultyTableAdapter, is called to load data from the database into your DataGridView tool. The Fill() method is a very powerful method, and it performs an equivalent operation as an SQL SELECT statement did. To make it 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 FacultyTableAdapter as the Grid_Form_Load() event procedure runs when you start your project. The data returned from executing this statement will be filled to the grid view tool in the Faculty form.

5.5.2 View Each Record or the 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 on the Faculty table. Click on the drop-down arrow and select the Detail item. Drag the Faculty table from the data source window to the Faculty form window.

Immediately, you can find that three new objects, FacultyBindingSource, FacultyTableAdapter, and FacultyBindingNavigator, are added into the project. All column headers in the Faculty table are displayed, which is shown in Figure 5.29.

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 5.30.



Figure 5.29. The grid view for specified columns.



Figure 5.30. The running status of the grid view for each record.

To view each record, you can click on 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 Grid Form window. For each column you drag, an individual data-bound control is created on the Grid Form, accompanied by an appropriately titled label control. When you run your 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 allow users to view all data from a table. But generally, we do not want to perform that data view like an inline SQL statement did. The so-called inline SQL statement means that the SQL statement must be already defined in full before your project runs. In other words, you cannot add any parameter into this statement after your project runs, which we called dynamic or runtime SQL statements, and all parameters must be predefined 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 concentrate on this technique.

5.6 USE DATASET DESIGNER TO EDIT THE STRUCTURE OF THE DATASET

After a new data source is added into your new project, your 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 a 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 and 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 box, 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 built-in query, such as Fill(). But right now we want to add a new query to perform our specified data query.

On the opened TableAdapter Configuration Wizard, perform the following operations to build this customer query:

1. On the opened Choose a Command Type wizard, keep the default radio button selection **Use SQL statements** checked, and click on the **Next** button.
2. On the opened Choose a Query Type wizard, keep the default radio button selection **SELECT which returns rows** checked, and click on the **Next** button.
3. In the next wizard, click on the **Query Builder** button to open the Query Builder window to build our desired dynamic query. The opened Query Build wizard is shown in Figure 5.31.

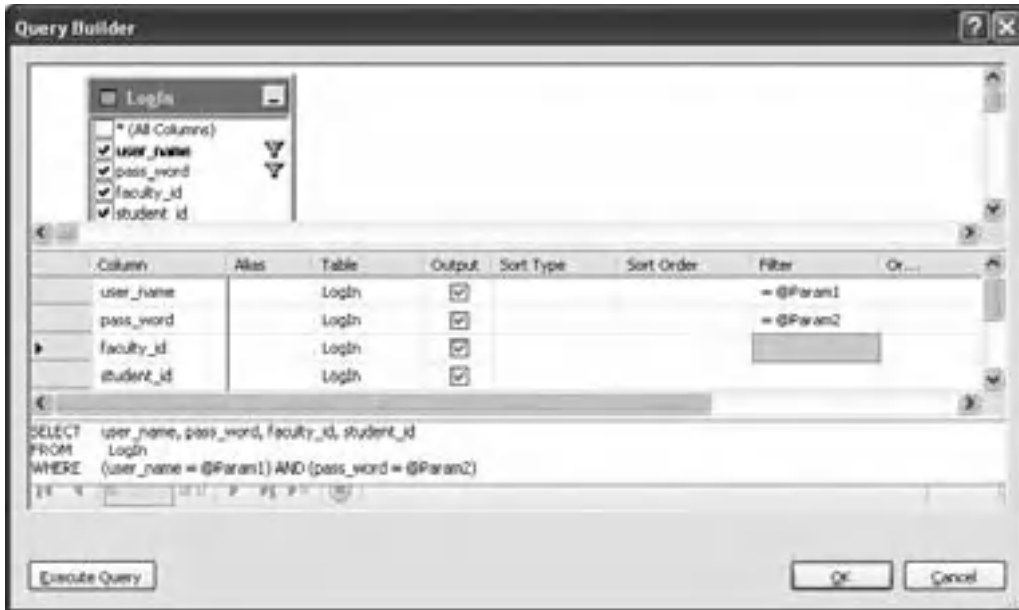


Figure 5.31. The Query Build window.

4. Move the cursor to the intersection cell of the `user_name` row and the Filter column, type `=?`, and press the *Enter* key from the keyboard. Perform the same operation to the intersection cell of the `pass_word` row and the Filter column.
5. Your finished query is displayed in the bottom text pane, as shown in Figure 5.31.

Query Builder provides a graphical user interface (GUI) for creating SQL queries, and it is composed of graphical panes and text 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 selected 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 pane. The selected columns will be displayed in the second graphical pane, which is also shown in Figure 5.31.

Since we try to build a dynamic SQL query for the `LogIn` table, what we want to do is: when the project runs, the username and password are entered by the user, and those two items will be embedded into an SQL `SELECT` statement that is sent to the data source, that is, to the `LogIn` table, to check if the username and password entered by the user can be found in 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 furthermore reflected on the bound control on the Visual Basic.NET 2010 `LogIn` form window.

The problem is that when we build this query, we do not know the values of the username and password, which will be entered by the user as the project runs. In other words, these two parameters are dynamic parameters. 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 an equal symbol followed by a question mark in the Filter column for `user_name` and `pass_word` rows in the second graphical pane, which is shown in Figure 5.31. The two question marks will become two dynamic parameters represented by `=@Param1` and `=@Param2`, respectively, after you press the *Enter* key from the keyboard. 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 5.31. The clause

```
WHERE (user_name = @Param1) AND (pass_word = @Param2)
```

is used to set up a dynamic criterion for this SELECT statement. Two dynamic parameters `Param1` and `Param2` will be replaced later by the 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 wizard. The next wizard 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 wizard provides you with three options: (1) allows you to modify the `Fill()` method to meet your specified query for your application; (2) allows you to modify the `GetData()` method that returns a new data table filled with the results of the SQL statement; and (3) allows you to add other SQL statements such as Insert, Update, and Delete.

For this application, we need to modify the name of the `Fill()` method by attaching `ByUserNamePassWord` to the end of the `Fill` method, which is shown in Figure 5.32. We will use this method in our project to run this dynamic SQL statement. Click on the Next button to go to the next wizard.

The next wizard 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 5.33. Click on the Finish button to complete this configuration.

Before we can begin to do our coding job, we need to bind data to controls on the `LogIn` form to set up the connection or binding relationship between each control on the `LogIn` form and each data item on the data source.

5.7 BIND DATA TO THE ASSOCIATED CONTROLS IN LOGIN FORM

Open the Solution Explorer window and select the `LogIn` Form, then click on the View Designer button to open its GUI. 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.



Figure 5.32. The Choose Methods to Generate window.



Figure 5.33. The result of the TableAdapter Configuration Wizard.

Click on the User Name TextBox, and then go to the **DataBindings** property that is 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

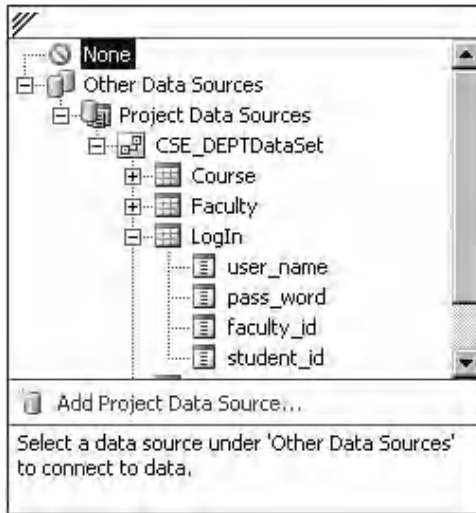


Figure 5.34. The DataBindings property.

- CSE_DEPTDataSet
- LogIn

The expansion result is shown in Figure 5.34.

Then select the `user_name` column by clicking on it. In this way, we finished 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 table in our sample database.



When you perform the first data binding, there is no `BindingSource` 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 newly created `BindingSource` to perform all data bindings.

You can find that three objects, `CSE_DEPTDataSet`, `LogInBindingSource`, and `LogInTableAdapter`, have been added into the project and displayed at the bottom of the window after you finish this binding operation.

Well, is 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, there is no `BindingSource` object available because you have not performed any data binding before, and the User Name is the first control you want to bind. You need to perform those steps as we did above, which is illustrated in Figure 5.34. However, after you finish that binding, a new `BindingSource` object, `LogInBindingSource`, is created. You need to

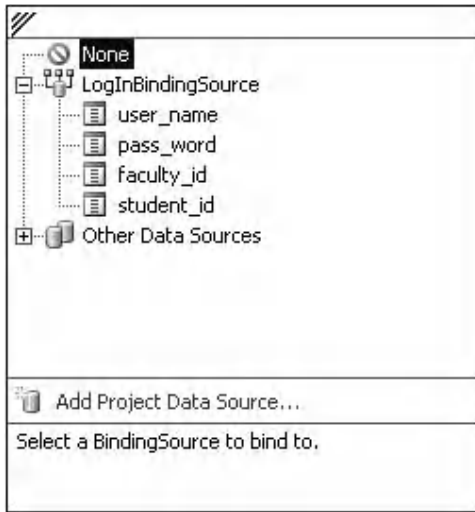


Figure 5.35. The created BindingSource object LogInBindingSource.

use this created BindingSource object to handle all other data binding jobs for all other controls on the LogIn form.

Let's perform the data binding for the Pass Word TextBox now.

Click that 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**, has shown up (Fig. 5.35).

Expand this new binding source object and select the **pass_word** column by clicking on it. The data binding for **pass_word** is done.

Some readers may have noted that when we call the **Fill()** method, that is, the **FillByUserNamePassWord()**, from the **LogInTableAdapter**, 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**, because 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 in the LogIn table, that means that the log in is successful and we can continue for the next step. Two bound-control on the form, **txtUserName** and **txtPassWord**, will be filled with the identical values stored in the LogIn table. It looks like that this does not make sense. In fact, we do not want to retrieve any column from the LogIn table, but instead, we only want to find the matched items of the username and password, which are entered by the user, for two columns from the LogIn table: **user_name** and **pass_word**. If we can find the matched user name and pass word, 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 given.

Before we can go ahead to our coding, one thing we need to point out is 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

runs. To make this happen to our project, you need to set the **PasswordChar** property of the textbox control **txtPassWord** to a star *.

Now it is the time for us to develop codes that are related to those 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 shown below:

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** event procedure.
3. The **LogIn** event procedure will first create some local variables or objects that will be used for the data query and displaying of the next form, **SelectionForm**.
4. Then the procedure will call the **FillByUserNamePassWord()** query method to fill the **LogIn** form.
5. If this **Fill** is successful, which means that a pair of matched data items for username and password has been found in the **LogIn** table, the next window form, **SelectionForm**, will be displayed for the next step.
6. Otherwise, a warning message is displayed.

The new objects created in step 3 include a new object of the **LogInTableAdapter** class, a new object of the next window form class, **SelectionForm**, since we need to use the **LogInTableAdapter** object to call the **FillByUserNamePassWord()** method and to use a new object to show the next window form **SelectionForm**.

Keep those points in mind, and now let's begin to do the coding for the **LogIn** button event procedure.

5.8 DEVELOP CODES TO QUERY DATA USING THE **FILL()** METHOD

Select the **LogIn** Form from the Solution Explorer window and click on the **View Designer** button to open its GUI. Double-click on the **LogIn** button to open its event procedure.

Based on step 3 in the above operation sequence, first we need to create two local objects: the **LogInTableApt** is an object of the **LogInTableAdapter** class, and the **selfForm** is an object of the **SelectionForm** class. The newly created objects are shown in the top two lines (A) in Figure 5.36.

You need to note that all **TableAdapters** in this project are located in the namespace **CSE_DEPTDataSetTableAdapters**. You need to use this namespace to access the desired **TableAdapter**.

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

- B. Before filling the **LogIn** 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 your project runs.

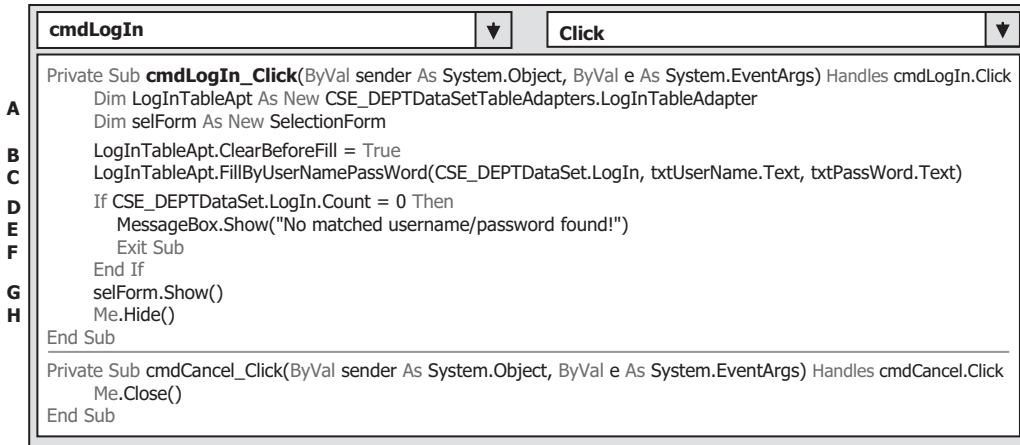


Figure 5.36. The codes of the LogIn button event procedure.

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`.

- C. Now we need to call the `Fill()` method we modified in Section 5.6, that is, the `FillByUserNamePassWord()`, 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`, 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 modify 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.

- D. If a matched username and password is found in the LogIn table in the database, the `Fill()` method will fill the LogIn table in the `DataSet`, and at the same time, these two filled columns will be reflected on two bound textbox controls on the LogIn form, `txtUserName` and `txtPassWord`. The `Count` property of the LogIn table in the `DataSet` will be set. Otherwise, this property will be reset to 0. By checking this property, we will know if this Fill is successful or not, or if a matched username and password is found in the database. If this property is 0, which means that no matched item is found in the database, and therefore no column is filled for the LogIn data table in the `DataSet`, the login has failed.
- E. Then a warning message is displayed to ask users to handle it.
- F. An `Exit Sub` is executed to exit the event procedure. You need to note that to exit the event procedure does not mean to exit the project, and your project is still running and waiting for the next login process.
- G. If the login process is successful, the next window form, `SelectionForm`, will be shown to allow us to continue to the next step.



Figure 5.37. The running status of the LogIn form.



Figure 5.38. The warning message.

- H.** After displaying the next form, the current form, LogIn form, should be hidden by calling the Hide() method. The keyword **Me** represent the current window form.

The code for the **Cancel** button event procedure is very simple. The Close() method should be called to terminate your project if this button is clicked by the user.

Before we can test this piece of codes by running the project, perform the following two operations:

1. Make sure that the LogIn form has been selected as the Start form from the project property window. To confirm this, go to **Project!SelectWizard Properties** to open the **Application** window, and select the LogInForm from the **Start** form box.
2. Remove all codes inside the LogInForm_Load() event procedure since those codes are generated by the system automatically and we do not need those codes.

Now click on the Start button to run the project. Your running project should match the one that is shown in Figure 5.37.

Enter a valid username, such as **jhenry**, to the User Name textbox and a valid password, such as **test**, to the Pass Word textbox, and then click on the LogIn button. The FillByUsernamePassWord() method will be called to fill the LogIn table in the data source. Because we entered the 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; a MessageBox will be displayed, which is shown in Figure 5.38, to ask 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 from those columns in a data table in the database by using the Visual Studio.NET tools and data wizards. The coding is relatively simple and easy to follow. In the next section, we try to discuss 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 may have experienced forgetting either the username or the password when they try to log on to a specified Web site to purchase, order, or try to obtain some information. In this section, we will show users how to use a method to retrieve a single data value from the database. This method belongs to the TableAdapter.

We still use the LogIn form and LogIn table as an example. Suppose you forget your password, but you want to log in 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, and add a new query, a new column, and even a new key. The Add Query method allows us to add a new data query with an SQL SELECT statement that returns a single value.

Open the LogIn form window from the Solution Explorer window and open the Data Source window by clicking 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()**. Then select **Add Query** to open the TableAdapter Query Configuration Wizard. Perform the following operation to build this query:

1. 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 the **SELECT which returns a single value** radio button. Click on the **Next** button to go to the next wizard and click on the **Query Builder** button to build our query.
2. Delete the default query from the third pane by right-clicking on that query, and select **Delete**.
3. Then right-click on the top pane and select the **Add Table** item from the pop-up menu to open the **Add Table** dialog, select the LogIn table and click on the **Add** button, and then click on the **Close** button to close this **Add Table** dialog.
4. Select the **pass_word** and **user_name** columns from the LogIn table by checking those two checkboxes.
5. Right-click on the **Group By** column in the middle graphical pane and select **Delete** to remove any group choice if the **Group By** tab is displayed.
6. Uncheck the **Output** checkbox from the **user_name** column since we do not want to use it as the output; instead, we need to use it as a criterion for this query.
7. Type **=?** on the **Filter** field from the **user_name** column and press the **Enter** key from your keyboard. Your finished Query Builder is shown in Figure 5.39.

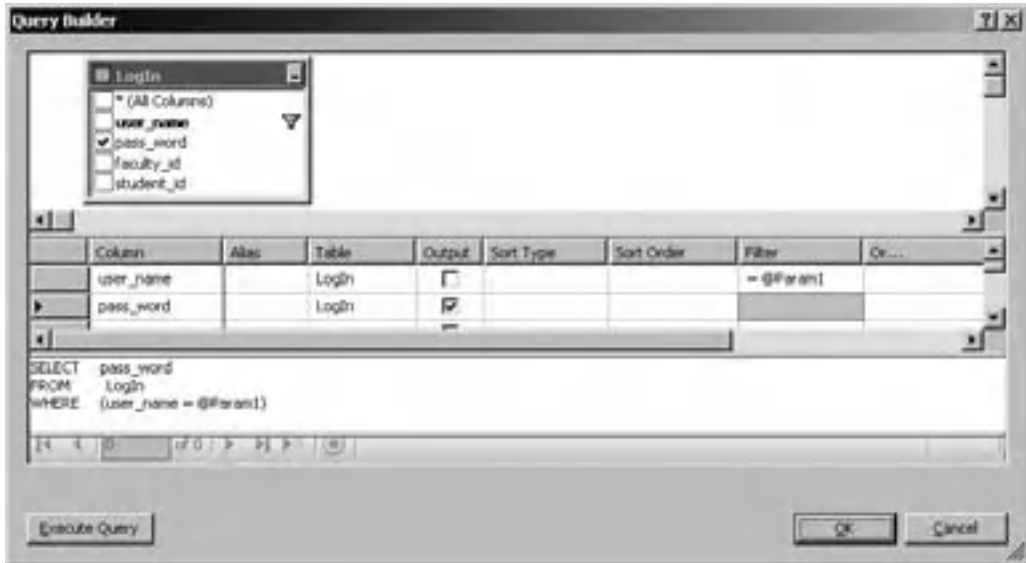


Figure 5.39. The finished Query Builder.

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 that 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 wizard.

The next wizard is used to confirm your terminal SQL statement. Click on the Next button to go to the next wizard.

This wizard asks you to choose a function name for your 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 the testing purpose, first we need to add a temporary button to the LogIn form to perform this password checking function. Go to the ToolBox window and drag a button control to the LogIn form, and set up the following properties to this button:

Name = cmdPW and Text = Password

Then open the Solution Explorer window, select and open the LogIn form, double-click on the new added Password button to open its event procedure, and enter the codes shown in Figure 5.40 into this event procedure.

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

- A.** Create two String local variables. The **passWord** is used to hold the returning queried single value of the **pass_word**, and it is a text string. The **Result** is used to compose a resulting string that contains the returned password from the query.
- B.** Call the query we just built in this section, **PasswordQuery()**, with a dynamic parameter **username** that is entered by the user as the project runs. If this query found a valid password from the LogIn table based on the **username** entered by the user, that the password will be returned and assigned to the local string variable **passWord**.

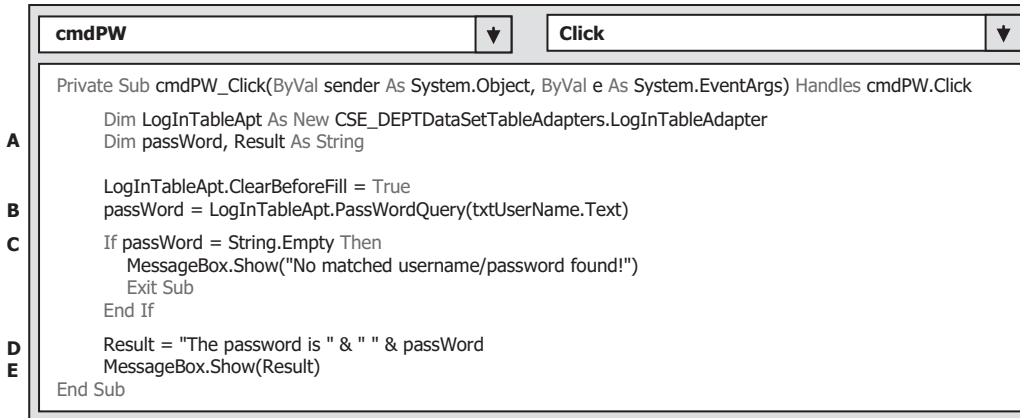


Figure 5.40. The codes for the cmdPW button event procedure.



Figure 5.41. The running status of the LogIn form.

- C. If this query cannot find any matched password, a blank string will be returned and assigned to the variable `passWord`. A MessageBox with a warning message will be displayed if this situation did happen. The program will be directed to exit this subroutine.
- D. If the query calling is successful, then a valid password is returned and assigned to the variable `passWord`. A composed string combined with the returned password is made and assigned to the String variable `Result`.
- E. A MessageBox is used to display this found password.

Click on the Start button to run the project, and your running project should match the one that is shown in Figure 5.41.

Enter a username, such as `jking`, 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 5.42.

Well, it looks like fun! Is not it?

Now you can remove the temporary button `PassWord` and its event procedure from this LogIn form if you like since we need to continue to develop our project.



Figure 5.42. The returned password.



Figure 5.43. The Selection Form.

In the following sections, we will discuss 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 Selection Form.

5.10 DEVELOP THE CODES FOR THE 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 5.43 shows an opened SelectionForm window.

Each piece of information in the ComboBox control is associated with a form window and is also associated with a group of data stored in a data table in the database.

The operation steps for this form are summarized as below:

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 on 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 coding to display three pieces of information should be located in the `Form_Load()` event procedure since this event procedure should be called first as the project runs.

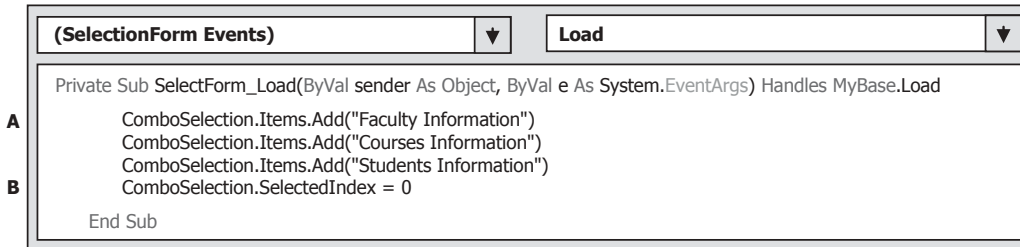


Figure 5.44. The codes for the Selection Form_Load event procedure.

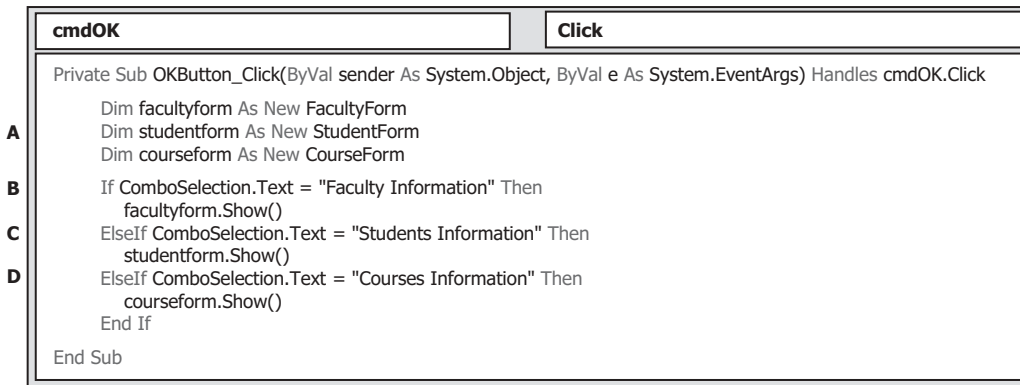


Figure 5.45. The codes for the OK button event procedure.

Open the Selection Form window and click on the View Code button to open its code window. Select the **SelectionForm Events** item from the **Class Name** combo box and choose the **Load** item from the **Method Name** combo box to open its **SelectionForm_Load()** event procedure, and enter the codes shown in Figure 5.44 into this event procedure.

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

- A.** The **Add** method of the **ComboBox** control is called to add all three pieces of information. The argument of this method must be a **String** variable and have to be enclosed by double quotation marks.
- B.** The **SelectedIndex** of this **ComboBox** control is reset to 0, which means that the first item, **Faculty Information**, is selected as the default information.

According to step 2 described above, when users click on the **OK** button, the form related to the information selected by the user should be displayed to allow users to browse information from that form. Click on the View Designer button to open the GUI of the **SelectionForm** object. Then double-click on the **OK** button to open its event procedure and enter the codes shown in Figure 5.45 into this procedure.

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

- A.** First, we need to create three new objects based on three classes. You need to note that when you add any new Window Form into your project, the new item is a class, not an object. You need to create a new object or new instance based on that class to use it.

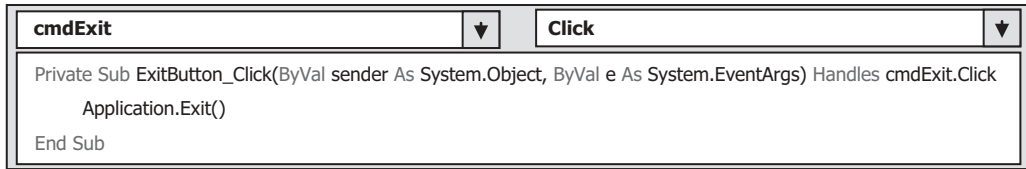


Figure 5.46. The codes for the Exit button event procedure.

- B.** Open the FacultyForm window if the user selected the Faculty Information.
- C.** Open the StudentForm window if the user selected the Student Information.
- D.** Open the CourseForm window if the user selected the Course Information.

The last coding for this form is the Exit button. Open the GUI of the SelectionForm, and double-click on the Exit button to open its event procedure. Enter the codes shown in Figure 5.46 into this procedure.

This piece of codes is very simple. As the user clicks on the Exit button, the project is exited and terminated by calling the Exit() method.

Suppose the user selected the first item—Faculty Information. A Faculty form window will be displayed, and it is supposed to be connected to a Faculty table in the database. If the user selected a faculty name from the ComboBox control and clicked on the Select button on that form (refer to Fig. 5.20), all pieces of information related to that faculty should be displayed on seven textboxes and a picturebox on that form.

Now let's first to see how to perform the data-binding to bind controls on the Faculty form to the associated columns in the database.

5.11 QUERY DATA FROM THE FACULTY TABLE FOR THE FACULTY FORM

First, let's open the Faculty form window from the Solution Explorer window and perform the following data bindings.

1. Select the Faculty ID textbox txtID by clicking on it, then go to the Properties Window and select the **DataBindings** property, select the **Text** item, and click on the drop-down arrow. Expand the following items (Fig. 5.47):
 - Other Data Sources
 - Project Data Sources
 - CSE_DEPTDataSet
 - Faculty

Then select the **faculty_id** column from the Faculty table by clicking on it. In this way, you finish the binding between the textbox control txtID on the Faculty form and the **faculty_id** column in the Faculty table. As soon as you finish this data binding, immediately, you can find that three components are displayed under your form; CSE_DEPTDataSet, FacultyBindingSource, and FacultyTableAdapter.

2. Perform a similar operation for the next textbox txtName in the Faculty form to bind the **Name** and the **faculty_name** column in the Faculty table. Go to the **DataBindings** property and then select the **Text** item, then click on the drop-down arrow. This time, you will



Figure 5.47. The expansion for data binding.



Figure 5.48. An example of the expansion for faculty_name column.

find that a new object `FacultyBindingSource` has been created. As we discussed in Section 5.7, as soon as you finish the first data binding, a new binding object related to the data-binding source will be created and served for the form in which the binding source is located. We need to use this binding source to bind our `Name` control. Expand this binding source until you find the `Faculty` table, then click on the `faculty_name` column to finish this binding. An example of this expansion is shown in Figure 5.48.

3. In the similar way, you can finish the data binding for the rest of the textbox controls; `txtTitle`, `txtOffice`, `txtPhone`, `txtCollege`, and `txtEmail`. The binding relationship is: `txtTitle` → `title` column, `txtOffice` → `office` column, `txtPhone` → `phone` column, `txtCollege` → `college`, and `txtEmail` → `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 FacultyTable-Adapter. Perform the following operations to complete this query building process:

1. Open the Data Source window by clicking the **DataShow Data Sources** menu item from the menu bar.
2. Right-click on any place inside that window and select **Edit DataSet with Designer** item to open the DataSet Designer Wizard.
3. Locate the Faculty table, then right-click on the last line of the Faculty table and select the **Ad Query** item from the pop-up menu to open the TableAdapter Configuration Wizard.
4. On the opened Wizard, click on the **Next** button to keep the default command type—**Use SQL statements** and click on the **Next** button to keep the default query type—**SELECT** which returns rows for the next wizard. Then click on the **Query Builder** button to open the Query Builder wizard.
5. In the middle graphical pane, move your cursor to the **Filter** column along the **faculty_name** row, then type a question mark **?** and press the Enter key from your keyboard. In this way, a **WHERE** clause with a dynamic parameter represented by **LIKE @Param1** is added into the SQL Server database. Note that the keyword **LIKE** is similar to an equal symbol used in the assignment operator in Microsoft Access query. In SQL Server data query, **LIKE** is used to replace the equal symbol.
6. Your finished Query Builder should match the one that is shown in Figure 5.49.
7. Click on the **OK** and **Next** buttons to modify the name of the FillBy() method to **FillByFacultyName**. Click on the **Next** button and then the **Finish** button to complete this configuration.

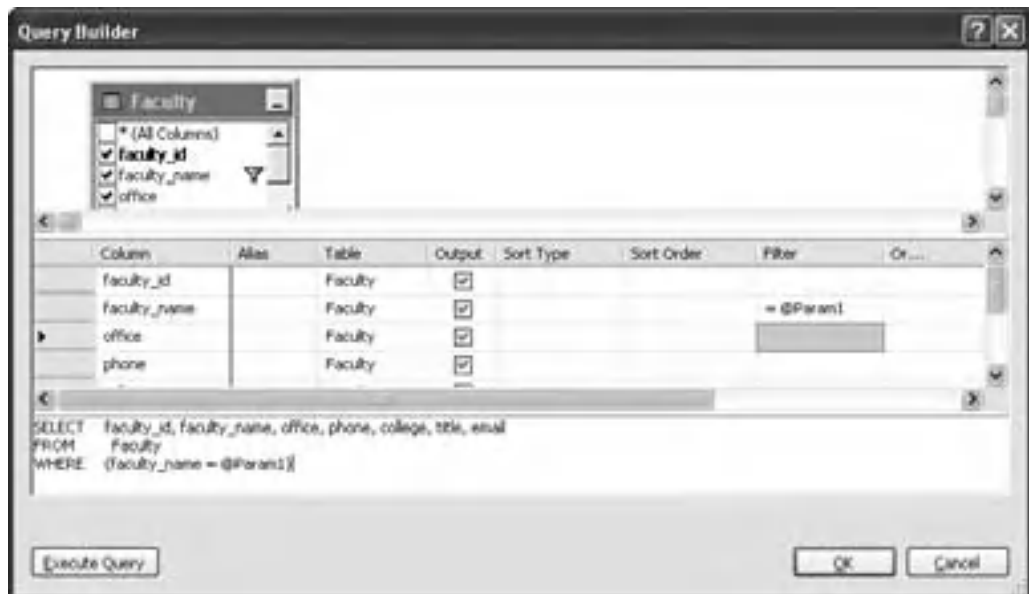


Figure 5.49. An example of the Query Builder.

Now let's develop the codes 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 THE 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 event procedures, and the coding for all other buttons will be discussed and coded in the later sections.

5.12.1 Develop Codes to Query Data Using the TableAdapter Method

As we mentioned above, the pseudo-code or the operation sequence of this data query can be described as:

- 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 pieces of information related to that desired faculty.

The main coding job is performed within the **Select** button event procedure. But before we can do that coding, first, we need to add all default 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` event procedure.

Select the `Faculty Form.vb` from the Solution Explorer window and click on the View Code button to open the code window. Go to the Class Name combo box and click on the drop-down arrow to select the `FacultyForm Events` item. Go to the Method Name combo box and click on the drop-down arrow and select the `Load` button. This will open the `FacultyForm_Load` event procedure. Remove all original codes and enter the codes shown in Figure 5.50 into this event procedure.

Now we need to do the coding for the **Select** button event procedure.

Click on the View Designer button to open the Faculty form window. On the opened Faculty form, double-click on the **Select** button to open its event procedure, and then enter the codes shown in Figure 5.51 into this event procedure.

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

- A.** First, create a new FacultyTableAdapter object, `FacultyTableApt`. In Visual Basic.NET 2010, you have to create a new instance or object based on the data component class if you want to use any method or property that belongs to that class.

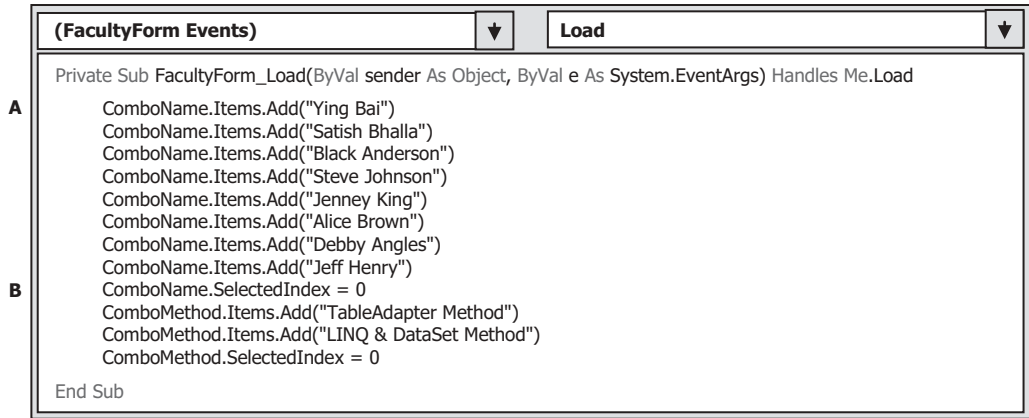


Figure 5.50. The codes for the FacultyForm_Load event procedure.

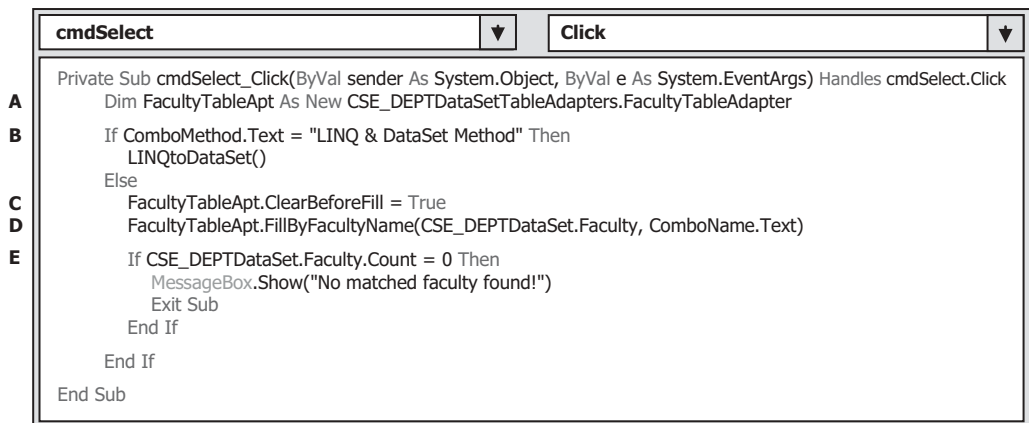


Figure 5.51. The codes for the Select button event procedure.

- B.** If the user selected the LINQ & DataSet Method, a user-defined subroutine LINQtoDataSet() that will be built later is called to perform a LINQ to DataSet query.
- C.** Otherwise, the SQL SELECT query method has been selected. First, we need to clean up the Faculty table before it can be filled by setting the ClearBeforeFill property to True.
- D.** Call the method FillByFacultyName() to fill the Faculty table with a dynamic parameter, Faculty Name, which is selected by the user from the Faculty Name ComboBox control as the project runs.
- E.** By checking on the Count property of the Faculty table that is involved in our DataSet, we can confirm whether this fill is successful or not. If this property is equal to 0, which means that no matched record has been found in 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 non-

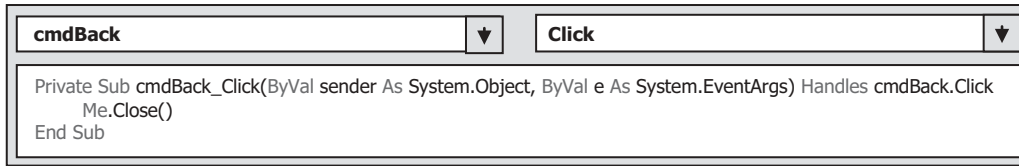


Figure 5.52. The codes for the Back button.

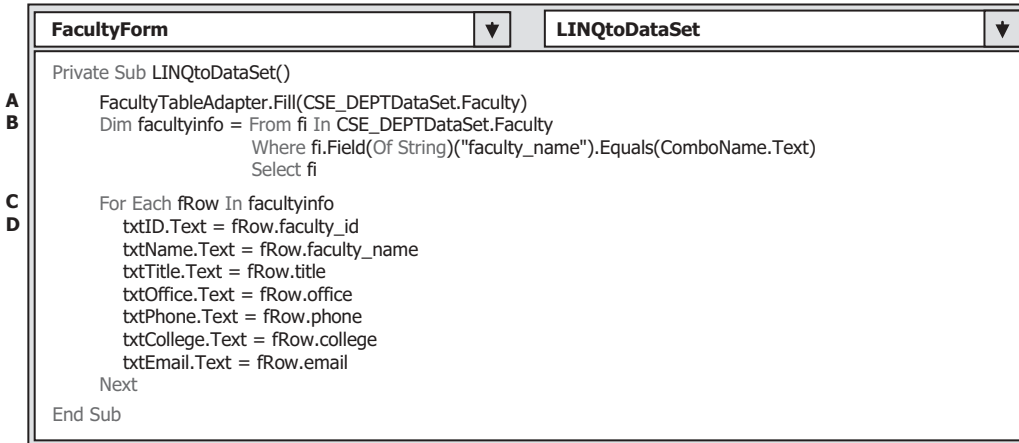


Figure 5.53. The codes for the LINQ to DataSet subroutine.

zero, which indicates that this fill is successful and a matched faculty name is found and the Faculty table in our DataSet has been filled. All information related to the matched faculty will be displayed in 7 textboxes and a picturebox.

As we mentioned, in this section, we only perform the coding for the **Select** and the **Back** buttons. The coding for all other buttons will be provided in the later sections.

The coding for the **Back** button is very simple. The Faculty form will be removed from the screen and from the project or from the memory when this button is clicked. A `Close()` method is used for this purpose, which is shown in Figure 5.52.

Next, we need to develop the codes to use LINQ to DataSet method to perform this faculty data query.

5.12.2 Develop Codes to Query Data Using the LINQ to DataSet Method

The faculty data query can be significantly integrated and improved by using the LINQ to DataSet technology. We have already provided a very detailed discussion about this technology in Chapter 4. Refer to that chapter to get a clear picture of this issue. In this part, we will concentrate on the coding for this subroutine.

Open the Code Window of the FacultyForm if it is not opened, create a user-defined subroutine, and enter the codes, which are shown in Figure 5.53, into this window.

Let's have a closer look at this piece of codes to see how it 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.
- B. A typical LINQ query structure is created and executed to retrieve all related information for the selected faculty member. The `facultyinfo` is an implicitly typed local variable. The Visual Basic.NET 2010 will be able to automatically convert this variable to a 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 an SQL SELECT statement is executed with the WHERE clause.
- C. A For Each 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 textbox to display it in the FacultyForm window.

One issue you may have found is that when you test this project, the related faculty picture is not displayed with those pieces of faculty information together. We try to solve this problem in the next section.

5.13 DISPLAY A PICTURE FOR THE SELECTED FACULTY

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 the project. We can load this picture into your project to show it as your project runs.

To display a 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 Basic.NET 2010 executable file is stored. The actual 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, we can 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 as you load them when the project runs.
- In order to select a correct faculty image based on the Faculty Name selected by the user, a function should be developed to complete this job.
- To display the image, a system method, `System.Drawing.Image.FromFile()`, is used.

First, let's take a look at the codes that need to be added into the **Select** button event procedure to select and display a matched faculty image.

5.13.1 Modify the Codes for the Select Button Event Procedure

Open the Faculty form window and double-click on the **Select** button to open its event procedure. Add the highlighted codes shown in Figure 5.54 into this event procedure.

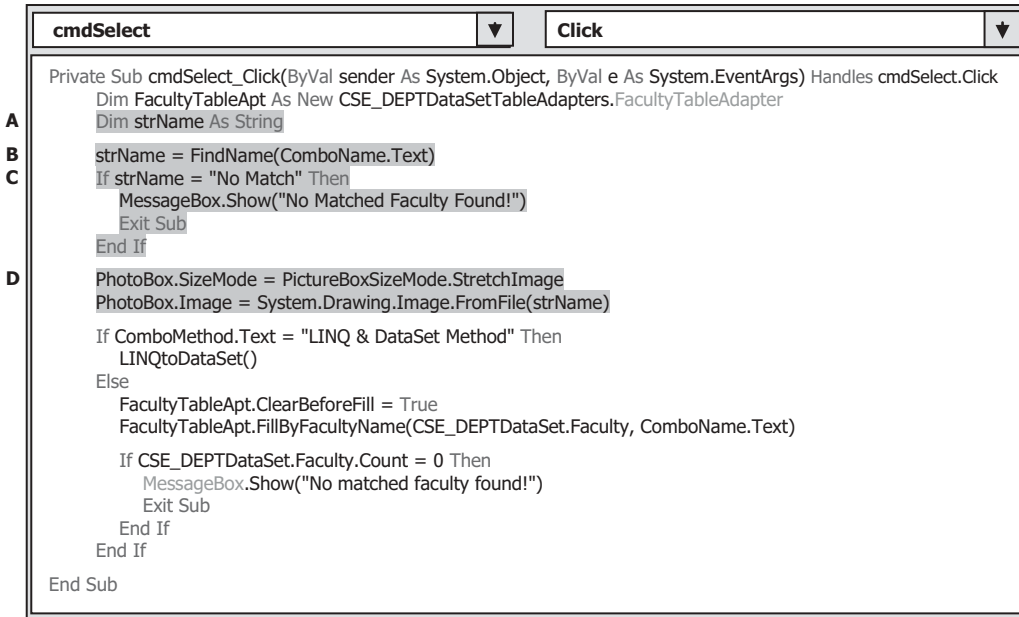


Figure 5.54. Add codes to select the faculty's image.

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

- A.** A local String variable `strName` is created to hold the name of the returned faculty image.
- B.** Call the function `FindName()` that will be developed later to identify and return the matched faculty image based on the input that is a selected faculty name.
- C.** If the function returns a `No Match` string, which means that no matched faculty image is found, a warning message will be given, and the program is directed to exit the application.
- D.** By setting the Picture's property `SizeMode` to the `StretchImage`, we allow the image to be enlarged enough to fill the whole `PictureBox`. Then the system method is called to load and display the image based on the selected image.

5.13.2 Create a Function to Select the Matched Faculty Image

Now let's develop a function to select the matched image for the faculty selected by the user. The input parameter should be a faculty name, and the output should be an image file's name, which is matched to the input faculty name.

Keep the Faculty form selected, and click on the View Code button from the Solution Explorer window to open its code window. Create a new function `FindName()` by entering the codes shown in Figure 5.55 into this code window.

This coding is straightforward. A local String variable `strName` is created to hold the selected image file name. The Select Case structure is used to choose the matched faculty image file. A string `No Match` is returned if no matched faculty image is found.

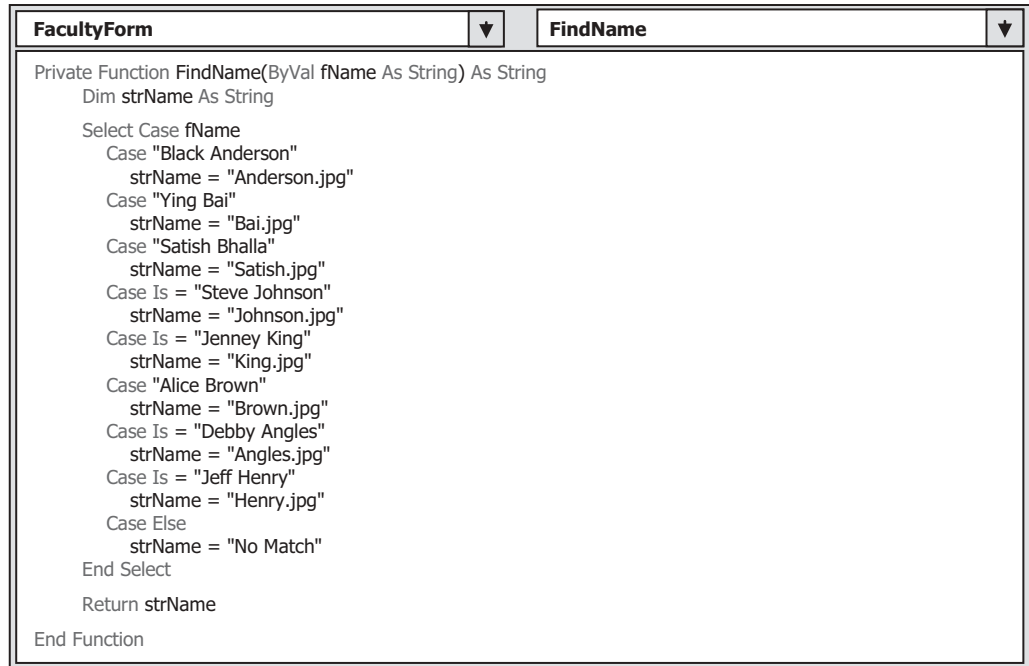


Figure 5.55. The codes for the function FindName.

Before we can run the project to test the faculty data query functions, one key point is that we must save all faculty image files into the folder in which our project executable file is located in order to make our project work properly. In this application, this folder is C:\SelectWizard\SelectWizard\bin\Debug. You can find all faculty and student image files from the folder **Images** located at the Wiley site (refer to Fig. 1.2 in Chapter 1). Copy and paste those image files to this folder.

Now we are ready to test our project. Click on the Start button to run the project. Enter jhenry as the username and test 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.

To perform this query using the TableAdapter Method, keep the default method in the Query Method combo box and 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 5.56.

To test querying the faculty data using the LINQ to DataSet method, select the LINQ to DataSet Method from the Query Method combo box. Then select a desired faculty name from the Faculty Name combo box and click on the Select button. You can find that the same query result can be retrieved and displayed in this form. Click on the Back button and then the Exit button to exit our project.

At this point, we complete the designing and building our Faculty form. Next we will take care of our Course form.



Figure 5.56. The running status of the Faculty form window.

5.14 QUERY DATA FROM THE COURSE TABLE FOR THE COURSE FORM

The functions of this form are illustrated as 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. All courses, that is, all `course_id`, are displayed in the Course ListBox.



In this example, we saved our faculty image file in the folder in which the project 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, which includes 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” when you call the function `FindName()`.

2. The detailed information for each course, such as the course title, course schedule, classroom, credits, and enrollment, can be obtained by clicking on 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 some other selections to obtain desired information related to those selections.

In this section, we only take care of two buttons; the **Select** and the **Back**, buttons, and the coding for the other buttons will be discussed in the later chapters.

5.14.1 Build the Course Queries Using the Query Builder

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 that is associated with the selected faculty from the Faculty Name combo box control when users click 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 can create another query in the Course table to find the courses (**course_id**) 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 combo box in the Course form.

1. Open the DataSet Designer Wizard and right-click on the last line of the Faculty table and select **Add Query** to open the TableAdapter Query Configuration Wizard.
2. Keep the default selection **Use SQL statements** and click on the **Next** button to go to the next wizard.
3. Check the radio button of **SELECT** which returns a single value to choose this query type, and click on the **Next** button to go to the next wizard.
4. Click on the **Query Builder** to build our query.
5. On the opened Query Builder wizard, remove the default query from the text pane or the third pane by highlighting it, right-clicking on it, and selecting the **Delete** button. Then right-click on the top pane and select the **Add Table** item to open the Add Table wizard. Select the Faculty table by clicking on it from the table list, and then click on the **Add** and the **Close** buttons to add this table.
6. Select the **faculty_id** and the **faculty_name** from the Faculty table by checking them in the top pane and unchecking the **Output** checkbox for the **faculty_name** row in the mid-pane since we do not want to query the **faculty_name** but only use it as the criterion to find the **faculty_id**.
7. Then type a question mark on the **Filter** column for the **faculty_name** row and press the Enter key from your keyboard. Your finished query should match the one that is shown in Figure 5.57.
8. The SQL statement shown in the text pane or the third pane is:

```
SELECT faculty_id FROM Faculty WHERE (faculty_name = @Param1)
```

9. Click on the **OK** buttons and the **Next** buttons to go to the next wizard. 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 second query to find the courses (**course_id**) 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.

1. Open the Data Source window by clicking the **DataShow 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.

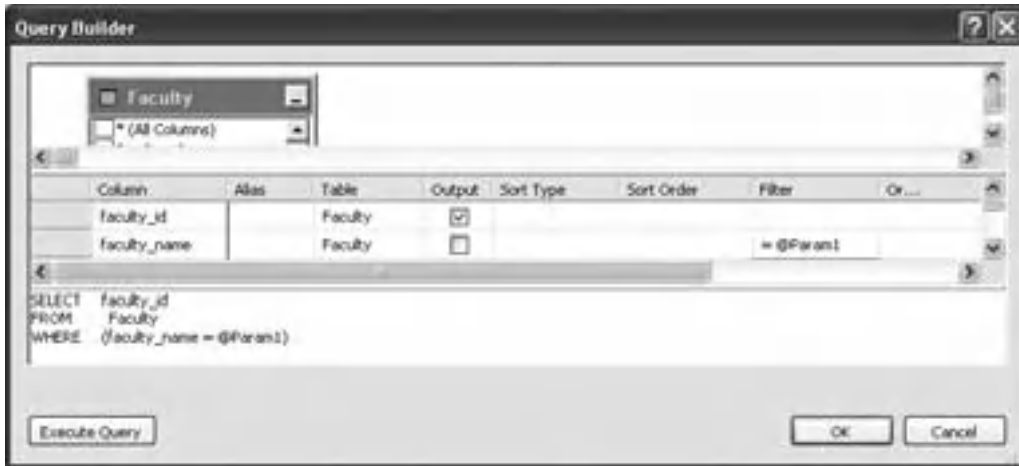


Figure 5.57. The finished query for the faculty_id.

2. Right-click on the last line of the Course table and choose the **Add Query** item to open the TableAdapter Configuration Wizard.
3. Keep the default selection **Use SQL statements** and click on the **Next** button to go to the next wizard. In the next wizard, keep the default selection **SELECT** which returns rows unchanged, and click on the **Next** button.
4. Then click on the Query Builder to open the Query Builder window, which is shown in Figure 5.58.
5. Keep the default selections for the top graphical pane even if we only need the **course_id** column, and we will show you why we need to keep these default items later. Go to the **Filter** column along the **faculty_id** row, and type a question mark (?) and press the Enter key from your keyboard. This is equivalent to set a dynamic parameter for this SQL SELECT statement.
6. 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)
```
7. The dynamic parameter @Param1 is a temporary parameter, and it will be replaced by the real parameter **faculty_id** as the project runs.
8. Click on the **OK** button and then the **Next** button to return to the TableAdapter Configuration Wizard to modify the **Fill()** method. Change the **Fill()** method to the **FillByFacultyID()**. Then click on the **Next** and the **Finish** button to complete this configuration.

The next step is to bind the controls from the Course form to the associated data column in the Course table in the DataSet. Select the **Course Form.vb** from Solution Explorer window and click on the **View Designer** button to open the Course form window.

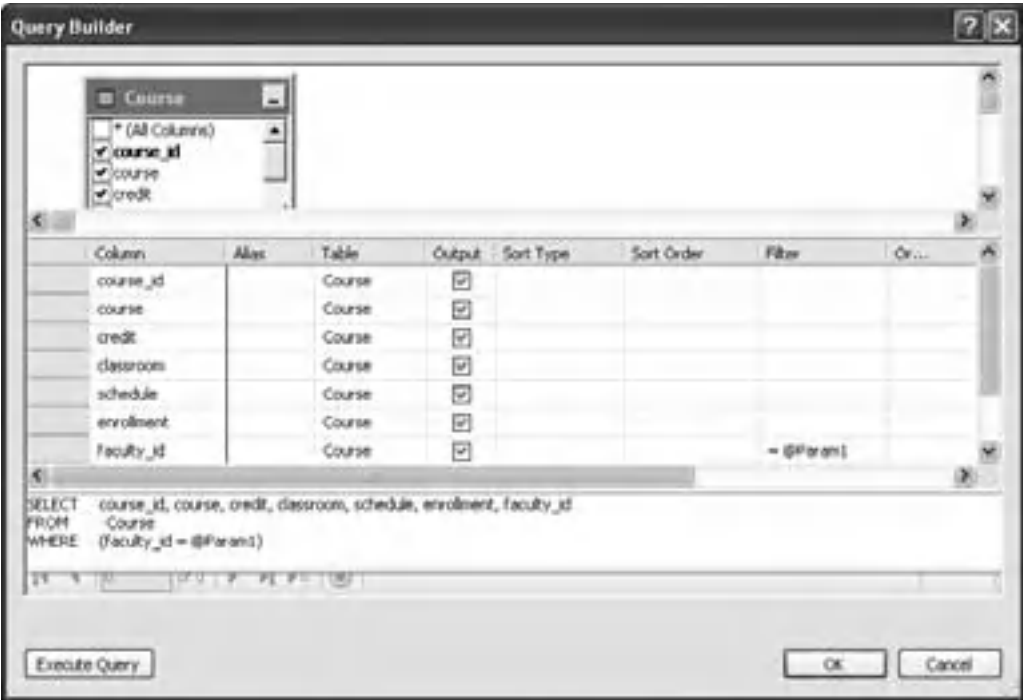


Figure 5.58. The Query Builder.

5.14.2 Bind Data Columns to the Associated Controls in the Course Form

First, we need to bind the CourseList to the `course_id` column in the Course table in the DataSet. Recall that there are many course records with the same `faculty_id` in this Course table when we build this sample database in Chapter 2. Those multiple records with the same `faculty_id` are distinguished by the different `course_id` taught by that faculty. To bind a ListBox to those multiple course records with the same `faculty_id`, we cannot continue to use the binding method we used before for 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 items 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_id`. So the binding of a ListBox control is to bind a ListBox to a table in the DataSet, that is, to the Course table in this application.

Perform the following operation to complete this binding:

1. Click on the CourseList control from the Course form, and go to the **DataSource** property. Then click on the drop-down arrow to expand the data source until the Course table is found. Select this table by clicking on it. Figure 5.59a shows this expansion situation.
2. Go to the **DisplayMember** property and expand the Course table to find the `course_id` column, and select it by clicking on this item (Figure 5.59b).

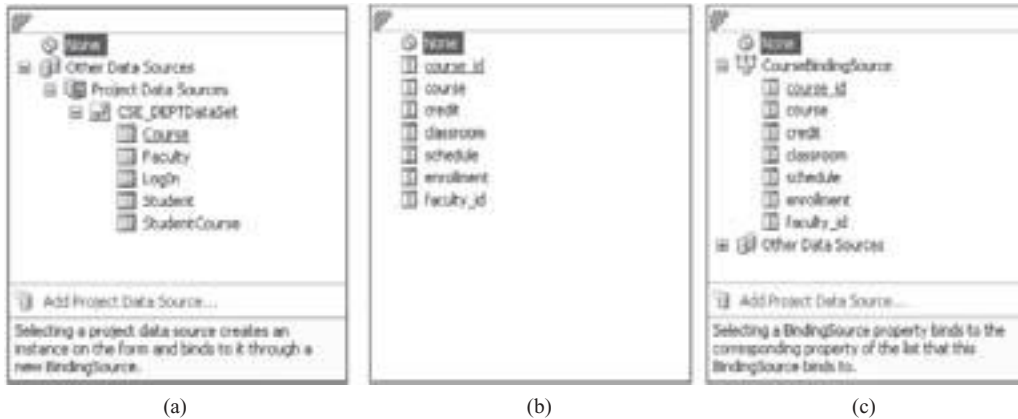


Figure 5.59. The expansion of the data source.

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.

Now, let's do the second binding; bind six textbox controls in the Course form to six columns in the Course data table in the DataSet. Perform the following operations to complete this binding:

1. Keep the Course form opened and then select the Course ID textbox from the Course form.
2. 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_id** column, which is shown in Figure 5.59c, and then choose it by clicking on the **course_id** column.

In this way, a binding is set up between the Course ID textbox in the Course form and the **course_id** column in the Course table in the DataSet.

Set up all other four data bindings for the following five textbox controls: Course, Schedule, Classroom, Credits, and Enrollment in a similar way.

One point you need to note is the order of performing these two bindings. You must first perform the binding for the CourseList control, and then perform the binding for six Textboxes.

Now we can answer the question why we need to keep the default selections at the top graphical pane when we build our query in the Query Builder (refer to Fig. 5.58). The reason for this is that we need those columns to perform data binding for our six 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 data table Course in the DataSet will be updated by the data columns in the Course data table in our real database each time a **FillByFacultyID()** method is executed. At the same time, all six textboxes' content will also be updated since those textbox controls have been bound to those data columns in the Course data table in the DataSet.

Ok, now it is time for us to create coding for this form.

5.15 DEVELOP CODES TO QUERY DATA FOR THE COURSE FORM

Based on the analysis of the functionality of the Course form we did above, when the user selected a faculty name and click on the **Select** button, all courses, that is, all `course_id` taught by that faculty, should be listed in the Course ListBox. To check the details for each course, click the `course_id` from the CourseList control, and all detailed information related to the selected `course_id` will be displayed in six 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 to DataSet method.

5.15.1 Query Data from the Course Table Using the TableAdapter Method

Open the code window of the Course form window. Click on the drop-down arrow from the Class Name combo box to select the (CourseForm Events) item, then click on the drop-down arrow from the Method Name combo box to select the Load to open the `CourseForm_Load` event procedure. Remove all original codes and enter the codes shown in Figure 5.60 into this event procedure.

The Add method is used to add all faculty names into the combo box. Resetting the `SelectedIndex` property to 0 is to select the first faculty and the first method as the default one from the combo box as the project runs.

Open the Course form window by clicking on the View Designer button from the Solution Explorer window, and then double-click on the **Select** button to open its event procedure. Enter the codes shown in Figure 5.61 into this event procedure.

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

- A. A new course table adapter object is created based on the `CourseTableAdapter` class that is located at the namespace `CSE_DEPTDataSetTableAdapters`.

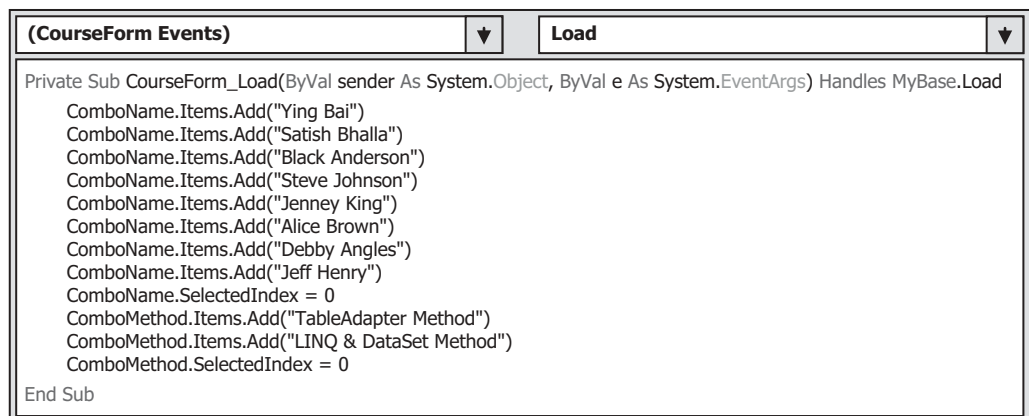


Figure 5.60. The codes for the `CourseForm_Load` event procedure.

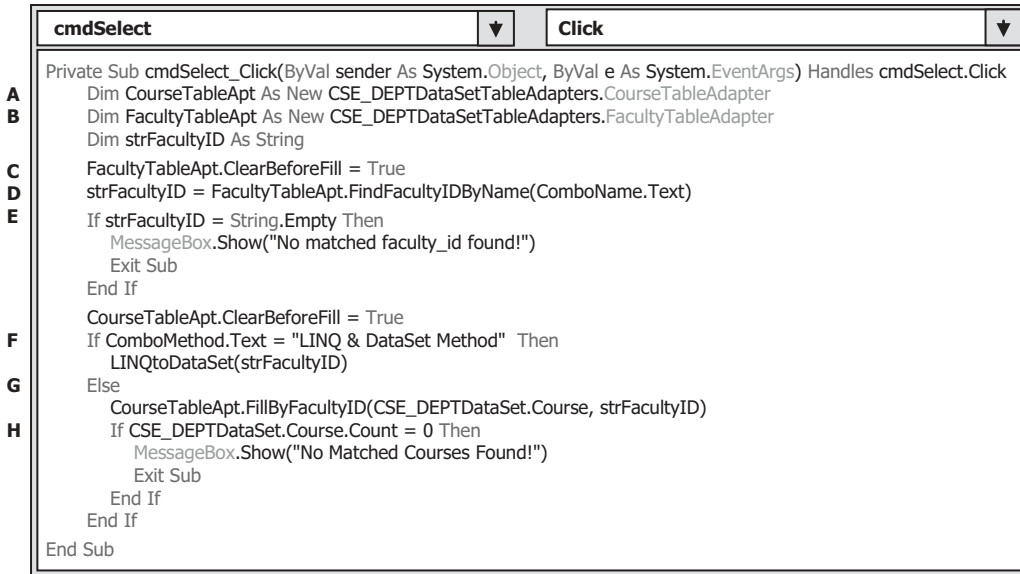


Figure 5.61. The codes for the Select button click event procedure.

- B.** A new faculty table adapter object is also created based on the FacultyTable-Adapter class. A local string variable `strFacultyID` is declared, and it is used to hold the returned `faculty_id` when our built query `FindFacultyIDByName()` is executed later.
- C.** Before the query `FindFacultyIDByName()` is executed, the faculty table adapter is first cleaned up.
- D.** The query `FindFacultyIDByName()` is called with an argument that is the faculty name selected by the user when the project runs. The returned `faculty_id` is assigned to the local string variable `strFacultyID`.
- E.** If the returned value is an empty string, which means that no matched `faculty_id` can be found and this calling has failed, an error message is displayed and the procedure is exited.
- F.** If the user selected the **LINQ to DataSet Method**, a user-defined subroutine `LINQtoDataSet()`, which will be built later, is called to perform this data query using the LINQ to DataSet method.
- G.** Otherwise, the TableAdapter method is selected by the user and the query we built in the DataSet Designer, `FillByFacultyID()`, will be called to fill the Course table in our DataSet using a dynamic parameter `@Param1` that is replaced by our real parameter `strFacultyID` now (refer to Fig. 5.58).
- H.** 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 data item is filled into our Course table in our DataSet, the fill has failed, and a warning message will be displayed to require users to handle this situation. Otherwise, the fill is successful, and all courses (`course_id`) taught by the selected faculty will be filled into the Course table and loaded into the Course ListBox control in our Course form, and furthermore, the detailed course information including the course ID, course schedule, classroom, credits, and enrollment for the selected `course_id` in the Course ListBox will be displayed in the six textbox controls since these textbox controls have been bound to those related data columns in the Course table.

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 event procedure and enter the code, **Me.Close()**, into this event procedure.

Next, let's handle the coding for the querying data using the LINQ to DataSet method.

5.15.2 Query Data from the Course Table Using the LINQ to DataSet Method

In the last coding (refer to Fig. 5.61), the project will be directed to calling the **LINQtoDataSet()** subroutine if the user selected the LINQ to DataSet method from the Query Method combo box. Refer to Chapter 4 to get a detailed discussion about the data query between LINQ to DataSet. In this part, we will develop the codes to use this subroutine 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 subroutine **LINQtoDataSet()**, and enter the codes, which are shown in Figure 5.62, into this subroutine.

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

- A.** First, the default **Fill()** method of the **CourseTableAdapter** is executed to load data from the Course table in the database into the Course 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.
- B.** A typical LINQ query structure is created and executed to retrieve all related information for the selected **faculty_id**. The **courseinfo** is an implicitly typed local variable. The Visual Basic.NET 2010 will be able to automatically convert this variable to a 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 an SQL **SELECT** statement is executed with the **WHERE** clause.

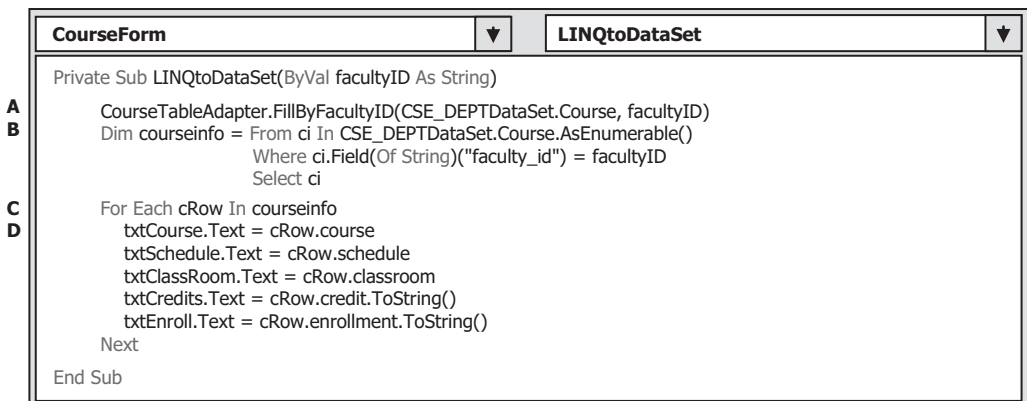


Figure 5.62. The codes for the LINQ to DataSet subroutine.

- C. A For Each 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.
- D. Assign each column to the associated textbox to display it in the CourseForm window.

That's it! All coding is done.

Let's test our project by running it. Click on the Start button to run our project. Complete the login process and select the Course Information from the Selection form. Then click on the OK button to open our Course form, which is shown in Figure 5.63.

On the opened Course form, select the default faculty name **Ying Bai** and keep the default query method **TableAdapter Method** from the Query Method combo box, and click on the **Select** button to test this data query using the TableAdapter method. The filled `course_id` are displayed in the Course ListBox, as shown in Figure 5.63.

Now let's go one more step forward by just clicking on a `course_id` from the Course ListBox. Immediately, the detailed information about that selected `course_id`, including the course, schedule, classroom, credits, and enrollment will be displayed in the six textbox controls. This makes sense since those textbox controls have been bound to those six associated columns in the Course table in our DataSet. As you click one `course_id` from the Course ListBox, in effect, 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, six columns of that record have been bound to the six textbox controls in the form, so the data related to those columns will also be reflected on these six textbox controls. These relationships can be represented and illustrated by connections shown in Figure 5.64.

It is very interesting, is not it?

Figure 5.63. The running status of the Course form.

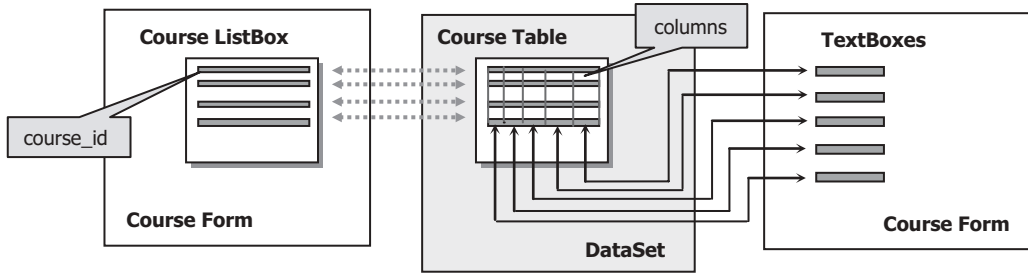


Figure 5.64. The relationships between Course ListBox, Course table and TextBox.

Yes! This is the power provided by Visual Basic.NET. By using those Tools and Wizards in Visual Studio.NET, it is very easy to develop a professional database programming in the Visual Basic.NET environment, and it becomes fun to develop a database programming in Visual Basic.NET 2010.

You can select the LINQ to DataSet Method from the Query Method combo box to test the course data query using that method. The same query results can be obtained and displayed in the CourseForm.

We have the last form, which is the Student form, and we want to leave this as the homework for students to allow them to finish the building and development of 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 Solution** that contains the coding for Student form has been developed, and you can find this project from the folder DBProjects\Chapter 5 that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1). The database used for that project is Microsoft Access 2007.

A completed project **SelectWizard**, including the source codes, GUI designs, Data Source, and Query Builders, can be found in the folder DBProjects\Chapter 5 that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

5.16 QUERY DATA FROM ORACLE DATABASE USING DESIGN TOOLS AND WIZARDS

Basically, there is no significant difference between building a Visual Basic.NET project to query data from an SQL Server or from an Oracle database using the Design Tools and Wizards provided by the earlier versions of .NET Framework and Visual Studio.NET, such as .NET Framework 2.0 and 3.5, as well as Visual Studio.NET 2005 and 2008. However, a significant change occurred starting from .NET Framework 4.0 and Visual Studio.NET 2010. Starting from .NET Framework 4.0 and Visual Studio.NET 2010, Microsoft no longer supports the Data Provider for Oracle database. Therefore, we cannot use those tools and wizards to build any data applications to access and manipulate data against Oracle databases in Visual Studio.NET 2010.

In order to solve this problem, in this section, we selected a third-party product, dotConnect for Oracle 6.30 Express developed by Devart™ Inc.

5.16.1 Introduction to dotConnect for Oracle 6.30 Express

dotConnect for Oracle Express Edition, formerly known as OraDirect.NET, is an enhanced ORM-enabled data provider for Oracle that builds on ADO.NET technology to present a complete solution for developing Oracle-based database applications.

dotConnect for Oracle 6.30 Express is a powerful tool that enables users to design and build professional database applications to access Oracle databases in .NET environment. This tool provides full support to the updated versions of Oracle Database Express Editions, such as 11g XE, and sets up connections between the Oracle database and .NET Framework, especially ADO.NET, which provides simple and convenient ways to access and manipulate Oracle Database XE from the Visual Studio.NET environment.

Some important and useful properties of using dotConnect for Oracle 6.30 Express to access Oracle Database 11g XE are:

- Direct access from .NET to Oracle database server without Oracle client
- 100% managed code
- High performance and easy to deploy
- Supports latest versions of Oracle server, including Personal and Express editions
- All Oracle data types support and full Oracle Objects support Compatible with ADO.NET Entity Framework v.1 and v.4

In the following sections, we will use the tools and wizards provided by dotConnect for Oracle 6.30 Express to build our database applications to access our sample Oracle database CSE_DEPT.

Appendix F provides a detailed discussion and introduction about downloading and installing the dotConnect for Oracle 6.30 Express. Refer to that Appendix to complete the downloading and installing this product.

When using the tools and wizards provided by dotConnect for Oracle 6.30 Express to build our Oracle database applications, the only small difference is the creation and connection to the different data sources when you select the data source for your applications. All other stuffs, including the codes, GUIs, and query building, are identical and can be used mutually. For this reason, in this part, we want to use a sample project, **SelectWizardOracle**, to illustrate how to perform data queries against an Oracle database using the Design Tools and Wizards. We will mainly concentrate on the data source selection and connection to an Oracle sample database CSE_DEPT, which was built in Chapter 2 using the Oracle Database 11g XE.

5.16.2 Create a New Visual Basic.NET Project: SelectWizardOracle

Now let's create a new Visual Basic.NET Windows project **SelectWizardOracle** and connect it to our Oracle database CSE_DEPT we built in Chapter 2.

Perform the following operations to create this new project:

1. Open Visual Studio 2010 and go to the **File|New Project** menu item to open the New Project wizard.
2. Expand the **Other Project Types** item under the **Installed Templates** and select the **Visual Studio Solutions**
3. Enter **SelectWizardOracle Solution** into the Name box and click on the **Browse** button to find a desired folder to save this new solution. Then click on the **OK** button to create this blank solution.
4. Right-click on the newly created solution **SelectWizardOracle Solution** from the Solution Explorer window and select the **Add|New Project** item.
5. Select **Visual Basic|Windows Form Application** from the Templates pane. Enter **SelectWizardOracle** into the Name box as the name for this project, and click on the **OK** button to create this new project.

Refer to Sections 5.3.1.1–5.3.1.5 to build five GUIs: **LogIn**, **Selection**, **Faculty**, **Course**, and **Student**. To save time, you can also copy and paste all of these five forms from the folder **VB Forms\Window** that is located at the Wiley site (refer to Fig. 1.2 in Chapter 1). Following the operational procedure described below to complete this copy and paste actions:

1. Open the Windows Explorer and create a new folder in your computer, such as **SelectWizardOracle Forms**.
2. Go to the folder **VB Forms\Window** located at the Wiley site to copy all files from that folder and paste them to your new folder **SelectWizardOracle Forms**.
3. On your opened Visual Basic.NET project, go to the **Project|Add Existing Item**.
4. Browse to your folder **SelectWizardOracle Forms** and select all five form files, **LogIn Form.vb**, **Selection Form.vb**, **Faculty Form.vb**, **Course Form.vb**, and **Student Form.vb**, by using the **Ctrl** key on your keyboard.
5. Click on the **Add** button to add all five forms to your project.

You can also replace the default form **Form1.vb** with the **LogIn Form.vb** and use it as your start up form. Perform the following operations to complete this replacement action:

1. Delete the default Form, **Form1.vb**, from your project by right-clicking on this form from the Solution Explorer window, and select the **Delete** from the pop-up menu.
2. Go to the **Build|Rebuild SelectWizardOracle** menu item to rebuild your project. The purpose of this operation is to find the mismatched default form, **Form1.vb**, from your project and replace it with your **LogIn Form.vb**.
3. Go to the **Error List** in the **Output** window, double-click on the error line **"Form1" is not a member of "SelectWizardOracle"** to open the **Application.Designer.vb** file.
4. On the opened file, locate the **Form1**, and replace it with the **LogInForm**. Your finished modification should look like:

```
Me.MainForm = Global.SelectWizardOracle.LogInForm
```

5. Rebuild and run your project, and you can find that now, the **LogIn Form** works as the start up form in your project.

Now we need to select the desired Oracle data source and connect it to our project.

5.16.3 Select and Add Oracle Database 11g XE as the Data Source

Perform the following operations to add our Oracle sample database CSE_DEPT into our current Visual Basic.NET project:

1. In the opened **SelectWizardOracle** project, open the Data Source window by selecting the **Data>Show Data Sources** menu item.
2. Click on the **Add New Data Source** link to open the Data Source Configuration Wizard.
3. Keep the default **Database** selection in the **Choose a Data Source Type** wizard and the **Dataset** selection in the **Choose a Database Model** wizard unchanged, and click on the **Next** button to open the **Choose Your Data Connection** wizard.
4. Click on the **New Connection** button since we need to create a new connection between our project and the Oracle database.
5. Click on the **Change** button for the **Data Source** box to open the **Change Data Source** wizard, which is shown in Figure 5.65, and then select the **Oracle Database** from the Data source listbox and **dotConnect for Oracle** from the Data provider combo box, as shown in Figure 5.65, as our new data source. Click on the **OK** button to return to the **Add Connection** wizard.
6. Enter the following items into the associated boxes for our Oracle database:
 - Server name: XE
 - User Id: CSE_DEPT
 - Password: reback

Recall that in Chapter 2, we built an Oracle customer database named CSE_DEPT with the password **reback** using Oracle Database 11g XE. 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 the one that is shown in Figure 5.66.

7. Click on the **Test Connection** button to confirm this database connection. A **Connection succeeded** message should be displayed if the connection is fine. Click on the **OK** button to go to the next wizard.

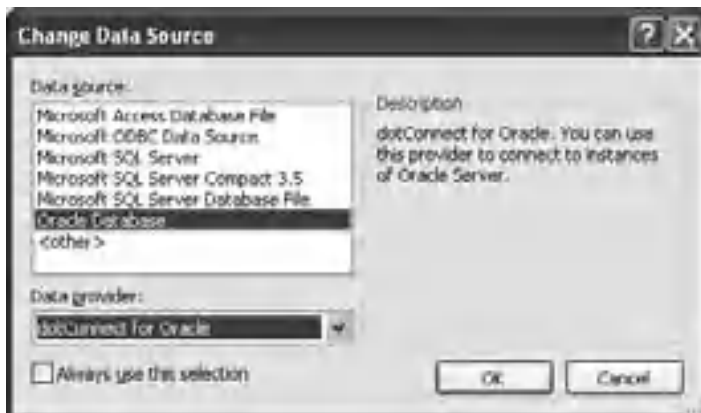


Figure 5.65. The Change Data Source wizard.



Figure 5.66. The Add Connection wizard.

8. The next wizard allows you to check and confirm the connection string. Click on the **Yes** radio button will allow us to add the username and password into this connection string to make this connection as an integrated body (Fig. 5.67).
9. Click on the **Next** button to open the next wizard. Change the name of this connection string to **ConnOracle** and click on the **Next** button.
10. The next wizard 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 built-in data tables may be selected, as our data table objects and most of them are built by the database vendors. For our application, we only need five 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 them one by one.
11. 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 wizard should match the one that is shown in Figure 5.68.
12. Click on the **Finish** button to complete this database connection. Immediately, you can find that five tables have been added into our **Data Source** window, which is shown in Figure 5.69.



Figure 5.67. The data source configuration wizard.



Figure 5.68. The database object selection wizard.

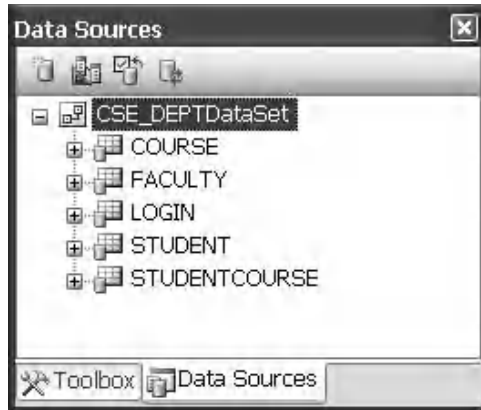


Figure 5.69. The added five data tables.

Next, let's take care of the small differences that exist in the coding parts between the SQL Server and the Oracle databases.

5.16.4 Modify the Codes to Access the Oracle Database

After this data source connection is complete, you can use all codes we developed in the project **SelectWizard** in this project. The following issues must be paid special attention in order to successfully develop this project by using the codes from the project **SelectWizard**:

- When you perform the copy and paste operations for those codes located inside different event procedures, you need first to open those event procedures from each form in our current project, **SelectWizardOracle**, by double-clicking on the associated button, and then you can copy the related codes from the associated event procedure in the project **SelectWizard** and paste them into the event procedure in our current project **SelectWizardOracle**. In other words, you cannot copy and paste the whole body of those event procedures, including the event procedure's header and ender, but you can copy and paste only the contents of each event procedure.
- 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 following:
 - **PasswordQuery()** and **FillByUserNamePassWord()** for **LogIn** form
 - **FillByFacultyName()** and **FindFacultyIDByName()** for the **Faculty** form, and
 - **FillByFacultyID()** for the **Course** form
- You need to change all the data tables' names and all the data columns' names in five tables (**LogIn**, **Faculty**, **Course**, **Student**, and **StudentCourse**) to the uppercase in your program codes since the Oracle database engine changed all of those names to uppercase when this new database is generated. Otherwise, you may encounter some debug errors when you build your project.
- 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 Wiley ftp site (refer to Fig. 1.2 in Chapter 1). The general folder used

to save those image files is the folder in which your executable Visual Basic.NET project file is located, such as `C:\Chapter 5\SelectWizardOracle\bin\Debug`.

- Remove all default codes, not the codes created by us, from each `Form_Load` event procedure, such as `LogInForm_Load()`, `FacultyForm_Load()`, `CourseForm_Load()`, and `StudentForm_Load()`, since those codes are created by the system, and we do not need those codes in our applications.

In addition to points listed above, you also need to perform the following operations to make your project work properly:

For `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 `Faculty` form:

1. Bind seven textbox controls, `txtID`, `txtName`, `txtTitle`, `txtOffice`, `txtPhone`, `txtCollege`, and `txtEmail`, to the corresponding seven columns in the `Faculty` table using the `FACULTYBindingSource`.
2. Build the query methods `FillByFacultyName()` and `FindFacultyIDByName()` using the Query Builder.

For `Course` form:

1. Bind six textbox controls, `txtID`, `txtCourse`, `txtSchedule`, `txtClassRoom`, `txtCredits`, and `txtEnroll`, to the corresponding six 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 `DBProjects\Chapter 5` that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1). This project contains query methods, binding objects, and full codes 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 programs we developed in Part I, in which a quite of few design tools and wizards provided by Visual Studio.NET are utilized to help us to finish those developments such as the `DataSet`, `BindingSource`, `BindingNavigator` and `TableAdapter`, the sample project developed in this part has nothing to do with those

tools and wizards. This means that we create those ADO.NET 4.0 objects by directly writing Visual Basic.NET 2010 codes without the aid of Visual Studio.NET design-time wizards and tools, as well as the auto-generated codes. All data-driven objects are created and implemented during the period of the project runs. In other words, all those objects are created dynamically.

The shortcoming of using those Visual Visual.NET tools and wizards to create data connections is that the auto-generated connection codes related to tools and wizards are embedded into your programs, and those connection codes are machine-dependent. Once that piece of connection information in your programs is compiled, it cannot be modified. In other words, your programs cannot be distributed to and run in other platforms.

Compared with tools and wizards, there are some advantages of using the runtime objects to make the data operations for your Visual Basic.NET 2010 project. One of the most important advantages is that it provides programmers more flexibility in creating and implementing connection objects and data operation objects related to ADO.NET and allows you to use different methods to access and manipulate data from the data source and the database. But anything has both a good and bad side, and it is 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 the last part, 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. But before we can continue to do that, a detailed understanding of the connection and data operations classes is very important, and those classes are directly related to ADO.NET. Although some discussions have been provided in Chapter 3, we will make a more detailed discussion for this topic in this section in order to make readers have a clear picture about this issue.

5.17 INTRODUCTION TO RUNTIME OBJECTS

The definition of runtime objects can be described as: objects or instances used for data connections and operations in a data-driven application are created and implemented during the period your project runs; in other words, those objects are created and utilized dynamically. To understand what kind of objects are most popularly used in an application, let's first have a detailed discussion about the most useful classes provided by ADO.NET.

According to Chapter 3, 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 5.70.

Data Provider contains four components

1. Data Connection
2. Data Command
3. DataReader
4. TableAdapter

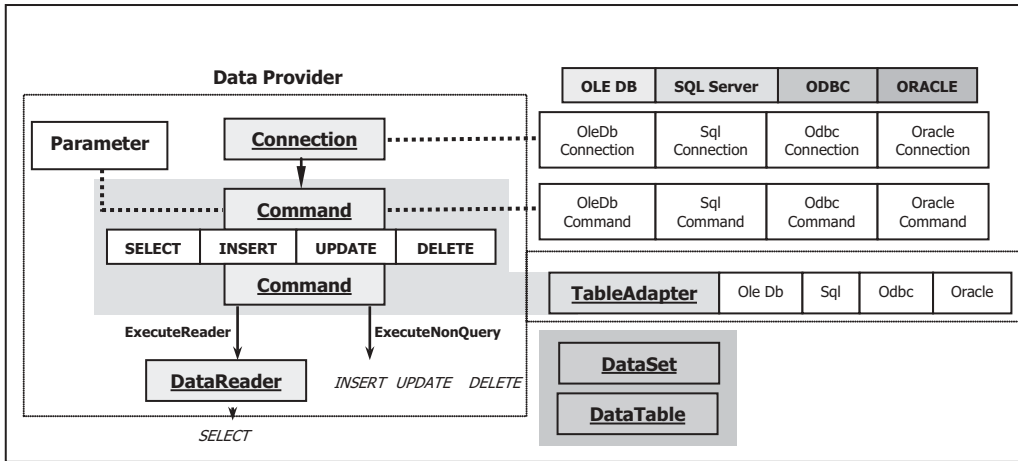


Figure 5.70. Classes provided by ADO.NET.

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

- Data Connection component: **SqlConnection**
- Data Command component: **SqlCommand**
- DataAdapter (TableAdapter): **SqlDataAdapter** (**SqlTableAdapter**)
- DataReader components: **SqlDataReader**

The same definition is needed for the other 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 5.70, four data providers are popularly used in database programming in Visual Basic.NET 2010. You must create the correct connection object based on your real database by using the specific prefix.

But two components in ADO.NET are Data Provider-independent, **DataSet** and **DataTable**. These two components are located at the **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 different methods. Among those 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 of a DataAdapter, such as the SelectCommand, InsertCommand, Update-Command, or DeleteCommand, when one develops a data-driven application using this method. As shown in Figure 5.70, when you use this method, the Command and Parameter objects are embedded or attached to the TableAdapter 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 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.

The second method is to allow 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 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 newly created Command object. You can fill any DataTable by calling the ExecuteReader() method to a DataReader object; also, you can perform any data manipulation by calling the ExecuteNonQuery() method to the desired DataTable.

In this section, we provide three sample projects to cover two methods: AccessSelectRTOobject, SQLSelectRTOobject, and OracleSelectRTOobject, which are associated with Microsoft Access 2007, Microsoft SQL Server 2008, and Oracle 11g XE databases.

To understand better for these two methods, we need to have a clear 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 Object

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 Basic project, you need follow the sequence listed below:

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 this database connection with the correct block, such as the Try . . . Catch block.
4. Create a new TableAdapter (DataAdapter) object.
5. Create a new DataTable object that is used to 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 Basic.NET 2010 form.

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 object for Microsoft Access 2007 database.

5.18 QUERY DATA FROM MICROSOFT ACCESS DATABASE USING RUNTIME OBJECT

The Microsoft Access 2007 database file used in this sample project is CSE_DEPT.accdb, and it is located at the Database\Access folder in the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

First, we need to create a Visual Basic.NET 2010 Windows-based project named AccessSelectRTOObject with five form windows: LogIn, Selection, Faculty, Course, and Student. Refer to Section 5.3.1 to build these form windows if you like. But in order to save your time, you can also copy and paste all of these five forms from the folder VB Forms\Window that is located at the Wiley site (refer to Fig. 1.2 in Chapter 1). Follow the operational procedure described in Section 5.16.2 to complete this copy and paste actions.

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 the LogIn Form

In this application, we want to use two methods to perform the data query from our LogIn table: the DataSet-TableAdapter method and the DataReader method. Therefore, we need to modify the LogIn Form by replacing the cmdLogIn button with two new buttons, TabLogIn and the ReadLogIn. Add these two new buttons with the Name and Text properties equaling to TabLogIn and ReadLogIn. Your modified LogIn form should match the one that is shown in Figure 5.71.

Now click on the View Code button to open its Code Window to begin our coding for these two event procedures to perform data query from the LogIn table with two



Figure 5.71. The modified LogIn form window.

different methods. But first, we need to create and declare some global variables and runtime objects.

5.18.1.1 Declare Global Variables and Runtime Objects

Starting from Visual Basic.NET 2010, the Module class is resumed by Microsoft. By using this class, we can declare some global variables used by the whole Visual Basic.NET project. One candidate for this kind of global variables is the connection object, since it will be used by our whole project.

First, let create a new module `ConnModule` by performing the following operations:

1. Go to the Project!Add Module menu item to open the Add New Item wizard.
2. Go to the Name box and change the module's name to `ConnModule.vb` and click on the Add button.
3. In the opened module code window, enter the codes shown in Figure 5.72.

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 database file, you need to use this Data Provider. You must first declare this namespace at the top line of your code window to allow Visual Basic.NET 2010 to know that you want to use this specified Data Provider.

The `Imports System.Data` is to provide 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 are located at this namespace, and those components will be used in this project later, so you must first provide the reference to that namespace to allow the Visual Basic.NET 2010 to access it.

The single code line is used to declare a new global instance of the `OleDbConnection` classes.

Public accConnection as OleDbConnection

The accessing mode `Public` makes this connection instance `accConnection` a global object, and it can be accessed by all event procedures defined in all different forms from this project.

After a connection object is declared, the next job is to connect your project with the database you selected.

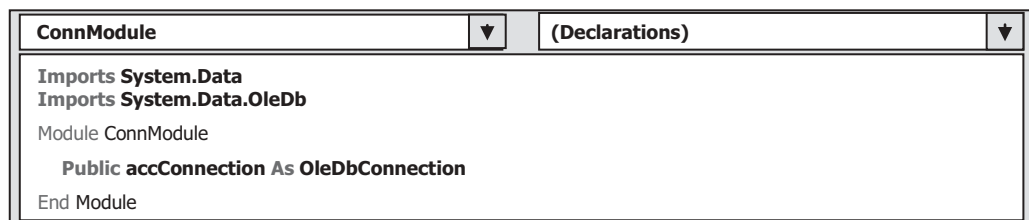


Figure 5.72. The declaration of the namespace for the OleDb Data Provider.

5.18.1.2 Connect to the Data Source with the Runtime Object

Because the connection job is the first thing you need to do before you can make any data query, you need to code the connection job in the first event procedure, the `Form_Load()` event procedure, to allow the connection to be performed first as your project runs.

In the code window, click on the drop-down arrow in the Class Name combo box and select the item (**LogInForm Events**). Then go to the Method Name combo box and click on the drop-down arrow to select the **Load** method to open the `LogInForm_Load()` event procedure, and enter the codes shown in Figure 5.73 into this event procedure.

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

- A.** Two namespaces related OleDb data providers are imported first since we need to use some components involved in that provider.
- B.** 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 connection information, including the name of the data provider, the location, and the name of the database, username, and password to access the selected database. In our case, no username and password are utilized for our database, so those two items are missed from this connection string. You need to add those two pieces of information if your database did utilize those two items. The database driver for Microsoft Access 2007 is Microsoft ACE OLEDB 12.0, and our database file is named `CSE_DEPT.accdb` that is located at `C:\Database` directory in our computer.
- C.** By using the keyword `New`, we initialize a new instance of the connection class `OleDbConnection` with the connection string we built in step **B**.
- D.** A Try . . . Catch block is utilized here to try to catch up any mistake 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 process and simplify this debug procedure.

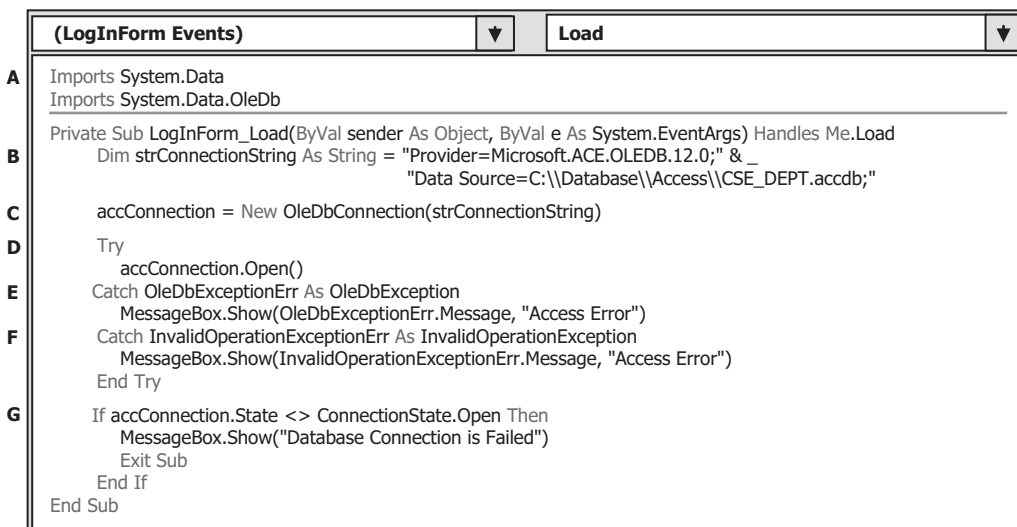


Figure 5.73. The codes for the `LogInForm_Load` event procedure.

- E. An `OleDbExceptionError` message will be displayed if an error related to the OleDb connection occurred.
- F. An `InvalidOperationExceptionError` message should be displayed if an invalid operation error were encountered.
- G. 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, next, 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-TableAdapter to Query Data

In this section, we discuss how to create and use the runtime objects to query data by using the DataSet-TableAdapter method.

Open the LogIn form window by clicking on the View Designer button, and then double click on the `TabLogIn` button to open its event procedure. Enter the codes shown in Figure 5.74 into this event procedure.

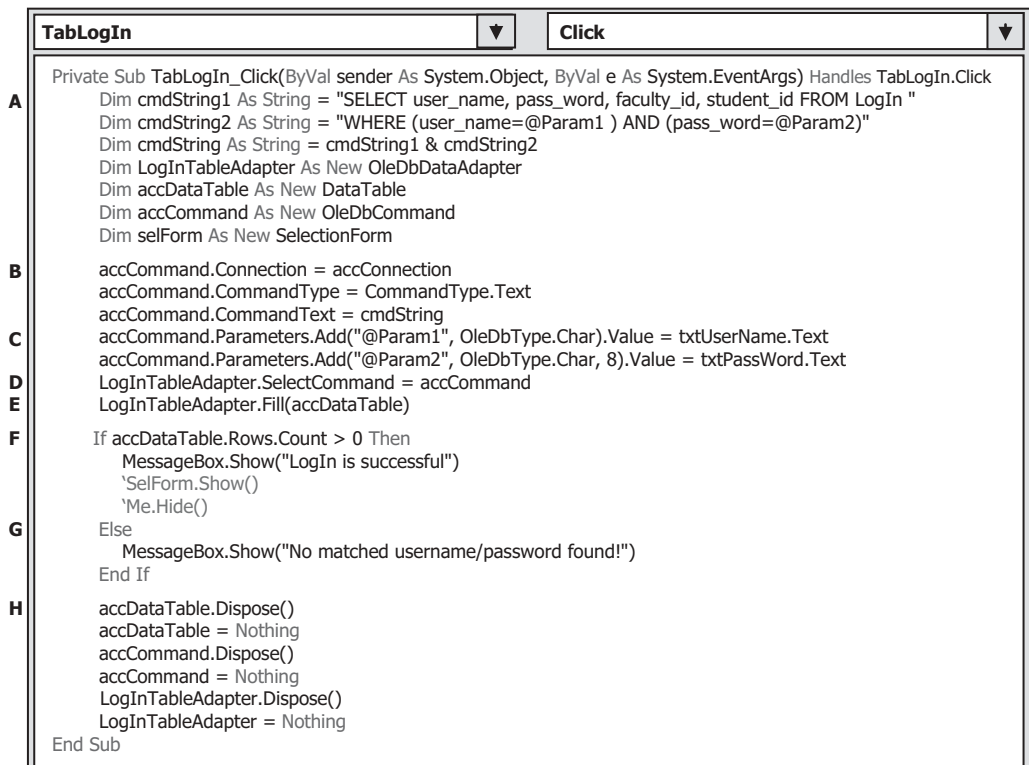


Figure 5.74. The codes for the TabLogIn button.

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

- A. Since the query string applied in this application is relatively long, we break it into two substrings: `cmdString1` and `cmdString2`. Then we combine these two substrings together to form a complete query string. A trick issue is existed when you break a long query string into multiple substrings in Visual Basic.NET, which is that you cannot break a long query string into multiple substrings by using the line breaker symbol (space + underscore) directly since Visual Basic cannot recognize a string that is broken into multiple lines. A string variable must be represented by a single string line in the Visual Basic programming. Another trick 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 would be encountered if you did not pay attention to this space, and this bug is not easy to find and correct.
- B. The Command object `accCommand` is initialized by using three properties: connection object, command type, and command string.
- C. Note that there are two dynamic parameters, `@Param1` and `@Param2`, in the second query string, and these two parameters need to be replaced by two real values that will be entered by the user via two textboxes, `txtUserName` and `txtPassWord`, 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, and 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`.
- D. 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 `TableAdapter`.
- E. The `Fill()` method of the `TableAdapter` is called to fill the LogIn table. Exactly when the `Fill()` is called, the `SelectCommand()` is executed to perform the query we built in the previous steps.
- F. 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 filled into the LogIn table. Note that the following two lines of codes that have been commented out will be used later for our normal project. The purpose of these two lines of codes is to display the next form window—Selection form—and hide the current form window—LogIn form—if the login process is successful. But right now in order to test our project, a message box is used. Later on, we need to use these two green color codes to replace the message box as our final codes.
- G. An error message will be issued if this property is 0, which means that no row or record is filled into the LogIn table, and the program is exited.
- H. A cleaning job is necessary to release all objects we used for this data query. This cleaning includes the `DataTable`, `TableAdapter`, and `Command` objects. A `Dispose()` method and a `Nothing` property are used to finish this cleaning job.

Now, let's take a look at the codes for the second method.

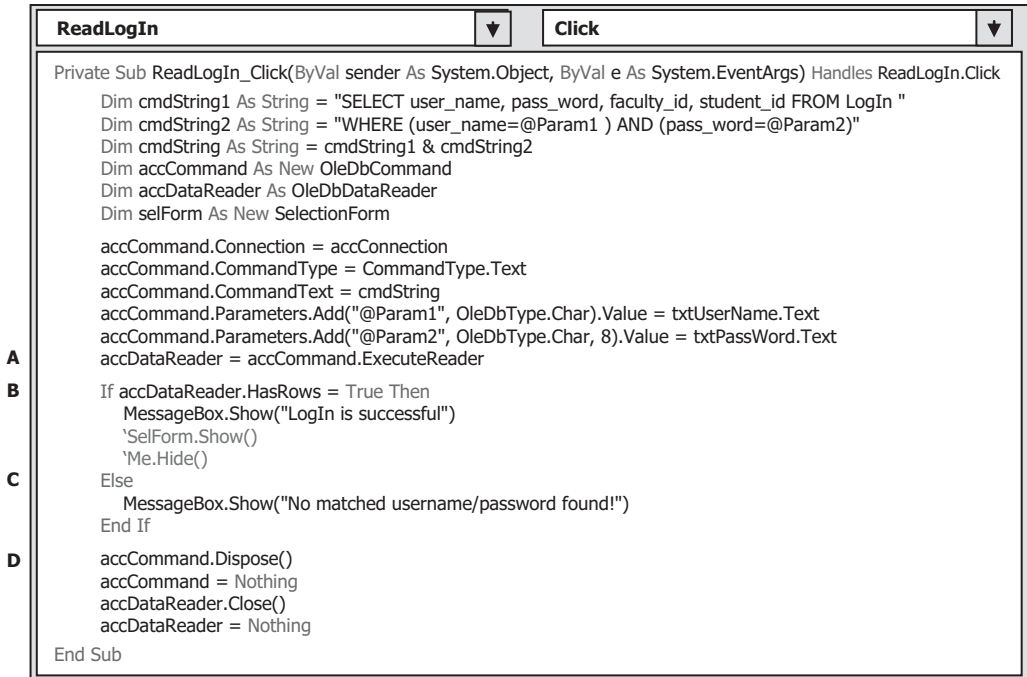


Figure 5.75. The codes for the ReadLogIn button event procedure.

5.18.1.4 Coding for Method 2: Using the 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 event procedure. Enter the codes shown in Figure 5.75 into this event procedure.

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

Most codes in the top section are identical with those codes in the TabLogIn button's event procedure with two exceptions. First, a **DataReader** object is created to replace the **TableAdapter** to perform the data query, and, second, the **DataTable** is removed from this event procedure since we do not need it for our data query in this method.

- 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 success message will be displayed if this property is **True**, which means that the **DataReader** has received the data from the LogIn table. The codes that have been commented out will be used later to replace this message box function for our normal project. But at this moment, we use this message box to test our project.
- 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.

5.18.1.5 Clean up the Objects and Terminate the Project

Before we can test our project, we need to finish the coding for our last button, **Cancel**. The purpose for this coding is to clean up the objects used in this form and terminate our project. 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 event procedure. Enter the codes shown in Figure 5.76 into this event procedure.

To release the Connection object, which is a global variable, a **Close()** method is called first. Then the **Dispose()** method and the **Nothing** property is used to finish this release. Finally, the system method **Close()** is called to close the whole project. The keyword **Me** represent the current form window—the LogIn form. Please note that the Connection instance would not be released if this **Cancel** button was not clicked. In the normal case, we still need to use this Connection object for the following data queries if the login process is successful.

It is time for us to test our project. Click on the Start button to begin our project. Enter **jhenry** and **test** as the username and the password into the two textboxes on the LogIn form window, and 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 5.77a.

Click on the OK button to the MessageBox, and then click on the **ReadLogin** button to test our data query by using the DataReader method. Similarly, a successful message is displayed, which is shown in Figure 5.77b. You can try to enter other correct usernames

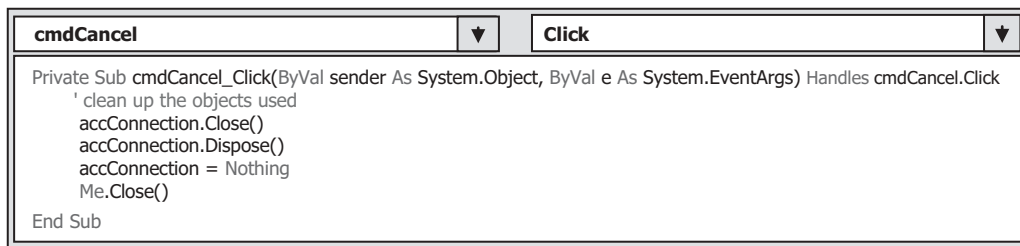
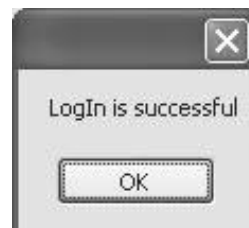


Figure 5.76. The codes for the Cancel button event procedure.



(a)



(b)

Figure 5.77. The running status of the project.

and passwords to test the project, and even to enter some wrong usernames or passwords to see what will be happened.

Click on the **Cancel** button to terminate the project.

Our project is successful! But before we can move on, we need to replace the successful message with two lines of green-color coding: `selfForm.Show()` and `Me.Hide()` for both event procedure `TabLogIn` and `ReadLogIn`. By using these two lines, the next form (Selection) will be displayed, and the current form (LogIn) will be disappeared from the screen.

Before we can move to the Faculty form, let's first handle the coding for the Selection form.

5.18.2 Coding for the Selection Form

This 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 `SelectionForm_Load()` event procedure
2. Coding for the OK button's Click event procedure
3. Coding for the Exit button's Click event procedure

Let's start from the first coding. Open the code window of the SelectionForm and the `SelectionForm_Load()` event procedure, and enter the codes, which are shown in Figure 5.78, into this event procedure.

The codes for this event procedure are straightforward with no tricks. We add three pieces of information related to the CSE_DEPT using the `Add()` method to the Selection combo box in the SelectionForm window to allow users to select one of them to perform the related data query.

The coding for the second step is for the OK button's Click event procedure. Open this event procedure by double-clicking the OK button from the SelectionForm window, and enter the codes, which are shown in Figure 5.79, into this event procedure.

When the OK button is clicked by the user, first, we create three instances for three form windows. Then we need to check which piece of information has been selected by the user from the Selection combo box. Based on that selection, we direct the program to the associated form window using the `Show()` method.

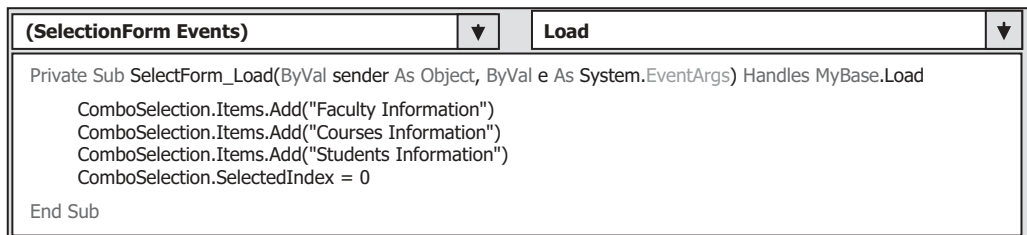


Figure 5.78. The codes for the SelectionForm_Load event procedure.

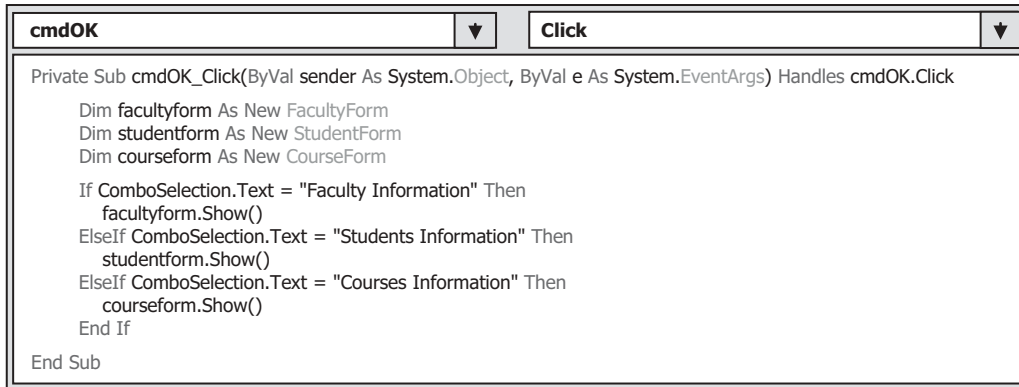


Figure 5.79. The codes for the OK Click button's event procedure.

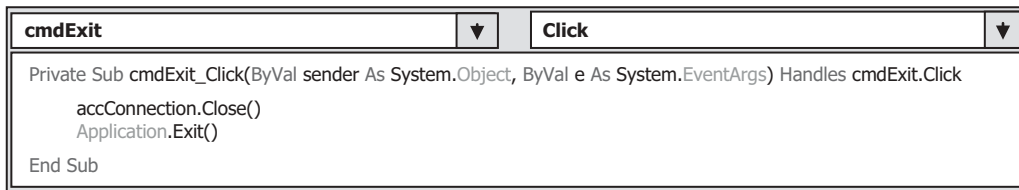


Figure 5.80. The codes for the Exit Click button's event procedure.

Finally, we come to the coding for the third step, coding for the Exit button's Click event procedure. Open the Exit button's Click event procedure and enter the codes shown in Figure 5.80 into this procedure.

Before we can terminate our project, we need first to close our database connection. Then we call a system method `Application.Exit()` to terminate our project.

Now we can move to the next form, Faculty form.

5.18.3 Query Data Using Runtime Objects for the Faculty Form

In this section, we will develop three different query methods to perform data query from the Faculty table in our sample database: `DataAdapter`, `DataReader`, and `LINQ to DataSet` method.

Now open the code window of the Faculty form by clicking on the View Code button from the Solution Explorer window. First, let's create some form-level variables and make coding for the `FacultyForm_Load()` event procedure to put our initialization codes in there. Click on the drop-down arrow from the Class Name combo box and choose the (FacultyForm Events) item, and then click on the drop-down arrow from the Method Name combo box and select the Load item to open this form load event procedure. Enter the codes shown in Figure 5.81 into this code window.

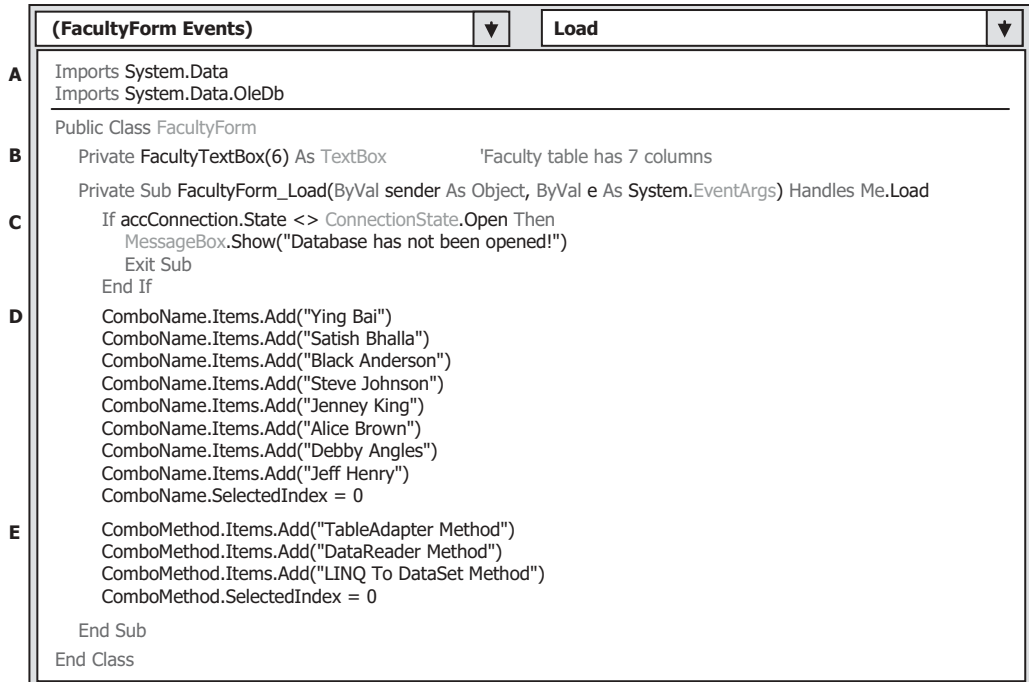


Figure 5.81. The codes for the FacultyForm_Load event procedure.

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

- A.** As we did before, we use the **Imports** keyword to indicate the reference for the namespace that contains the protocols of the **DataTable** class (**System.Data**) and **OleDb** data components (**System.Data.OleDb**).
- B.** In order to hold the retrieved data, an object array is declared as a form-level or module-level textbox array. Since the array index is 0-based, and there are seven columns in our faculty data table, so the array size is selected as six. A little trick for the size of this array is that the first index of this array is 0, not 1. So in total, we have seven textbox objects defined with an index of six.
- C.** Before we can perform any data query, we must make sure that the connection from our project to the database is still active, which means the connection is still open. An error message will be displayed if the current **Connection** instance is not open and the project is exited.
- D.** Eight faculty names are added into the Faculty Name combo box to allow the user to make selection as the project runs. Resetting the **SelectedIndex** property to 0 means that the first faculty name from the combo box has been chosen as the default one.
- E.** Three query methods are added into the ComboMethod combo box to enable the user to choose one method to perform the data query. The **TableAdapter Method** is selected as the default one.

Now let's do the coding for the **Select** button's Click event procedure.

As the project runs, the user can choose one method from the **ComboMethod** combo box to perform the data query job. Then the user needs to click on the **Select** button to begin the related data query.

Open the form window of the **FacultyForm** and double-click on the **Select** button to open the **cmdSelect** button click event procedure. Enter the codes shown in Figure 5.82 into this procedure.

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

- A. The necessary variables and objects to be used in this event procedure are declared first. The objects **accDataTable** and **FacultyTableAdapter** are used for the first method, the **TableAdapter** method, and **accDataReader** is used for the second method, the **DataReader** method, and the **DataSet ds** is used for the third method, the **LINQ to DataSet** method. The object **accCommand** will be used for all three methods in this event procedure.
- B. The **Command** instance is initialized with the connection object, the command type, and the connection string.
- C. The **Add()** method is called to add the content of the **Faculty Name** combo box control, which will be 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 with the column name in the **Faculty** table, also it must be identical with the dynamic parameter we defined in the query string **cmdString2**.
- D. A user-defined subroutine procedure **ShowFaculty()** that will be developed later is executed to display the selected faculty photo in the **PhotoBox** control on the **Faculty** form window. The argument of this subroutine is the faculty name.
- E. Next, we need to check which data query method the user has selected. If the first method, **TableAdapter**, is chosen, we need to assign the completed **Command** object to the **SelectCommand** property of the **TableAdapter**, and then 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. If this table filling is fine, a user-defined subroutine procedure **FillFacultyTable()** is executed with the filled **Faculty** table as the argument to fill the textbox controls on the **Faculty** form window.
- G. Otherwise, an error message is displayed.
- H. A cleaning job is performed to release the **FacultyTableAdapter** and the **DataTable** objects.
- I. If the user selected the second method or the **DataReader** method to perform this data query, the method **ExecuteReader()** 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 subroutine procedure **FillFacultyReader()** is called with the **DataReader** as an argument to fill the textbox 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.
- L. The **DataReader** object is released.

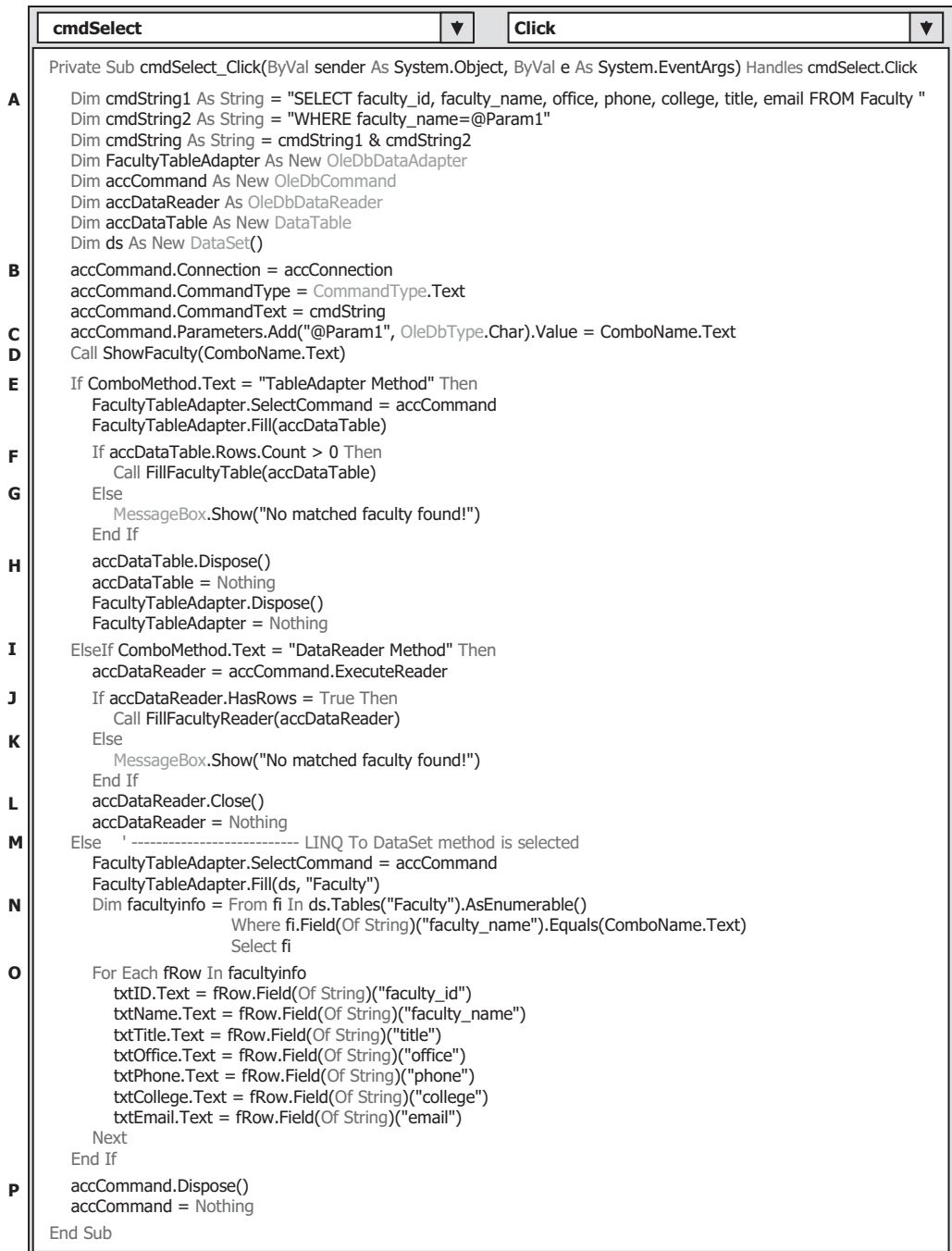


Figure 5.82. The codes for the cmdSelect button Click event procedure.

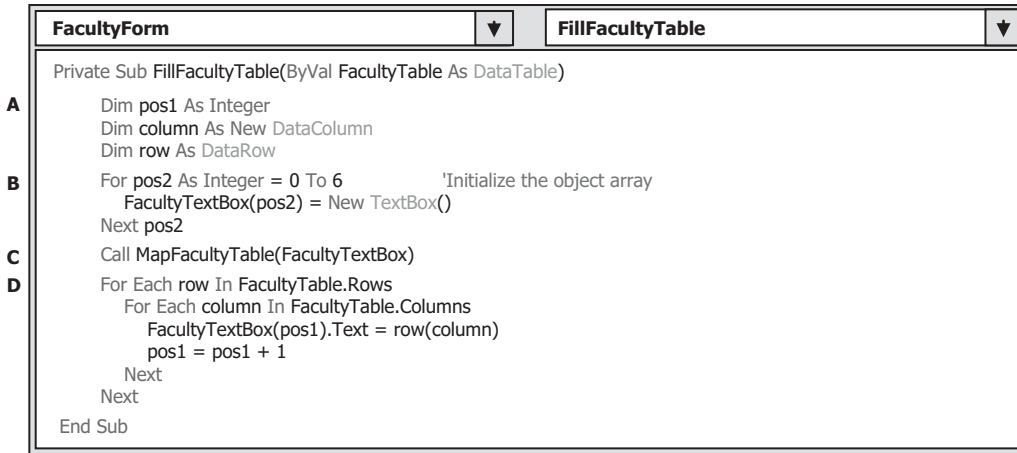


Figure 5.83. The codes for the user-defined subroutine procedure FillFacultyTable.

- M.** If the user selected the third method, **LINQ To Object**, the initialized command object is assigned to the **SelectCommand** property, and the **Fill()** method is executed to fill the **DataSet** object **ds** with the data queried from the **Faculty** table.
- N.** A **LINQ to DataSet** query is created with an implicit variable **facultyinfo**.
- O.** A **For Each** loop is executed to perform this **LINQ To Object** query, and the query results are assigned to seven textbox controls in the **Faculty** form to display the query result.
- P.** Finally, the **Command** object is released.

Now let's take a look at the coding for four user-defined subroutine procedures used in the above **Select** button Click event procedure. The first one is the **FillFacultyTable()**, and its codes are shown in Figure 5.83.

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

- A.** To access the **DataTable** object, one must use the suitable properties of the **DataTable** class. The **DataRow** and the **DataColumn** are two important properties. By using these two properties, we can scan the whole **DataTable** to get each data from each row. 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 textbox control on the **Faculty** form.
- B.** Next, we need to initialize the module-level object array **FacultyTextBox** by creating 7 instances of the **TextBox** control. Recall that we have seven columns in our **Faculty** table, so the size of this textbox array is seven (from 0 to 6).
- C.** Then another user-defined subroutine procedure **MapFacultyTable()**, which will be built later, is called to set a correct mapping relationship between each textbox object in the **TextTable** array and the data retrieved from the **DataTable**.
- D.** Two **For Each . . . Next** loops are utilized to assign each data read out from the **DataTable** to the mapped textbox control on the **Faculty** form window. In this application, we have only one row (one record) selected from the **Faculty** table based on the faculty name, so the outer loop is only executed one time, and the inner loop is executed seven times. Because the distribution order of the textbox controls in the **Faculty** form and the column

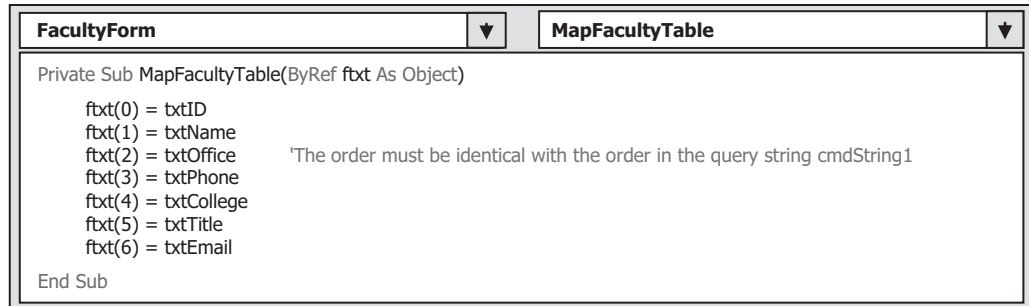


Figure 5.84. The codes for the user-defined subroutine procedure MapFacultyTable.

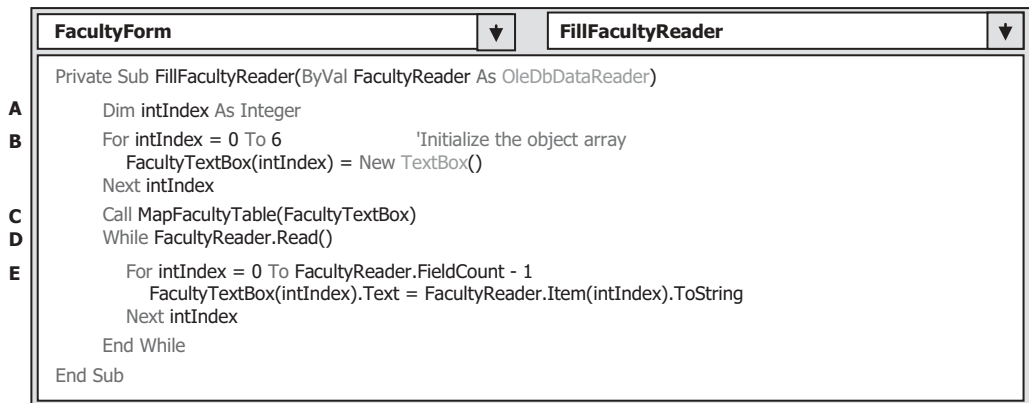


Figure 5.85. The codes for the subroutine procedure FillFacultyReader.

order in the query string (cmdString1) is not identical, we need this MapFacultyTable() subroutine procedure to align them.

A clear picture can be obtained about this MapFacultyTable() subroutine procedure after we have a detailed analysis for this procedure. The detailed codes for this subroutine procedure are shown in Figure 5.84.

The order of textboxes on the right-hand side of the equal symbol is the order of the queried columns in the query string—cmdString1, but the distribution order of seven textbox controls on the Faculty form window is different. By performing this assignment, the seven textbox controls on the Faculty form window have a one-to-one mapping relation with the queried columns in the Faculty table.

Now let's do the coding for another subroutine procedure FillFacultyReader(). This subroutine is used to retrieve the queried data from the DataReader and distribute them to the seven textbox controls on the Faculty form window. The detailed codes for this subroutine procedure are shown in Figure 5.85.

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

- A.** A loop counter intIndex is declared.
- B.** Seven instances of the object array, textbox array, is created and initialized. These seven objects are mapped to seven columns in the Faculty table in the database.

- C. The user-defined subroutine procedure **MapFacultyTable()** is called to set up the correct mapping between the seven textbox controls on the Faculty form window and seven columns in the Faculty table in our sample 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 any 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 . . . Next** 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 form window. The **Item** property with the index is used here to identify each data from the **DataReader**.

The last user-defined subroutine procedure we need to develop the coding of is the **ShowFaculty()**.

This subroutine is used to identify the faculty photo based on the faculty name, and display the selected faculty photo in the **PhotoBox** control on the Faculty form window. The codes for this part are very similar to those we made in Section 5.13.2.

A **Select Case** structure is utilized to select the correct faculty photo, and the system drawing method **System.Drawing.Image.FromFile()** is called to display the faculty photo. 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 for those faculty image, including the drive and path, as well as the faculty image name, to the system drawing method to perform this displaying. A convenient way to do this is to save all faculty image files in a folder in which your Visual Basic.NET executable file is located. In this way, you do not need to provide the full name for those faculty image files, but only the name of each faculty image file.

For example, in this application, our Visual Basic.NET 2010 project executable file, **AccessSelectRTOBJ.exe**, is located at the folder **Chapter 5\AccessSelectRTOBJ Solution\AccessSelectRTOBJ\bin\Debug**. So all faculty image files should be saved to this folder. The detailed codes for this user-defined subroutine procedure is shown in Figure 5.86.

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

- A. A local string variable **FacultyImage** is declared first, and it is used to hold the selected faculty image file.
- B. The **Select Case** structure is used to choose the correct faculty image file, and save it into the local string variable **FacultyImage**. Since these faculty image files are saved in the folder in which our Visual Basic.NET 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 were found, a default faculty image is assigned to the **FacultyImage** variable and a mismatch message is displayed.
- D. The system drawing method is executed to draw the faculty image based on the name of the selected faculty image file.

The last coding job we need to do is to make codes for the **Back** button Click event procedure. This coding process is very easy. The current form window, the Faculty form,

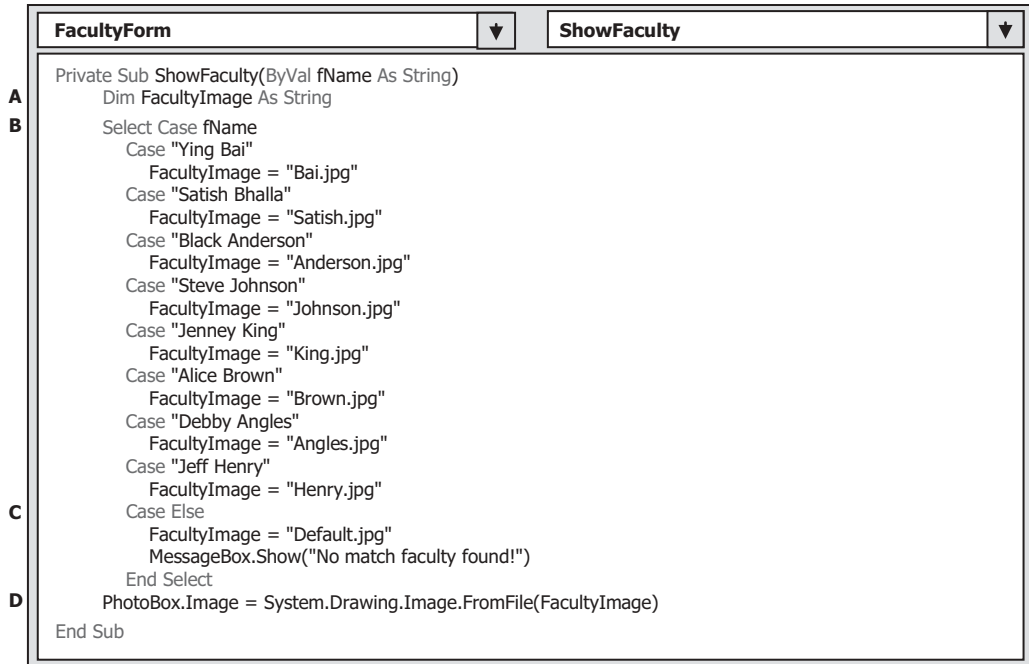


Figure 5.86. The codes for the user-defined subroutine procedure ShowFaculty.

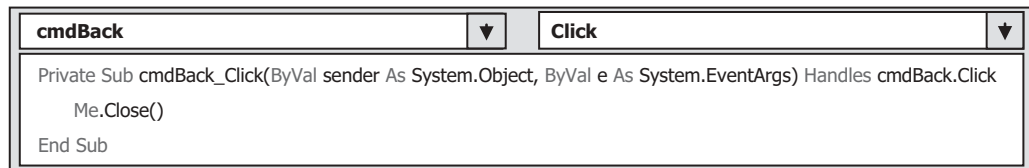


Figure 5.87. The codes for the Back Button Click event procedure.

should disappear from the project if this button is clicked. The method `Close()` is called to close the FacultyForm window, as shown in Figure 5.87.

At this point, we have finished the coding process for our FacultyForm. However, before we can run the project to test the function of this form, we need to confirm that our LogInForm is the startup form in this project. To do that, go to **Project! AccessSelectRTOObject Properties** menu item to open the Project Properties wizard. Keep the **Application** tab selected, and make sure that the **Startup** form combo box contained our LogInForm.

Click on the Start button to run our project. Enter `jhenry` and `test` as the username and password for the LogIn form window, and then click on the **TabLogIn** button to open the Selection form window. Make the default selection **Faculty Information** unchanged and click on the OK button to open the Faculty form window, which is shown in Figure 5.88.



Figure 5.88. The running status of the Faculty form.

Select any method from the Query Method combo box, choose the desired faculty member from the Faculty Name combo box, and click on the **Select** button to make the information query for the selected faculty. Seven textbox controls that are bound to the associated columns in the Faculty table are updated with the queried faculty information, and the selected faculty image is also displayed in the PhotoBox control, which is shown in Figure 5.88. You can try to select the different method and different faculty member to test the function of this Faculty form. Yes, the project works fine without problem at all!

Our next job is to develop the codes for the Course form to access the course information stored in the Course table in our sample database.

5.18.4 Query Data Using Runtime Objects for the Course Form

Similar to query data from the Faculty table, three data query methods can be used for the course data query in this form: `DataAdapter`, `DataReader`, and `LINQ To DataSet` method. However, because of the space limitation, we only discuss the first two methods in this section and leave the third method, `LINQ To DataSet`, as a homework to enable the readers to develop this method.

First, let's develop the codes for the `CourseForm_Load()` event procedure. Open the code window by clicking on the View Code button from the Solution Explorer window. Click on the drop-down arrow from the Class Name combo box and select the (CourseForm Events), and then click on the drop-down arrow from the Method Name combo box and select the Load item to open its `Form_Load` event procedure. Enter the codes shown in Figure 5.89 into this event procedure.

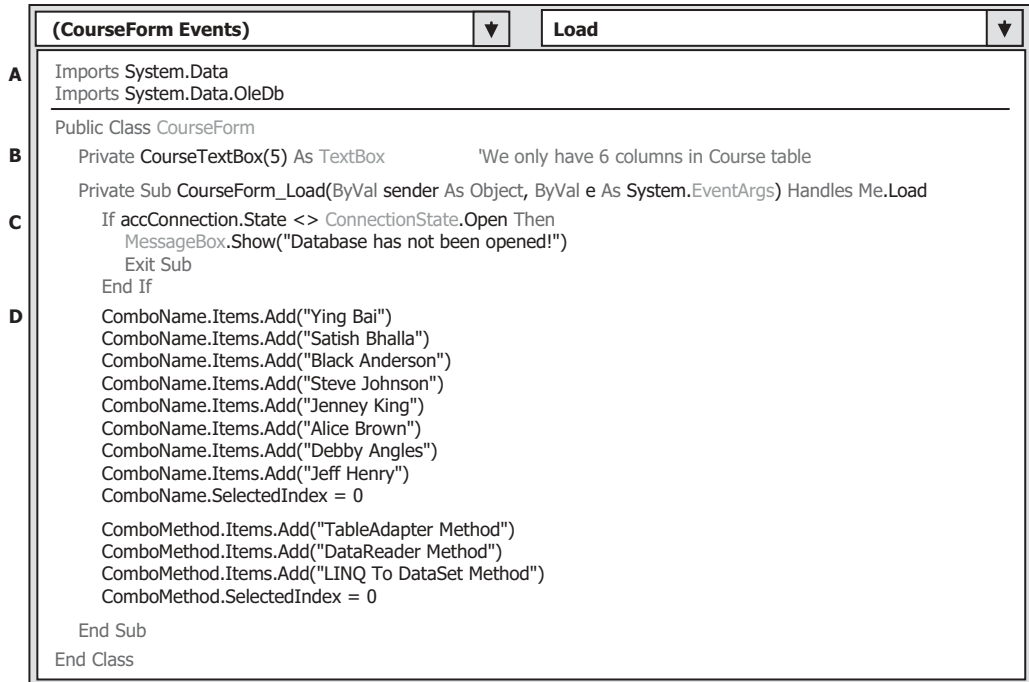


Figure 5.89. The codes for the CourseForm_Load event procedure.

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

- A.** This coding fragment is very similar to the one we did for the Faculty form. First, two namespaces related to the OleDb Data Provider are imported since we need to use some components related to that provider.
- B.** Similar to the FacultyTextBox array, here, a CourseTextBox() array is created. The only difference is that the size of the textbox array has been reduced to 6 (0~5) since we used six textbox controls to display the detailed course information that is related to the selected faculty from the Faculty Name combo box. The Course table has 7 columns, but we only need 6 of them (refer to Fig. 5.21), 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.
- C.** The functionality of this code segment is: before we can perform any data query, we need to check whether a valid connection is still open. An error message would be displayed if no valid database connection exist and the project will be exited.
- D.** The Faculty Name and the Query Method combo boxes are initialized with eight faculty members and three query methods using the Add() method. The first item in both combo boxes have been selected as the default item and will be displayed as the project runs.

The next coding job is for the **Select** button Click event procedure. After the user selected the desired data query method from the Query Method combo box and the faculty member from the Faculty Name combo box, the **Select** button is used to trigger its event procedure to retrieve all courses (course_id) taught by the selected faculty.

One point you need to note is that we need two queries to perform this data action in this event procedure 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 matched to the faculty name selected by the user from the Faculty Name combo box in the Course form, and then we can make the second query to the Course table to pick up all **course_id** based on the **faculty_id** we obtained from the first query. The queried **course_id** are displayed in the CourseList box, and the detailed course information for each course can be displayed in six textboxes when the user clicks on the associated **course_id** from the CourseList box.

Now open the **Select** button Click event procedure and enter the codes shown in Figure 5.90 into this procedure. The codes of this part are very similar to those we did for the **Select** button event procedure in the Faculty form.

Let's have a closer look at this piece of codes to see how it 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. There are six query items related to six columns: **course_id**, **course**, **credit**, **classroom**, **schedule**, and the **enrollment**. The queried **course_id** will be displayed in the CourseList box, and the all queried six items will be displayed in six textboxes as the detailed information for the selected **course_id**. The **faculty_id** is used as the criterion to query the desired course information for the selected faculty. Other necessary instances are also created at this part to aid the data query task. Two TableAdapters, two Command and DataTable objects are declared to facilitate these two queries.
- B. 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 combo box, in which the selected faculty name will be entered by the user as the project runs. The completed Command object **accCmdFaculty** is then assigned to the **SelectCommand** property of the **FacultyTableAdapter**, which is ready to make a query by using the **Fill()** method. The first query is used to find the **faculty_id** that is associated with the selected faculty name.
- C. The **Fill()** method is called to fill the faculty data table named **accFacultyTable**. By checking the **Count** property, we can inspect whether this **Fill** is successful or not. An error message will be displayed if this **Fill** has failed. If the **Fill** is successful, which means that at least one row is returned (exactly only one field is returned since we only query for **faculty_id**) based on the faculty name. The first row, **Rows.Item(0)**, which is the only returned row, is assigned to an object of the **DataRow** class, **rawFaculty**. Since we only need the first column from this queried row, which is **rowFaculty(0)**, and this column is the **faculty_id**, and it is assigned to the local string variable **strFacultyID** that will be used later.
- D. Next, the Course Command object **accCmdCourse** is initialized, and the dynamic parameter **faculty_id** is replaced by the real **faculty_id** obtained above.
- E. As we did for the Faculty form, the user can make a choice between three methods: **TableAdapter**, **DataReader**, and **LINQ to DataSet**. If the user selected the **TableAdapter** method, the built command object will be assigned to the **SelectCommand** property of the **CourseTableAdapter**, and the **Fill()** method of the **TableAdapter** will be executed to fill the Course table.
- F. 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 subroutine

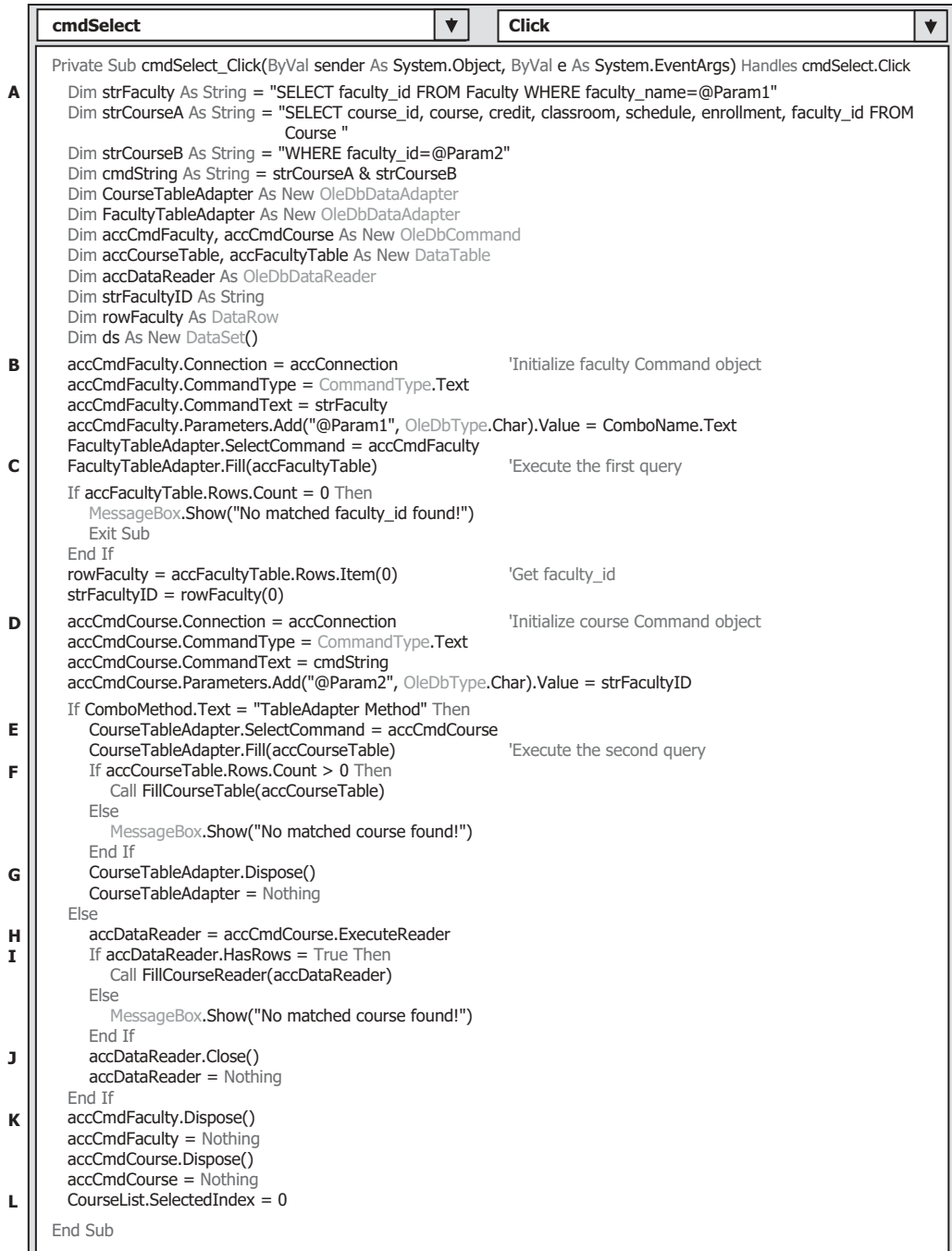


Figure 5.90. The codes for the Select button Click event procedure.

`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. Otherwise, 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.

- G. Some necessary cleaning job is performed to release objects used for this query.
- H. If the user selected the `DataReader` method, the `ExecuteReader()` method is called to perform a reading-only operation to the database.
- I. If the `HasRows` property of the `DataReader` is `True`, which means that the `DataReader` did receive some data, the user-defined subroutine `FillCourseReader()` is executed with the `DataReader` as the argument to fill the `CourseList` box control on the `Course` form window. An error message will be displayed if the `HasRows` property is `False`.
- J. Finally, the `DataReader` and the `Command` objects are released.
- K. Other used components are also cleaned up and released.
- L. This coding line is very important, and it is used to select the first `course_id` as the default one from the `CourseList` box, and this coding line can be used to trigger the `CourseList_SelectedIndex Changed()` event procedure to display detailed information for the selected `course_id` in the six textboxes. Without this default `course_id` selected, no detailed course information can be displayed as the `Course List_SelectedIndex Changed()` event procedure is executed for the first time.

Now let's take a look at the codes for two user-defined subroutine procedures used in this part, `FillCourseTable()` and `FillCourseReader()`. These two subroutines are used to fill the `CourseList` box control on the `Course` form window by using the queried data. Figure 5.91 shows the codes for these two user-defined subroutine procedures.

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

- A. Before we can fill the `CourseList` box, a cleaning job is needed. This cleaning is very important and necessary; otherwise multiple repeat courses (`course_id`) would be displayed in this listbox if you forget to do this cleaning job.

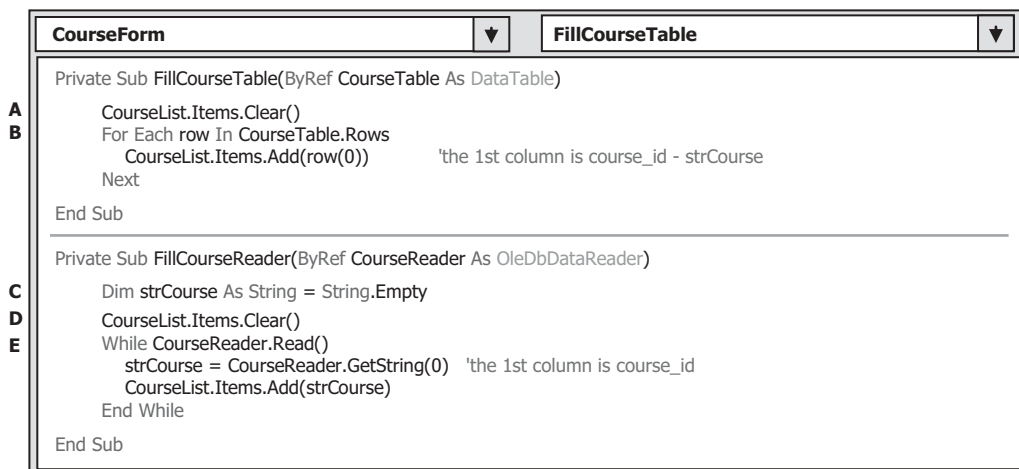


Figure 5.91. The codes for two user-defined subroutine procedures.

- B.** A For Each loop is used to scan all rows of the filled Course table. Recall that we filled seven columns from the Course table in the database to this Course table in the DataTable object starting with the first column `course_id` (refer to query string `strCourseA` defined in Fig. 5.90). Now we need to pick up the first column, `course_id` (column index = 0) for each returned row of the Course table. Then the `Add()` method is used to add each retrieved `row(0)`, which equals to `course_id`, into the CourseList Box control.
- C.** For the `FillCourseReader()` subroutine, 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 data from the Course table.
- D.** Similarly, we 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 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()` event procedure. The functionality of this procedure is to display the detailed information related to the selected `course_id` from the CourseList box, which includes the course ID, classroom, schedule, credit, and enrollment, and these pieces of information will be displayed in six textbox controls on the Course form window. This event procedure 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()` event procedure. Enter the codes that are shown in Figure 5.92 into this event procedure. The code segment in this part is very similar to the one we did for the `cmdSelect` button event procedure.

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

- A.** The query string is defined with six data columns. Note that the first column is the `course_id` with a column index of 0, and the criterion for the WHERE clause is also `course_id`. This is because we want to retrieve all information related to the selected `course_id` and display those information in 6 textbox controls, and one of them is `course_id`. Also, some necessary objects are created here, and the command object is initialized here. The dynamic parameter is the `course_id`.
- B.** The dynamic parameter `course_id` now is replaced by the real `course_id` parameter located in the CourseList by using the `SelectedItem` property.
- C.** If the user selected the `TableAdapter` method, the built command object is assigned to the `SelectCommand` property of the `CourseTableAdapter`, and the `Fill()` method is called with the Course table as the argument to fill the Course table.
- D.** If this fill is successful, which can be detected by checking the `Count` property of the `DataTable`, the queried data should have been stored in the Course table. Next, a user-defined subroutine procedure `FillCourseTextBox()` that will be built later is executed with the `DataTable` as the argument to fill six textbox controls in the Course form window. An error message will be displayed if this fill has failed.
- E.** A cleaning job is performed to release used objects, which include the `DataTable` and the `TableAdapter`.

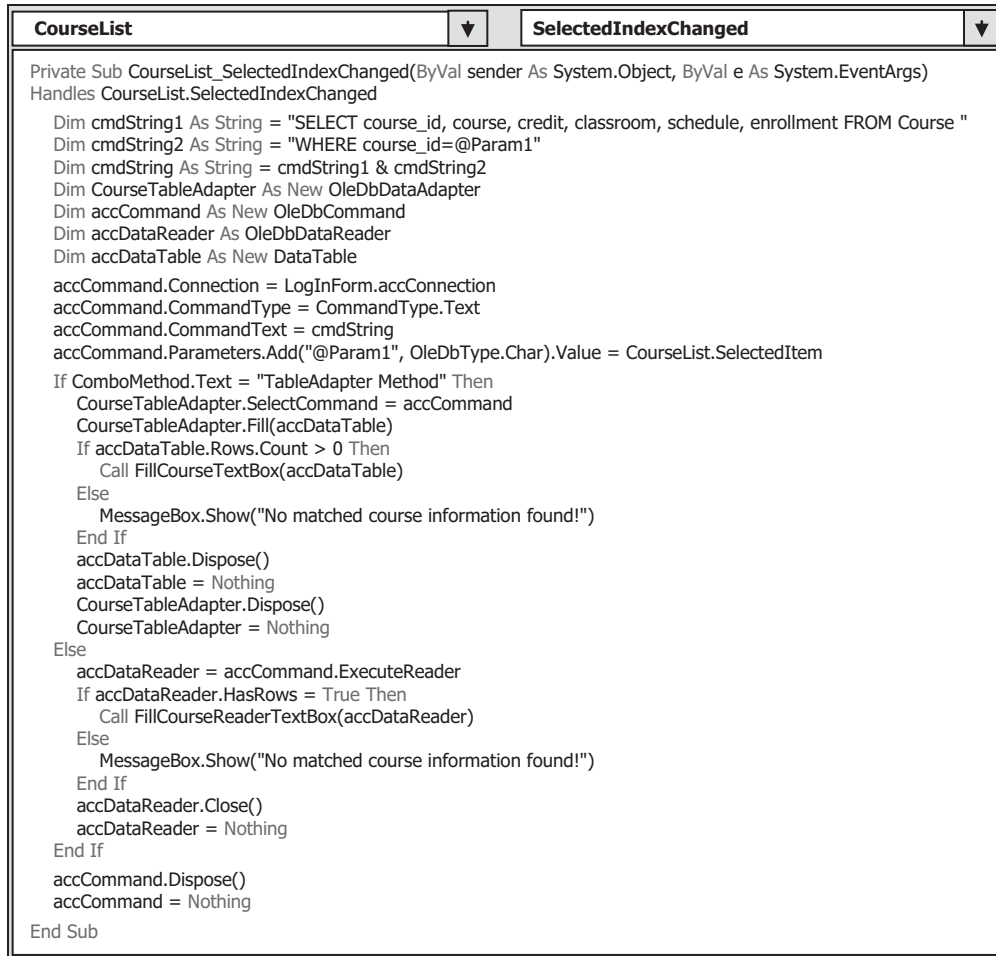


Figure 5.92. The codes for the CourseList SelectedIndexChanged event procedure.

- F.** If the user selected the DataReader method, the ExecuteReader() method is executed to perform a read-only operation to retrieve the detailed information related to the selected course_id from the CourseList box.
- G.** If this read-only operation is successful, the HasRows property of the DataReader will be True, another user-defined subroutine procedure FillCourseReaderTextBox() is called to fill six textbox controls on the Course form window. An error message will be displayed if this operation has failed.
- H.** A cleaning job is performed to release used objects for this data query.

The codes for two user-defined subroutine procedures, FillCourseTextBox() and FillCourseReaderTextBox(), are shown in Figure 5.93.

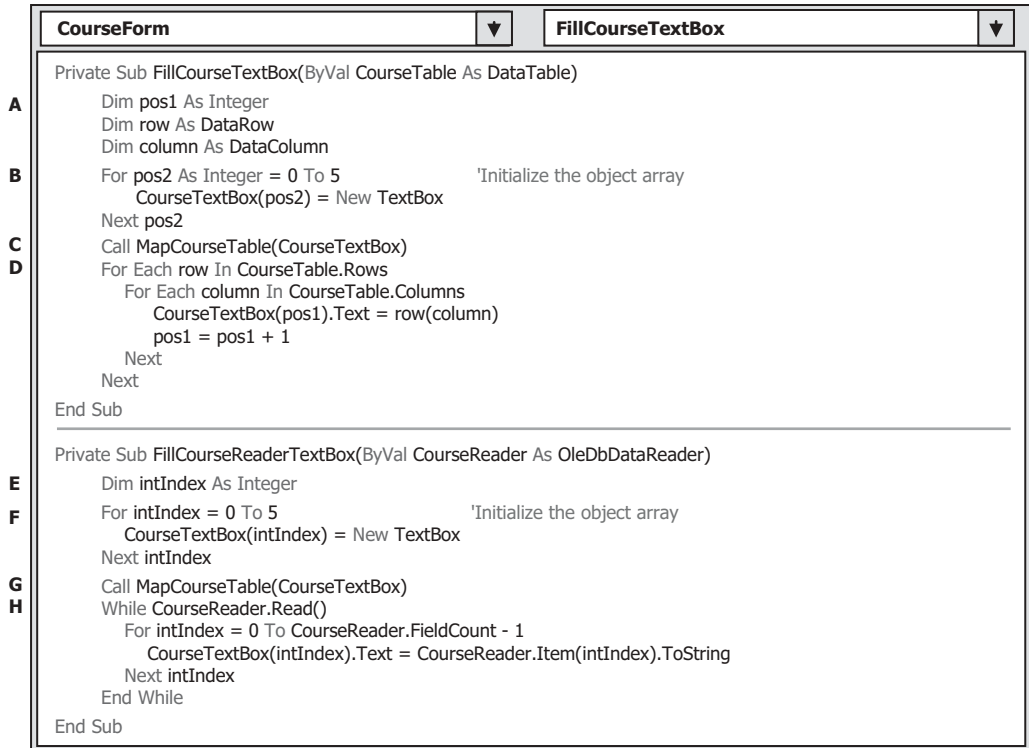


Figure 5.93. The codes for two user-defined subroutine procedures.

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

- A.** As we mentioned in the coding process for the Faculty form window, the **DataTable** can be scanned by using two important objects: **DataRow** and **DataColumn**. You must use these two objects to access the **DataTable** to retrieve data stored in that **DataTable**.
- B.** The module-level object array, **CourseTextBox()**, are created and initialized here. For any object or object array, it should be created by using the **New** operator. Six textbox objects are created, and they can be mapped to six textbox controls in the **Course** form window. We use these six textbox objects to display the detailed course information for the selected **course_id** from the **CourseList** box later.
- C.** Another user-defined subroutine procedure **MapCourseTable()** is executed to set up a one-to-one mapping relation between each textbox control on the **Course** form window and each queried column in the queried row. This step is necessary since the distribution order of six textbox controls on the **Course** form is different with the column order in the query.
- D.** A double **For Each** loop is utilized to retrieve all columns and all rows from the **DataTable**. The outer loop is only executed by once since we only query one record (one row) course's information based on the selected **course_id** from the **Course** data table. The inner loop is exactly executed by six times to pick up six pieces of course-related information that contains the course title, classroom, credit, schedule, and the enrollment. Then, the retrieved

information is assigned to each textbox control in the textbox array, which will be displayed in that textbox control.

- E. For the subroutine `FillCourseReaderTextBox()`, a loop counter `intIndex` is first created, and it is used to create six textbox objects array and retrieve data from the `DataReader` later.
- F. This loop is used to create the textbox objects array and perform the initialization for those objects.
- G. Same functionality as described in step C.
- H. A `While` and a `For . . . Next` loop are used to pick up all six pieces of course-related information from the `DataReader` one by one. The `Read()` method is used as the `While` loop's condition. A 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 . . . Next` 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 objects array.

The detailed codes for the subroutine `MapCourseTable()` is shown in Figure 5.94.

This piece of codes is straightforward, with no tricks. The order of the textboxes on the right-hand side of the equal operator is the column order of the query string, `cmdString1`. By assigning each textbox control on the Course form window to each of its partner, the textbox in the textbox objects array in this order, a one-to-one mapping relationship is built, and the data retrieved from the `DataReader` can be exactly mapped to and displayed in the associated textbox control.

The last coding process is for the Back button Click event procedure. This coding is very simple, and the codes are shown in Figure 5.95.

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 Query Method combo box, and then select your desired faculty from the Faculty Name combo box. Click on the **Select** button, and all courses, that is, all `course_id`, taught by the selected faculty will be displayed in the CourseList box, which is shown in Figure 5.96. Then select any `course_id` by clicking on it from the CourseList box, and the detailed course information related to that selected course will be displayed in six textbox controls, as shown in Figure 5.96.

It is so funny!

5.18.5 Query Data Using Runtime Objects for the 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 six textboxes as the **Select** button is clicked.

CourseForm

MapCourseTable

```
Private Sub MapCourseTable(ByRef fCourse As Object)
    fCourse(0) = txtID           'The order must be identical with the column order in the query
    fCourse(1) = txtCourse       'string – cmdString1 in CourseList_SelectedIndexChanged procedure
    fCourse(2) = txtCredits
    fCourse(3) = txtClassRoom
    fCourse(4) = txtSchedule
    fCourse(5) = txtEnroll
End Sub
```

Figure 5.94. The codes for the subroutine MapCourseTable.

cmdBack

Click

```
Private Sub cmdBack_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdBack.Click
    Me.Close()
End Sub
```

Figure 5.95. The codes for the Back button Click event procedure.

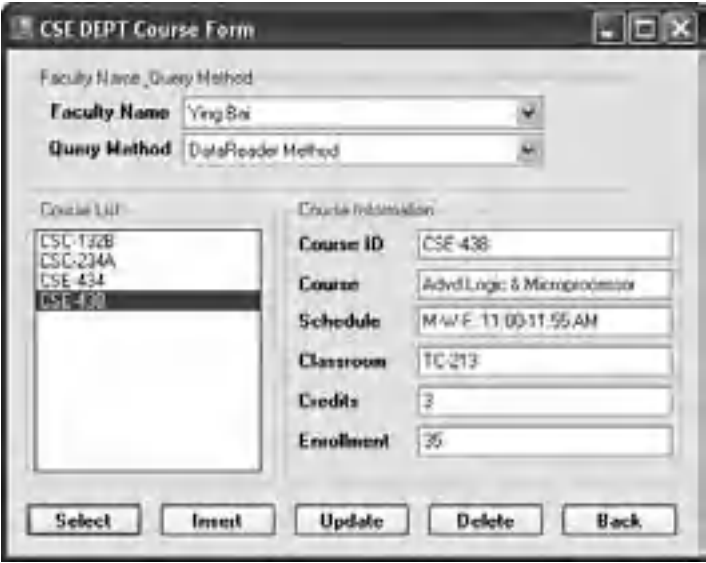


Figure 5.96. The running status of the Course form window.

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 general information from the **Student** table, and the second is to pick up all courses (**course_id**) 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 **Form_Load()** event procedure and the **Select** button Click event procedure. The first one is used to initialize the Student form and display all students' names on the combo box 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 the Student Form_Load Event Procedure

The codes for this event procedure are shown in Figure 5.97.

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

- A. As we did before for the **LogIn**, **Faculty**, and **Course** forms, import the system related **Data** and the **OleDb** related namespace, **System.Data** and **System.Data.OleDb**, into this code window, since we need to use some data components involved in those namespaces.
- B. A **TextBox** array **StudentTextBox()** is created in here, and this object array is used to set up a bridge between the seven textboxes in the Student form and seven query columns in the query string **strStudent**, which includes **student_id**, **student_name**, **gpa**, **credits**, **major**, **schoolYear**, and **email**.

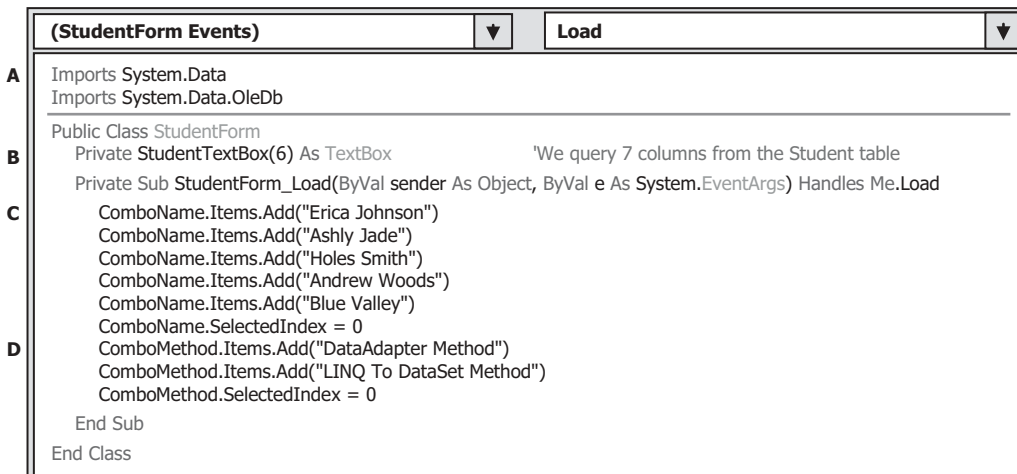


Figure 5.97. The Student Form_Load event procedure.

- C. Five default students' names are added into the Student combo box. As the project runs, the user can select any student by clicking on the associated name to review the detailed information for the selected student in six textboxes and all courses taken by that student in the CourseList box. The first student's name is selected as the default name by setting the SelectedIndex property value as 0.
- D. Two query methods, `DataAdapter` and `LINQ to DataSet`, are added into the Query Method combo box to enable users to select one of them to perform the related data query from the Student and StudentCourse tables.

Next, let's handle the coding for the **Select** button Click event procedure.

5.18.5.2 Coding for the Select Button Click Event Procedure

As the project runs, the user can select a student's name from the Student Name combo box and click on the **Select** button. The detailed information for the selected student is queried from the Student table in our sample database CSE_DEPT and displayed in six textboxes. Also, all courses that are represented by all `course_id` and taken by the selected student are retrieved from the StudentCourse table and displayed in the CourseList listbox. So this event procedure needs to perform two queries from two different tables.

The coding for this event procedure is shown in Figure 5.98. Let's have a closer look at this piece of codes to see how it works.

- A. The query string for the Student table is declared, and the string contains seven columns in the Student data table, which are: `student_id`, `student_name`, `gpa`, `credits`, `major`, `schoolYear`, and `email`. The criterion for this query is the student name stored in the student combo box. Since the string is relatively long, two substrings are used, and an ampersand operator "&" is used to concatenate them together to form a complete query string.
- B. The second string, or the query string for the StudentCourse table, is created, and two columns are queried, which are `course_id` and `student_id`, and the query criterion is the `student_id`. The reason we query the `student_id` is that the `LINQ to DataSet` method needs to fill a `DataSet` with both columns later.
- C. All data operation objects are created here, such as the Student and the StudentCourse TableAdapters, Commands, and DataTables. The variable `strName` is used to hold the returned student's photo from calling the function `FindName()`.
- D. The `FindName()` function, which will be built later, is called to get the appropriate student's photo based on the student name. If no matched photo is found, an error message is displayed, and the procedure is exited.
- E. The picture box is initialized and executed to display the selected student's photo.
- F. The user-defined subroutine `BuildCommand()` is called to build the Student Command object with the Student Command object and the student query string as the arguments. You will find that the data type of the first argument, `accCmdStudent`, is a reference (`ByRef`), which is equivalent to a memory address or a pointer variable in C++, from the subroutine `BuildCommand()` protocol later. When the subroutine is done, the built command object is still stored in that reference, and we can use it without problem. The dynamic parameter name is replaced by a real student name obtained from the student name combo box, and the completed Command object is assigned to the `SelectCommand` property of the TableAdapter.

```

cmdSelect Click
Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
A   Dim strStudent1 As String = "SELECT student_id, student_name, gpa, credits, major, schoolYear, email FROM
    Student "
    Dim strStudent2 As String = "WHERE student_name=@Param1"
    Dim strStudent As String = strStudent1 & strStudent2
B   Dim strStudentCourse As String = "SELECT course_id, student_id FROM StudentCourse WHERE
    student_id=@Param2"
C   Dim StudentTableAdapter As New OleDbDataAdapter
    Dim StudentCourseTableAdapter As New OleDbDataAdapter
    Dim accCmdStudent, accCmdStudentCourse As New OleDbCommand
    Dim accStudentTable, accStudentCourseTable As New DataTable
    Dim ds As New DataSet()
    Dim strName As String
D   strName = FindName(ComboName.Text)
    If strName = "No Match" Then
        MessageBox.Show("No Matched Student Found!")
        Exit Sub
    End If
E   PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage
    PhotoBox.Image = System.Drawing.Image.FromFile(strName)
F   Call BuildCommand(accCmdStudent, strStudent) 'Initialize the Student Command object
    accCmdStudent.Parameters.Add("@Param1", OleDbType.Char).Value = ComboName.Text
    StudentTableAdapter.SelectCommand = accCmdStudent
G   If ComboMethod.Text = "DataAdapter Method" Then
        StudentTableAdapter.Fill(accStudentTable) 'Execute the first query
        If accStudentTable.Rows.Count > 0 Then
            Call FillStudentTextBox(accStudentTable)
        Else
            MessageBox.Show("No matched student found!")
        End If
H   Call BuildCommand(accCmdStudentCourse, strStudentCourse) 'Initialize the StudentCourse Command object
    accCmdStudentCourse.Parameters.Add("@Param2", OleDbType.Char).Value = txtID.Text
    StudentCourseTableAdapter.SelectCommand = accCmdStudentCourse
I   StudentCourseTableAdapter.Fill(accStudentCourseTable) 'Execute the second query
    If accStudentCourseTable.Rows.Count > 0 Then
        Call FillCourseList(accStudentCourseTable)
    Else
        MessageBox.Show("No matched course_id found!")
    End If
    Else '-----LINQ to DataSet Method Selected
J   StudentTableAdapter.Fill(ds, "Student")
K   LINQStudent(ds)
L   BuildCommand(accCmdStudentCourse, strStudentCourse)
M   accCmdStudentCourse.Parameters.Add("@Param2", OleDbType.Char).Value = txtID.Text
N   StudentCourseTableAdapter.SelectCommand = accCmdStudentCourse
    StudentCourseTableAdapter.Fill(ds, "StudentCourse")
O   LINQStudentCourse(ds)
P   ds.Clear()
    End If
Q   StudentTableAdapter.Dispose()
    StudentTableAdapter = Nothing
    StudentCourseTableAdapter.Dispose()
    StudentCourseTableAdapter = Nothing
    accCmdStudent.Dispose()
    accCmdStudent = Nothing
    accCmdStudentCourse.Dispose()
    accCmdStudentCourse = Nothing
End Sub

```

Figure 5.98. The codes for the Student Select button Click event procedure.

- G.** If the user selected the **DataAdapter Method**, the **Fill()** method is called to fill the Student table. 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 is filled into the Student data table and the fill is successful, the subroutine **FillStudentTextBox()** is called with the filled Student table as the argument to fill seven textboxes in the Student form with the detailed student's information, such as **student_id**, **student_name**, **gpa**, **credits**, **major**, **schoolYear**, and **email**, which are stored in the filled Student table. Otherwise, an error message is displayed.
- H.** To enable the second query to the StudentCourse table to find all courses taken by the selected student, the subroutine **BuildCommand()** is called again to initialize the StudentCourse Command object. The dynamic parameter **student_id** is replaced by the real **student_id** that was obtained from the last query and stored in the textbox **txtID**. The completed StudentCourse Command object is assigned to the **SelectCommand** property of the **StudentCourseTableAdapter**.
- I.** The **Fill()** method is called to fill the StudentCourse data table. If the **Count** property is greater than 0, which means that the fill is successful, the subroutine **FillCourseList()** is executed to fill all courses (exactly **course_id**) stored in the filled StudentCourse table into the CourseList box in the Student form. If the **Count** is equal to 0, which means that this fill has failed, an error message is displayed.
- J.** If the user selected the **LINQ to DataSet** method, the **Fill()** method is executed to fill the Student table in the DataSet **ds**.
- K.** A user-defined subroutine **LINQStudent()** is called to perform this LINQ to DataSet method to query data from the Student table.
- L.** The **BuildCommand()** subroutine is executed to initialize the StudentCourse command object **accCmdStudentCourse**.
- M.** The dynamic parameter **student_id** is replaced by the real **student_id** value stored in the Student ID textbox.
- N.** The initialized StudentCourse command object is assigned to the **SelectCommand** object, and the **Fill()** method is executed to run this command to fill the StudentCourse table in our DataSet.
- O.** A user-defined subroutine procedure **LINQStudentCourse()** is called to perform a data query from the StudentCourse table in our sample database.
- P.** The filled DataSet **ds** is cleaned up and released after this data query.
- Q.** A cleaning job is performed to release all used objects by this event procedure.

Now let's continue to finish the coding for all user-defined subroutine procedures used in this event procedure, and these procedures are:

- **FindName()**
- **BuildCommand()**
- **FillStudentTextBox()**
- **MapStudentTextBox()**
- **FillCourseList()**
- **LINQStudent()**
- **LINQStudentCourse()**

StudentForm	FindName
<pre> Private Function FindName(ByVal sName As String) As String Dim strName As String Select Case sName Case "Erica Johnson" strName = "Erica.jpg" Case "Ashly Jade" strName = "Ashly.jpg" Case "Holes Smith" strName = "Holes.jpg" Case Is = "Andrew Woods" strName = "Andrew.jpg" Case Is = "Blue Valley" strName = "Blue.jpg" Case Else strName = "No Match" End Select Return strName End Function </pre>	

Figure 5.99. The codes for the subroutine FindName.

StudentForm	BuildCommand
<pre> Private Sub BuildCommand(ByRef cmdObj As OleDbCommand, ByVal cmdString As String) cmdObj.Connection = accConnection cmdObj.CommandType = CommandType.Text cmdObj.CommandText = cmdString End Sub </pre>	

Figure 5.100. The codes for the subroutine BuildCommand.

First, let's handle the coding for the subroutine FindName(). This method is similar to one that we developed in the Faculty form, and the codes for this method are shown in Figure 5.99.

A Select Case structure is used to select the desired student's photo based on the input student's name. One point you need to note is the location in which the student photo files are located. You can save those photo files in any folder in your computer or a server, but you must provide the full name for these photos 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 photo name. An easy way to save these photos is to save them in the folder in which your Visual Basic.NET executable file is located. For instance, in this application, our VB executable file `AccessSelectRTOObject.exe` is located at the folder `C:\Chapter 5\AccessSelectRTOObject\bin\Debug`. When you save all the students' photos in this folder, you don't need to provide the so-called full name for those photos, and you only need to provide the photo name and assign it to the variable `strName`, as we did in this piece of codes. That is much simpler and easier!

The codes for the subroutine `BuildCommand()` are shown in Figure 5.100.

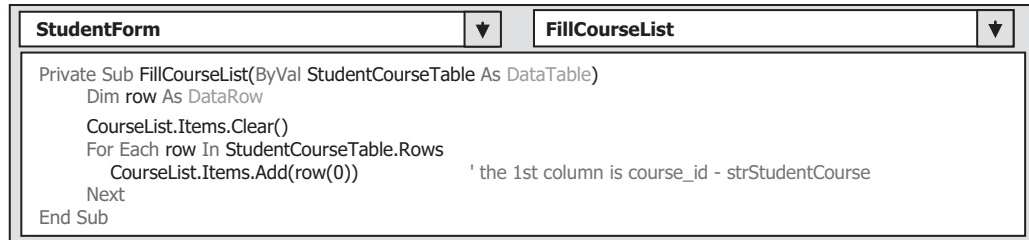


Figure 5.101. The codes for the subroutine FillCourseList.

This 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. The only point one needs to note is the data type of the first argument, `cmdObj`, which is a reference (ByRef) as we mentioned in illustration **F** in Figure 5.98. A reference in Visual Basic.NET is equivalent to a memory address or a pointer in C++, and the argument `cmdObj` is called a passing by reference. When the 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 subroutine is completed. We can use this built `cmdObj` as a returned object even it is an argument without needing to return this `cmdObj` object from this subroutine.

For some other user-defined subroutines used in this form, such as `FillCourseList()`, `FillStudentTextBox()`, and `MapStudentTextBox()`, the coding process for them are similar to those we developed in the Course form. For your convenience, we list them here again with some simple explanations.

The codes for the subroutine `FillCourseList()` are shown in Figure 5.101.

The function of this subroutine is to fill the CourseList box with all courses (`course_id`) 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 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 `For Each` loop is executed to pick up each row from the StudentCourse table. The first column `row(0)`, which is the `course_id`, is added into the CourseList box by executing the `Add()` method.

The next one is the subroutine `FillStudentTextBox()`, and the codes for this subroutine are shown in Figure 5.102.

The function of this piece of codes is to fill seven textboxes in the Student form with seven columns of data obtained from the Student table, such as `student_id`, `student_name`, `gpa`, `credits`, `major`, `schoolYear`, and `email`, which is the first query we discussed above. The `StudentTextBox` array is initialized first, and then the subroutine `MapStudentTextBox()` is called to set up a one-to-one mapping relationship between the `StudentTextBox` array and seven textboxes in the Student form. A nested `For Each` loop is executed to pick up each column's data from the queried row. Only one row data that matches to the selected student name is obtained from the Student table; therefore, the outer loop is only executed one time. The reason of using a double loop is that both the `DataRow` and the `DataColumn` are classes, and in order to pick up data from any

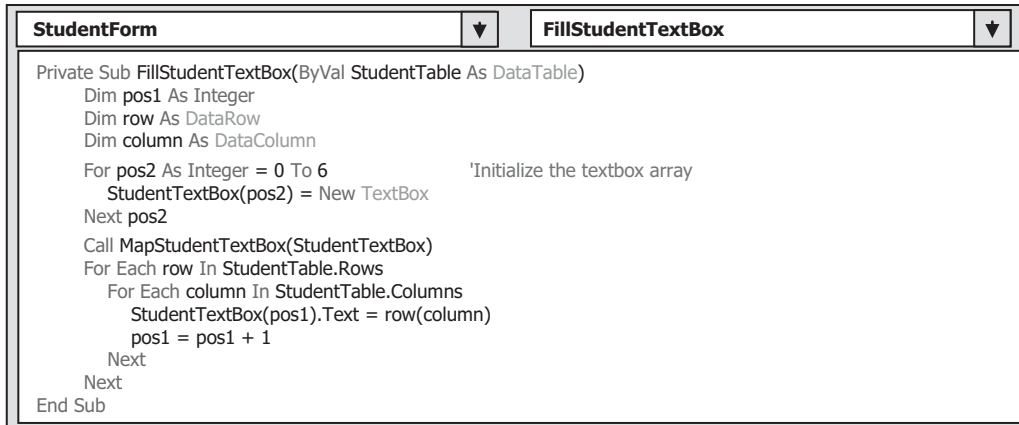


Figure 5.102. The codes for the subroutine FillStudentTextBox.

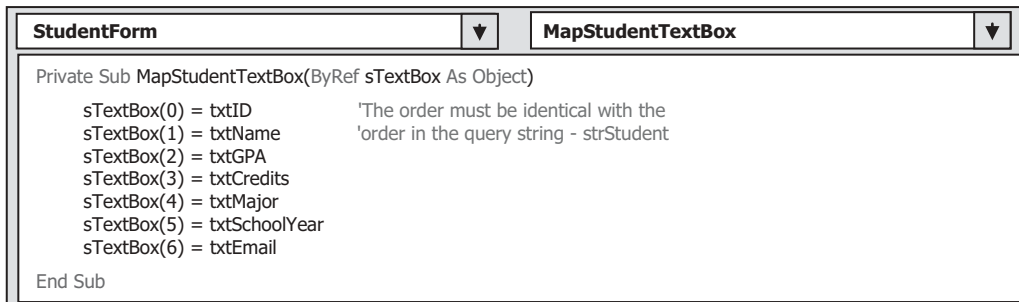


Figure 5.103. The codes for the subroutine MapStudentTextBox.

DataTable, one must use the objects `row` and `column`, which are instances of the `DataRow` and `DataColumn`, as the index to access each row or column of the `DataTable` instead of using an integer. The local integer variable `pos1` works as an index for the `StudentTextBox` array.

The codes for the subroutine `MapStudentTextBox()` are shown in Figure 5.103.

The purpose of this piece of codes is to set up a one-to-one mapping 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`, `student_name`, `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 need to be set up after calling this subroutine.

Another important point one needs to note is the data type of the argument `sTextBox`, which is a nominal reference variable of the `StudentTextBox` array. A reference data type (`ByRef`) must be used for this argument in order for us to use the modified textbox controls in the `StudentTextBox` array when this subroutine returns to our main procedure.

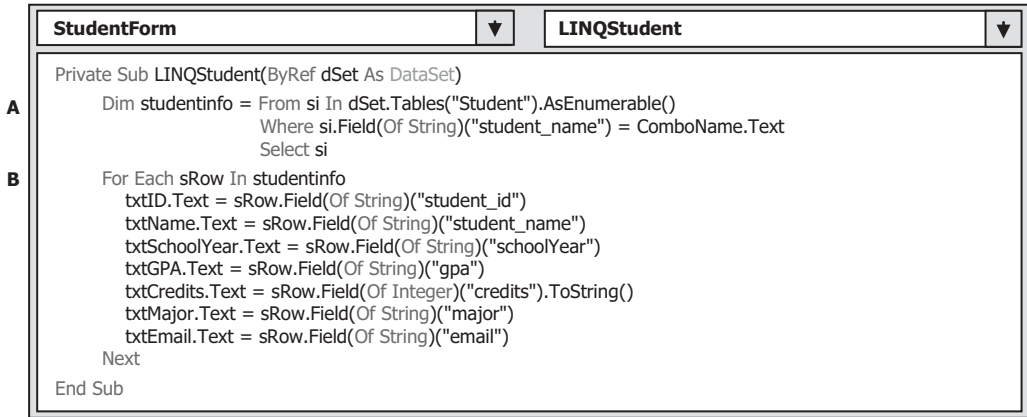


Figure 5.104. The codes for the subroutine LINQStudent.

Now, let's handle the coding for two user-defined subroutine procedures related to the LINQ to DataSet method. First, let's take care of the subroutine LINQStudent(). The codes for this subroutine are shown in Figure 5.104.

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

- A.** A typical LINQ query structure is created and executed to retrieve the detailed student information related to the `student_name`. The `studentinfo` is an implicitly typed local variable. The Visual Basic.NET 2010 can automatically convert this variable 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, therefore, those collections must be converted to the type of `IEnumerable(Of T)` in order to use the LINQ technique, because all data operations in LINQ use a Standard Query Operator methods that can perform complex data queries on an `IEnumerable(Of T)` sequence. A compiling error would be encountered without this operator. The second key point is that you have to use the explicit cast (`Of String`) to convert the data type for each field of queried collection.
- B.** The For Each loop is utilized to pick up each column from the selected data row `sRow`, which is obtained from the `studentinfo` collection 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`, therefore, we must indicate each column clearly with the `field(Of string)` and the column's name as the position for each of them.

The codes for the subroutine LINQStudentCourse() are shown in Figure 5.105. Let's see how this piece of codes works.

- 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 duplicated `course_id` will be added and displayed in this `CourseList` listbox control as the users click on the `Select` button and run this subroutine to perform the student's information query.
- B.** A typical LINQ query structure is created and executed to retrieve the course information related to the `student_id`. The `scinfo` is an implicitly typed local variable, and the Visual

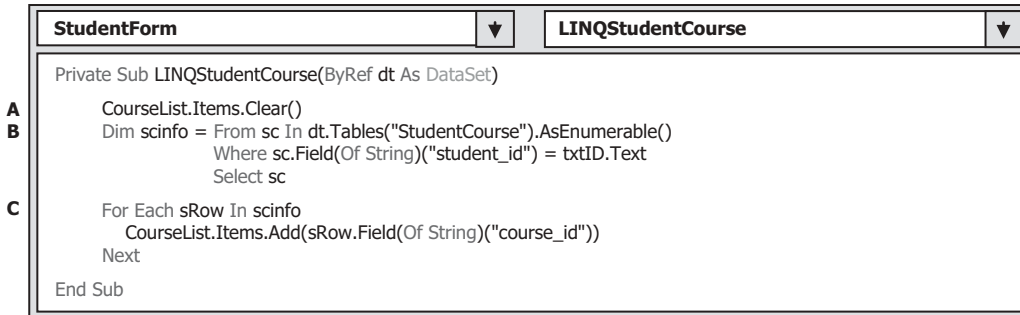


Figure 5.105. The codes for the subroutine LINQStudentCourse.

Basic.NET can automatically convert it 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 `For Each` 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`, therefore, we must indicate each column clearly with the `field(Of String)` and the column's name as the position for each of them.

The last coding job is for the `Back` button. Open the `cmdBack_Click` event procedure and enter the code: `Me.Close()` into this procedure.

Now it is the time for us to run and test our project for this `Student` form. One thing you need to confirm before you run this project is to make sure that all students' photo files have been stored in the same folder as your Visual Basic.NET executable file is located. Click on the `Start Debugging` button to run our project. Enter a 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` combo box, and then click on the `Select` button. All courses taken by this student is shown in the `CourseList` box, and the detailed information about this student is displayed in seven textboxes.

A completed project, `AccessSelectRTOObject`, which includes all GUIs, five form windows, and related codes, can be found in the folder `DBProjects\Chapter 5` that is located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).

Next, we want to discuss how to develop a professional data driven project using the runtime object for SQL Server database.

5.19 QUERY DATA FROM SQL SERVER DATABASE USING RUNTIME OBJECT

In the previous section, you learned how to build a data-driven application using the runtime objects for the Microsoft Access database. Microsoft Access is a very good can-



Figure 5.106. The running status of the Student form.

didate when a small group of users with small amounts of data are dealt with. 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 provider to access the different database, 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.

5.19.1 Migrating from Access to SQL Server and Oracle Databases

Basically, similar runtime objects and structures can be 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, TableAdapter, 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 a 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`, `SqlTableAdapter`, and `SqlDataReader`. Same thing will be worked to the Oracle data provider.

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 provider needs to use the different connection string to make the connection to the associated database.

Regularly, a Connection String is composed of five parts:

- Provider
- Data Source

where the value for the **Data Source** parameter is: *Computer Name\SQL Server 2008 Express name*, since we installed the Express version of the SQL 2008 Server in our local computer. Also, we installed the SQL 2008 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 install and configure 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:

```
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** in your application.

To build a connection string to be used by the Oracle database using the **OracleClient**, you should know that most of 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 is:

```
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 discuss 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 2008 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, and it is located at the folder `Database\SQLServer` that can be found in the Wiley ftp site (refer to Fig. 1.2 in Chapter 1). The advantages of using the Express version of SQL Server 2008 include, but are not limited to:

- The SQL Server 2008 Express is fully compatible with SQL Server 2008 database and has full functionalities of the latter
- The SQL Server 2008 Express can be easily downloaded from the Microsoft site, free of charge
- The SQL Server Management Studio 2008 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

Now we need to create a Visual Basic.NET 2010 project named `SQLSelectRTOject` with five form windows: `LogIn`, `Selection`, `Faculty`, `Course`, and `Student`. Because of the similarity between this project and the project `AccessSelectRTOject` we developed in the last section, you do not need to redevelop all codes for this one. What you need to do is to create a new project, named `SQLSelectRTOject`, and add all five forms from the last project to this one by using `Project\Add Existing Item` menu. The only differences between these two projects are Data Provider-dependent objects, and the most important part is the connection string. To save time, in this section, we only emphasize the different codes that exist between this project and those that exist in the last one.

Now, create a new Window-based project and name it `SQLSelectRTOject`.

5.19.2 Query Data Using Runtime Objects for the LogIn Form

Open the default `Form1` window by double-clicking on the `Form1.vb` from the Solution Explorer window, and then right-click on this form and click on the `Delete` item, and OK on the message box to remove this form from our project. This deletion will cause a compiling error, and we will fix this later.

Now we need to add all five form windows and the Module `ConnModule` from the `AccessSelectRTOject` project to this project. Perform the following operations to complete this adding action:

1. Click on the `Project\Add Existing Item`, and browse to the `AccessSelectRTOject` project folder.
2. Press and hold the `Ctrl` key on your keyboard and click on five forms one by one: `LogIn Form.vb`, `Selection Form.vb`, `Faculty Form.vb`, `Course Form.vb`, and `Student Form.vb`, and `ConnModule.vb`. Click on the `Add` button to add these forms and module into our current project.

Now let's fix the error caused by our deleting the default form window.

1. Click on the **Show All Files** button on the top of the Solution Explorer window to display all files in the current project.
2. Expand **My Project** to **Application.myapp** and finally to the **Application.Designer.vb**. Double-click on this file to open it. The codes in this file are auto-generated by the system as you create a new project. Move to the bottom of this file and try to find a line of codes like:

```
Me.MainForm = Global.SQLSelectRTObject.Form1
```

3. The default main form is **Form1** when you create a new Window-based project. A blue-line appears under this **Form1** now since we have deleted this guy. Replace this default **Form1** with our **LogIn** form **LogInForm**.

Then you need to confirm that your startup form should be the **LogIn** form window. To do that,

1. Go to the **Project|SQLSelectRTObject Properties** to open the project property window.
2. Keep the **Application** tab selected and make sure that the **LogInForm** is located in the **Startup** form box. If not, select the **LogInForm** as the **Startup** form.

An easy way to develop the codes for this project is to replace the prefix **acc** that is preceded in all Data Provider-dependent objects in the last project, such as **accConnection**, **accCommand**, **accDataReader**, and **accDataAdapter**, with the prefix **sql**. Because the major difference between this project and the last one is the connection string, and most of the other codes are identical as long as the connection string is modified and matched to the selected database or the data provider.

Now open our module class **ConnModule.vb** by clicking on it the from the Solution Explorer window to begin our coding process.

5.19.2.1 *Declare the Runtime Objects*

As we mentioned in Chapter 3, all components related to the SQL Server Data Provider supplied by ADO.NET are located at the namespace **System.Data.SqlClient**. To access the SQL Server database file, you need to use this Data Provider. You must first declare this namespace at the top of each of your code window to allow Visual Basic.NET 2010 to know that you want to use this specific Data Provider. Enter the codes shown in Figure 5.107 to the **ConnModule** code window.

The namespace has been changed from the **System.Data.OleDb** to the **System.Data.SqlClient** since we need to use data components provided by the SQL Server Data Provider in this project. The connection instance has been changed to the **sqlConnection** with the **SqlConnection** class since we need this connection object for our whole project.

The first job you need to do is to connect your project with the database you selected after a new instance of the data connection object is declared.

5.19.2.2 *Connect to the Data Source with the Runtime Object*

Since 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 first event procedure, **Form_Load()** event procedure, to allow the connection to be made first as your project runs.



Figure 5.107. The declaration of the namespace for the SQL Server Data Provider.

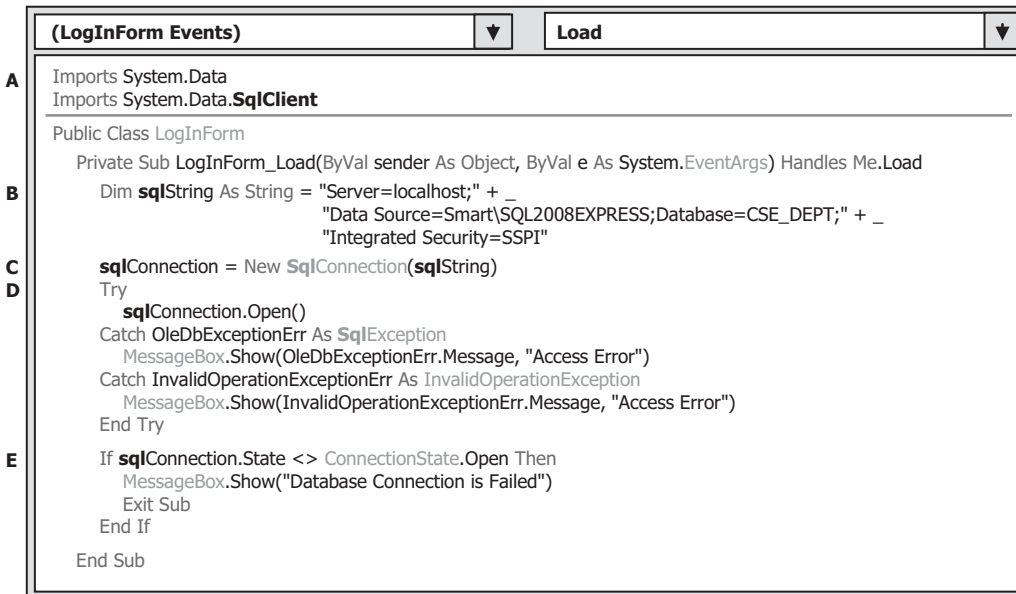


Figure 5.108. The codes for the database connection.

Open the code window for the LogIn Form and click on the drop-down arrow in the Class Name combo box and select the (LogInForm Events). Then go to the Method Name combo box and click on the drop-down arrow to select the Load method to open the LogInForm_Load() event procedure, which is shown in Figure 5.108. Enter the codes shown in Figure 5.108 into this event procedure.

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

- A.** The namespace `System.Data.SqlClient` is imported to the top of this code window since we need to use some data components related to the SQL Server Data Provider to query our sample SQL Server 2008 database.
- B.** An `SqlConnection` String is created first, and the connection string is used to connect your project with the SQL Server database selected. Please note that this connection string is different with the one we created in the last section. The Server parameter is assigned by 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

installed the server in our local computer, we need to use the local computer's name (Smart) followed by the server name (SQL2008EXPRESS). To identify your computer's name, right-click on the **My Computer** icon on your desktop screen, select and open the System Properties window, then click on the **Network Identification** 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. In this application, no username and password 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.

- C. A new instance of `SqlConnection` class is created with the connection string as an argument.
- D. A Try . . . Catch block is utilized here to try to catch up any mistake caused by opening this connection. The advantage of using this kind of strategy is avoiding unnecessary system debug process and simplifying 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, next we need to use this connection to access the SQL Server database to perform our data query job.

5.19.2.3 Coding for Method 1: Using the TableAdapter to Query Data

In this section, we will discuss how to create and use the runtime objects to query the data from the SQL Server database by using the TableAdapter method.

Open the `TabLogIn` button's Click event procedure by double clicking on the `TabLogIn` button and enter the codes shown in Figure 5.109 into this procedure.

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

- A. Since the query string applied in this application is relatively long, we break it into two substrings: `cmdString1` and `cmdString2`. Then we combine these two substrings to form a complete query string `cmdString`. One point you need to know is the relational operator applied in the SQL Server database, which is different with that used in the Microsoft Access database. The criteria of the data query are represented by using an equal operator that is located between the desired data column and a nominal parameter in Microsoft Access. But in the SQL Server database, this equal operator is replaced by a comparison operator `LIKE`. Another point is that another method is used to add the parameters into the Parameters collection. Unlike the method we utilized in the last section, here, we first create two `SqlParameter` objects, and initialize these two objects with the parameter's name and dynamic data value separately.
- B. The Command object `sqlCommand` is created based on the `SqlCommand` class and initialized using a blank command constructor.
- C. Two dynamic parameters are assigned to the `SqlParameter` objects, `paramUser-Name` and `paramPassWord`, separately. The parameter's name must be identical with the name of 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.
- D. Two parameter objects are added into the Parameters collection that is the property of the Command object using the `Add()` method, and the command object is ready to be used. It is then assigned to the method `SelectCommand()` of the TableAdapter.

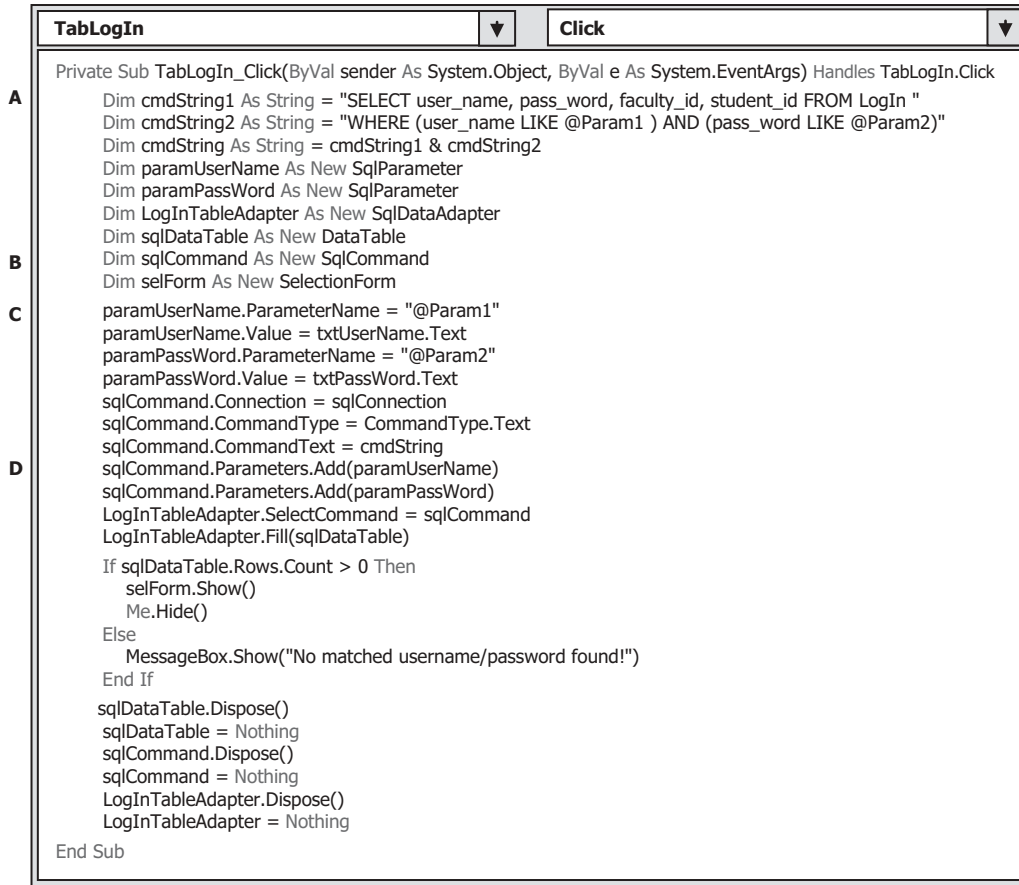


Figure 5.109. The codes for the TabLogIn button event procedure.

The rest of the codes is identical with those we developed in the last section, and a detailed explained has been given in the last section, too.

Now let's take a look at the codes for the second method.

5.19.2.4 Coding for Method 2: Using the 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 event procedure. Enter the codes shown in Figure 5.110 into this event procedure.

Most codes in the top section are identical with those codes in the TabLogIn button's event procedure with two exceptions. First, a DataReader object is created to replace the TableAdapter to perform the data query, and, second, the DataTable is removed from this event procedure, since we do not need it for our data query in this method.

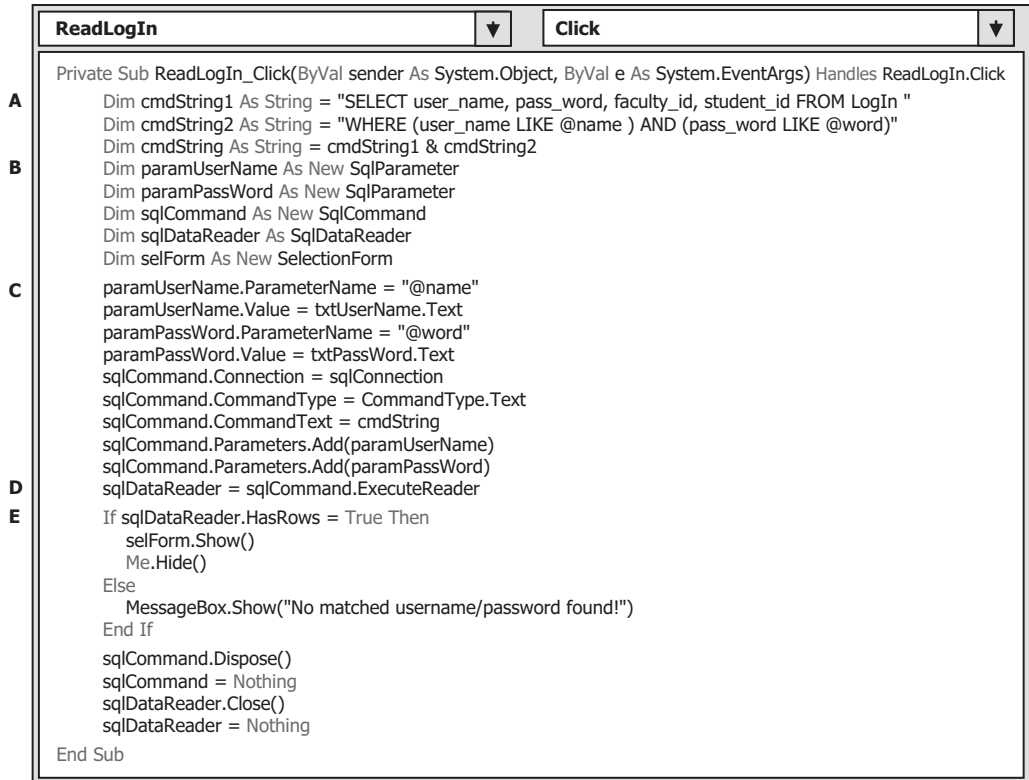


Figure 5.110. The codes for the ReadLogIn button event procedure.

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

- A.** As we did in the coding for method 1, the comparator LIKE is used to replace the equal operator in the second query string, and this is the requirement of the data query format for the SQL Server database.
- B.** Two `SqlParameter` 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, `@name`. The nominal value must be identical with that defined in the SQL query statement. The same situation is true for the value of the second nominal parameter `@word`.
- D.** The `ExecuteReader()` method is called to perform the data query, and the returned data should be filled in the `DataReader`.
- E.** 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.

The rest of the codes is identical with the codes we did in the last section.

The codes for the **Cancel** command button event procedure are similar with the codes we did in the last section, and the only difference is the prefix of the Connection instance. Change the prefix for each Connection instance from **acc** (**accConnection**) to **sql** (**sqlConnection**) sincere we are using an SQL Data Provider, and the Connection is a Data Provider-dependent component.

Next, let' handle the coding process for the Selection Form window.

5.19.3 The Coding for the Selection Form

Most codes in this form are identical with those 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 database is used, and all Data Provider-dependent objects are preceded with a prefix **acc**, such as **accConnection**. In this project, we used an 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 been closed and released. Since the connection object is created as a global variable in the **ConnModule** class, we can directly use this connection object from this **SelectionForm**. The only modification is to change the prefix **acc** to **sql** for the connection instance, which is highlighted in bold and shown in Figure 5.111.

5.19.4 Query Data Using Runtime Objects For the Faculty Form

First, let's take a look at the codes for the **Form_Load()** event procedure. The differences between this piece of codes with those in the last project are:

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

- A. The namespace of the Data Provider. The **System.Data.OleDb** was utilized for the last project since an Access database is used. Since we use the SQL Server database in this section, change that namespace to **System.Data.SqlClient**, which is shown in Figure 5.112.
- B. The prefix of the connection object is changed to **sql**, since an SQL Server Data Provider is utilized in the project.

The next coding is for the **Select** button event procedure. Open the Faculty form window by clicking on the View Designer button from the Solution Explorer window,

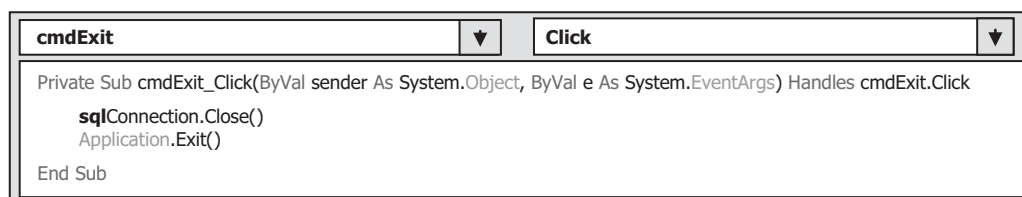


Figure 5.111. The modified codes for the Exit button event procedure.

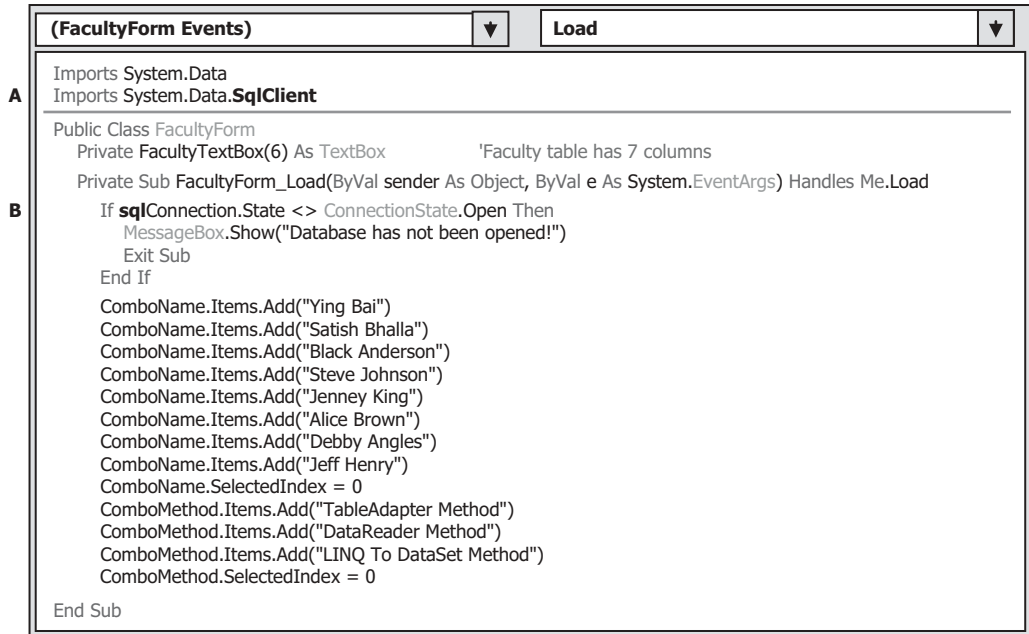


Figure 5.112. The modified codes for the FacultyForm_Load() event procedure.

then double click on the **Select** button to open its event procedure. Make the modifications shown in Figure 5.113 for this event procedure.

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

- A.** The query string is modified, especially for the query qualification. LIKE is used to replace the equal operator. Also, the name of the dynamic parameter is changed from @Param1 to @facultyName.
- B.** An SqlParameter object is created, and it is used to hold the dynamic parameter's name and value later.
- C.** Two Data Provider-dependent objects are created, and they are: sqlCommand and sqlDataReader. The sqlDataTable is a Data Provider-independent object.
- D.** The DataSet instance is used for data query using the LINQ to DataSet method.
- E.** The SqlParameter object is initialized by assigning it with parameter's name and parameter's value.
- F.** The sqlCommand object is initialized by assigning it with three values.
- G.** Starting from step G until step Q, replace the prefix acc for all Data Provider-dependent objects with the prefix sql, such as accCommand to sqlCommand, accDataTable to sqlDataTable, and accDataReader to sqlDataReader, since we are using an SQL Server Data Provider to perform the data query in this section.

For three user-defined subroutine procedures, FillFacultyTable(), MapFacultyTable(), ShowFaculty(), and the Back button's Click event procedure, there are no any modifications. The only modification, which is for the user-defined subroutine FillFacultyReader(),

	cmdSelect	▼	Click	▼
	Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click			
A	Dim cmdString1 As String = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty "			
	Dim cmdString2 As String = "WHERE faculty_name LIKE @facultyName"			
	Dim cmdString As String = cmdString1 & cmdString2			
B	Dim paramFacultyName As New SqlParameter			
	Dim FacultyTableAdapter As New SqlDataAdapter			
C	Dim sqlCommand As New SqlCommand			
	Dim sqlDataReader As SqlDataReader			
	Dim sqlDataTable As New DataTable			
D	Dim ds As New DataSet()			
E	paramFacultyName.ParameterName = "@facultyName"			
	paramFacultyName.Value = ComboName.Text			
F	sqlCommand.Connection = sqlConnection			
	sqlCommand.CommandType = CommandType.Text			
	sqlCommand.CommandText = cmdString			
G	sqlCommand.Parameters.Add(paramFacultyName)			
	Call ShowFaculty(ComboName.Text)			
	If ComboMethod.Text = "TableAdapter Method" Then			
	FacultyTableAdapter.SelectCommand = sqlCommand			
	FacultyTableAdapter.Fill(sqlDataTable)			
H	If sqlDataTable.Rows.Count > 0 Then			
	Call FillFacultyTable(sqlDataTable)			
	Else			
	MessageBox.Show("No matched faculty found!")			
	End If			
I	sqlDataTable.Dispose()			
J	sqlDataTable = Nothing			
	FacultyTableAdapter.Dispose()			
	FacultyTableAdapter = Nothing			
	ElseIf ComboMethod.Text = "DataReader Method" Then			
K	sqlDataReader = sqlCommand.ExecuteReader			
L	If sqlDataReader.HasRows = True Then			
	Call FillFacultyReader(sqlDataReader)			
	Else			
	MessageBox.Show("No matched faculty found!")			
	End If			
M	sqlDataReader.Close()			
N	sqlDataReader = Nothing			
	Else '----- LINQ To DataSet method is selected			
O	FacultyTableAdapter.SelectCommand = sqlCommand			
	FacultyTableAdapter.Fill(ds, "Faculty")			
	Dim facultyinfo = From fi In ds.Tables("Faculty").AsEnumerable()			
	Where fi.Field(Of String)("faculty_name").Equals(ComboName.Text) Select fi			
	For Each fRow In facultyinfo			
	txtID.Text = fRow.Field(Of String)("faculty_id")			
	txtName.Text = fRow.Field(Of String)("faculty_name")			
	txtTitle.Text = fRow.Field(Of String)("title")			
	txtOffice.Text = fRow.Field(Of String)("office")			
	txtPhone.Text = fRow.Field(Of String)("phone")			
	txtCollege.Text = fRow.Field(Of String)("college")			
	txtEmail.Text = fRow.Field(Of String)("email")			
	Next			
	End If			
P	sqlCommand.Dispose()			
Q	sqlCommand = Nothing			
	End Sub			

Figure 5.113. The modified codes for the Select button event procedure.

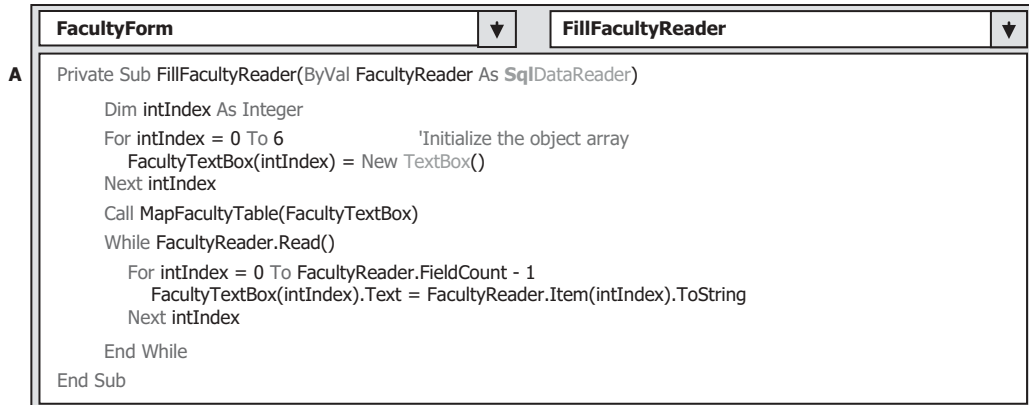


Figure 5.114. The modified codes for the subroutine FillFacultyReader().

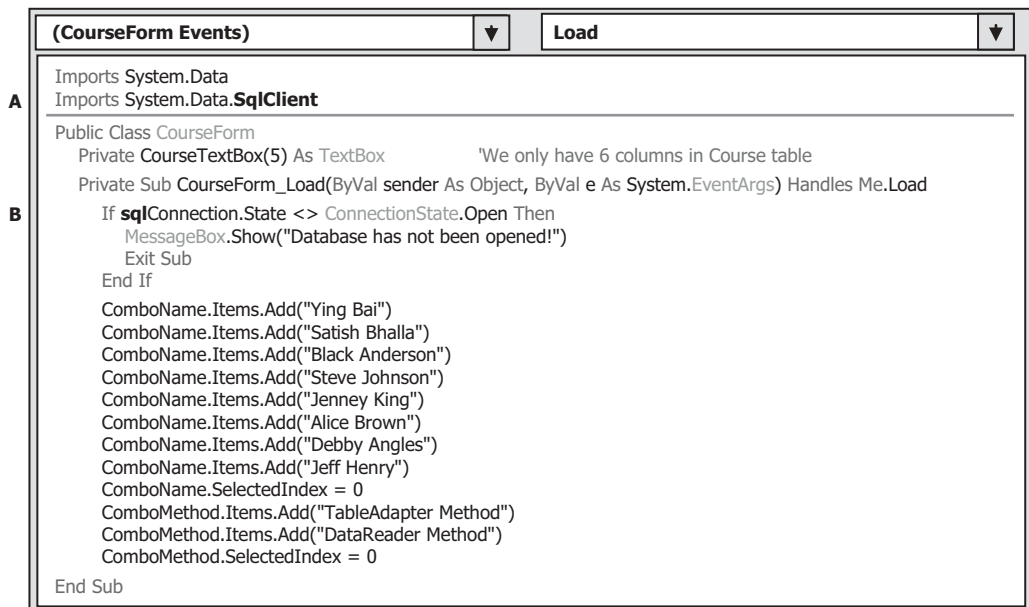


Figure 5.115. The modified codes for the CourseForm_Load() event procedure.

is to change the data type of the input argument FacultyReader from the OleDbDataReader to SqlDataReader, as shown in step **A** in Figure 5.114.

Now we have finished the coding process for the Faculty Form, next, let's develop the codes for the Course Form.

5.19.5 Query Data Using Runtime Objects for the Course Form

First, let's do the coding for the CourseForm_Load() event procedure.

Basically, the codes of this event procedure are similar to those we did for the same event procedure in the last project. The only modifications are (refer to Fig. 5.115):

- A. Data Provider namespace modification. The `System.Data.SqlClient` namespace is used to replace the original `System.Data.OleDb` since we are using an SQL Server data provider in this section.
- B. The original prefix `acc` is replaced by `sql` for the connection object.

The next coding job is for the **Select** button event procedure. This piece of codes is similar to those we did in the same event procedure in the last project. However, one important improvement made to this piece of codes is that an inner join query is utilized to simplify the data query. Recall that in the last project, we used two queries to finish the query for the courses taught by the selected faculty in the Course form. 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 functionality.

5.19.6 Retrieve Data from Multiple Tables Using Tables JOINS

To have a clear picture why we need to use the Join query method for this data action, let's first take a look at the data structure in our sample database. A part of 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. What we want to do is to pick up all `course_id` from the Course table based on the selected faculty name that is 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` 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 those `course_id` records. 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

Table 5.8. A 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	faculty_id	classroom
Computers in Society	A52990	TC-109
Computers in Society	A52990	TC-109
Introduction to Programming	J33486	TC-303
Introduction to Programming	B78880	TC-302
Algorithms & Structures	A77587	TC-301
Programming I	A77587	TC-303
Introduction to Algorithms	H99118	TC-302

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 key.

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 as the project runs.

One point to be noted 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 Course form. The modified codes are shown in Figure 5.116.

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

- 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 and selects the `course_id` from the CourseList box. We only need the `course_id` column for this query, but it does not matter if other columns, such as the course column, is included in this query. The assignment operator `LIKE` is used to replace the original equal symbol 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. Some new SQL objects are created, such as the `CourseTableAdapter`, `SqlCommand`, `SqlDataReader`, and `SqlDataTable`. All of these objects should be prefixed by the keyword `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 combo box, which should be entered by the user as the project runs later.
- D. The following codes are similar to those we developed in the last project. If the `TableAdapter` method is selected by the user, the `Fill()` method of the `TableAdapter` is executed to fill the Course table. The `FillCourseTable()` subroutine is called to fill the `course_id` into the CourseList box.
- E. Otherwise, the `DataReader` method is selected by the user and the `Execute-Reader()` method is executed to read back all `course_id`, and the `FillCourseReader()` subroutine is called to fill the `course_id` into the CourseList box.
- F. Finally, some cleaning jobs are preformed to release objects used for this query.

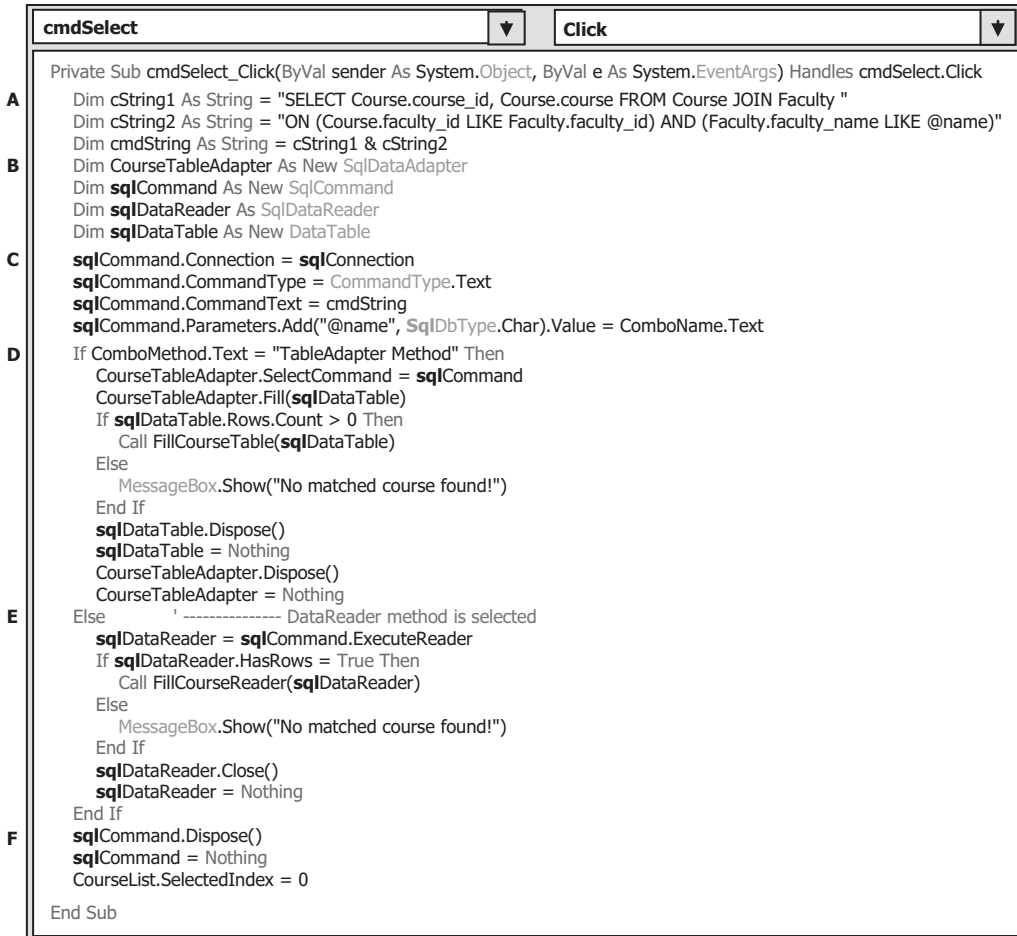


Figure 5.116. The modified codes for the Select button event procedure.

Three user-defined subroutine procedures `FillCourseTable()`, `MapCourseTable()`, `FillCourseTextBox()` and the Back button's Click event procedure have nothing to do with any object used in this project, so no coding modification is needed. However, two other user-defined subroutine procedures, `FillCourseReader()` and `FillCourseReaderTextBox()`, need only one small modification, which is to change the data type of the argument `CourseReader` from the `OleDbDataReader` to `SqlDataReader`, since now we are using an SQL Server data provider. An example of this modification is shown in step **A** in Figure 5.117.

Next, we need to take care of the coding for the `CourseList_SelectedIndexChanged()` event procedure.

All detailed information related to the selected `course_id` from the `CourseList` box should be displayed in six textbox controls when the user clicked and selected a `course_id` from the `CourseList` box control. The codes for this event procedure are similar with those we did in the same event procedure in the last project, with the modifications shown in Figure 5.118.

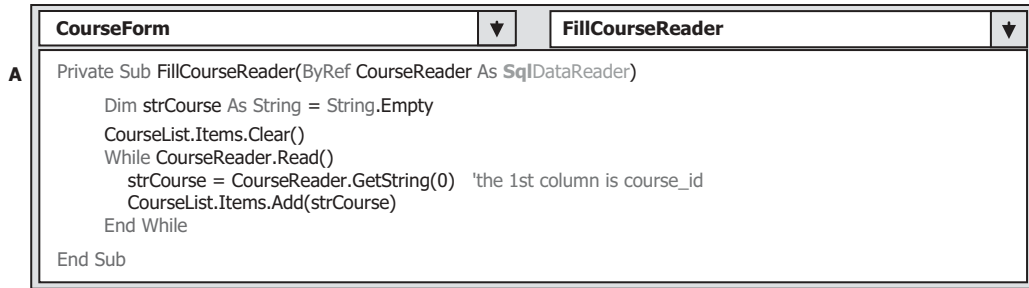


Figure 5.117. The modified codes for the user-defined subroutine FillCourseReader.

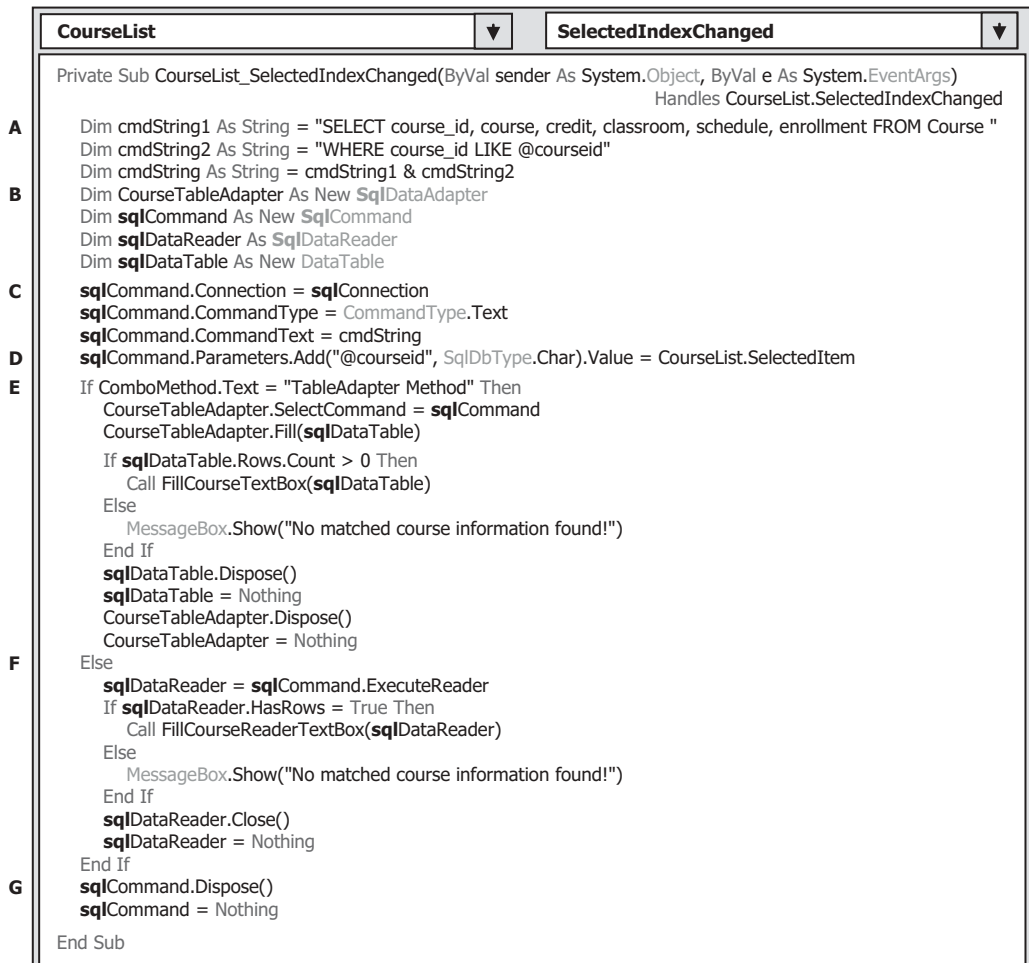


Figure 5.118. The modified codes for the CourseList_SelectedIndexChanged procedure.

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

- A. The query string is created with six queried columns, such as `course_id`, `course`, `credit`, `classroom`, `schedule`, and `enrollment`. 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 equal symbol for the criteria in the `WHERE` clause in the definition of the query string, and this is required by 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 the keyword `Sql` and all 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 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.
- E. If the `DataAdapter Method` is selected by the user, the `Fill()` method is called to fill the `Course` table, and the user-defined subroutine procedure `FillCourseTextBox()` is executed to fill six 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 subroutine procedure `FillCourseReaderTextBox()` is called to fill those pieces of course information into six textboxes.
- G. Finally, a cleaning job is performed to release objects used for this query.

Replace the prefix `acc` with the prefix `sql` for all Data Provider related components in this piece of codes.

You can test the codes we just developed for the `CourseForm` class by running the project now. Do not forget to copy all faculty image files to the folder in which your Visual Basic executable file is located before you can run this project. In this application, it is the `Debug` folder of the project.

5.19.7 Query Data Using Runtime Objects for the Student Form

Now let's finally come to the coding process 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, student name, gpa, credits, major, school year, and email, and display them in seven 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.

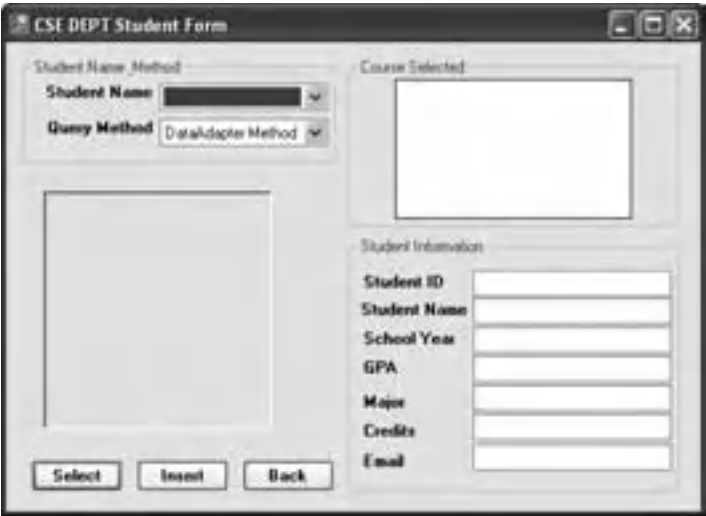


Figure 5.119. The Student form window.

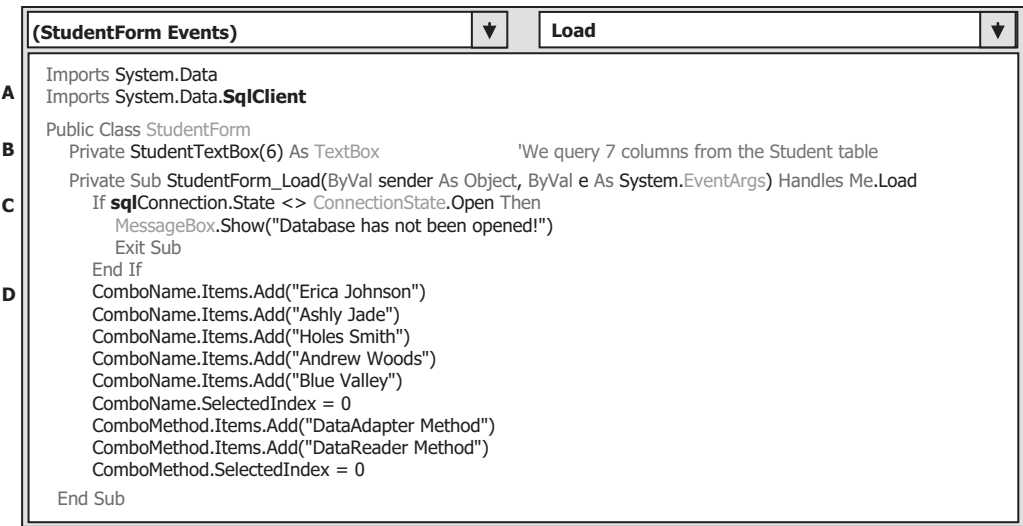


Figure 5.120. The codes for the Form_Load event procedure.

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 codes simple, two stored procedures are developed and implemented in this section. By using the stored procedures, the query can be significantly simplified and integrated, and the efficiency of the data query can also be improved.

Let's start from the Form_Load event procedure. The codes for this event procedure are shown in Figure 5.120.

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

- A.** The namespaces of the SQL Server data class library are imported to provide the prototypes of all data components to be created and used in this procedure.
- B.** A form-level 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 seven textboxes in the Student form later.
- C.** The database connection is checked first before we can perform any data operation between the project and the database related.
- D.** All sampled students' names and query methods are added into the related combo box, and the default item is the first one in both combo boxes.

Next, let's take a look at the coding process for the **Select** button Click event procedure. As we mentioned at the beginning of this section, when this **Select** button is clicked by the user, seven pieces of student information are displayed in seven related textboxes, and the courses (`course_id`) taken by that student are displayed in the CourseList box. Regularly, two queries are needed for this operation. However, in order to save time and space, we want to use two stored procedures to replace two queries to improve the query integrity and efficiency. Let's go a little deep for the stored procedure.

5.19.8 Query Student 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 codes developed in the Visual Basic.NET environment. You can easily call the stored procedure from your Visual Basic.NET project as the project runs. The result of using the stored procedure is that the performance of your data-driven application can be greatly improved, and the data query's speed can be significantly higher. 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.

5.19.8.1 Create the 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 Basic.NET 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.

The prototype or syntax of creating a stored procedure is in Figure 5.121.

For the SQL Server database, the name of the stored procedure is always prefixed by the keyword **dbo**. A sample stored procedure **StudentInfo** is shown in Figure 5.122.

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 in here. All 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 results.

```
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. The prototype of an SQL Server stored procedure.

```
CREATE PROCEDURE dbo.StudentInfo
{
    @StudentName VARCHAR(50)
}
AS
SELECT student_id FROM Student
WHERE student_name LIKE @StudentName
RETURN
```

Figure 5.122. A sample SQL Server stored procedure.

5.19.8.2 Call the Stored Procedure

When the stored procedure is created, it is ready to be called by your project that was developed in Visual Basic.NET. You can use any possible ways to finish this calling. For example, you can use the Fill() method defined in the TableAdapter to fill a data table, or you can use the ExecuteReader() method to return the queried result to a DataReader object. Both 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 Visual Basic.NET project, one needs to follow the syntax described as below (Fill() method in TableAdapter):

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 if you have
4. Execute the stored procedure by using the Fill() method in the TableAdapter class

Figure 5.123 shows a piece of example codes that illustrate how to call a stored procedure named `dbo.StudentInfo` (assuming a Connection object has been created):

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

- A. Some useful data components are declared here, such as the TableAdapter, Command, and DataTable.
- B. The Command object `CmdStudent` 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 create the stored procedure.

```

A Dim StudentTableAdapter As New SqlDataAdapter
  Dim sqlCmdStudent As New SqlCommand
  Dim sqlStudentTable As 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
  StudentTableAdapter.SelectCommand = sqlCmdStudent
F StudentTableAdapter.Fill(sqlStudentTable)
  If sqlStudentTable.Rows.Count > 0 Then
    Collect the retrieved data columns....
  Else
    MessageBox.Show("No matched student found!")
  End If

```

Figure 5.123. An example of calling the stored procedure.

- E. The stored procedure `dbo.StudentInfo` needs one input parameter `StudentName`, so a real parameter that will be obtained from the Student Name combo box 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 Select-Command property of the TableAdapter, 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 part, we will use our `Student` form to illustrate how to create two stored procedures and how to call them from our Visual Basic.NET project.

5.19.8.3 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, and the second one is used to obtain the courses taken by the selected student based on the `student_id`. The reason why we need to use two queries is: we want to query all courses (`course_ids`) taken by the selected student based on the student's name, not the `student_id`, from the `StudentCourse` table. But only the `student_id` column is available in the `StudentCourse` table, and there is no student name available 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 (exactly 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 Visual Studio.NET 2010 and open the Server Explorer window by clicking on the `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. Select the `Microsoft SQL Server Database File` item and click on the `OK` button.
3. Click on the `Browse` button to go to our database file folder: `C:\Program Files\Microsoft SQL Server\MSSQL10.SQL2008EXPRESS\MSSQL\DATA`, and select our database file `CSE_DEPT.mdf` by clicking on it, and then click on the `Open` button.
4. Click on the `Test Connection` button to confirm this connection.

Your finished `Add Connection` dialog box is 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 wizard, which is shown in Figure 5.124b.

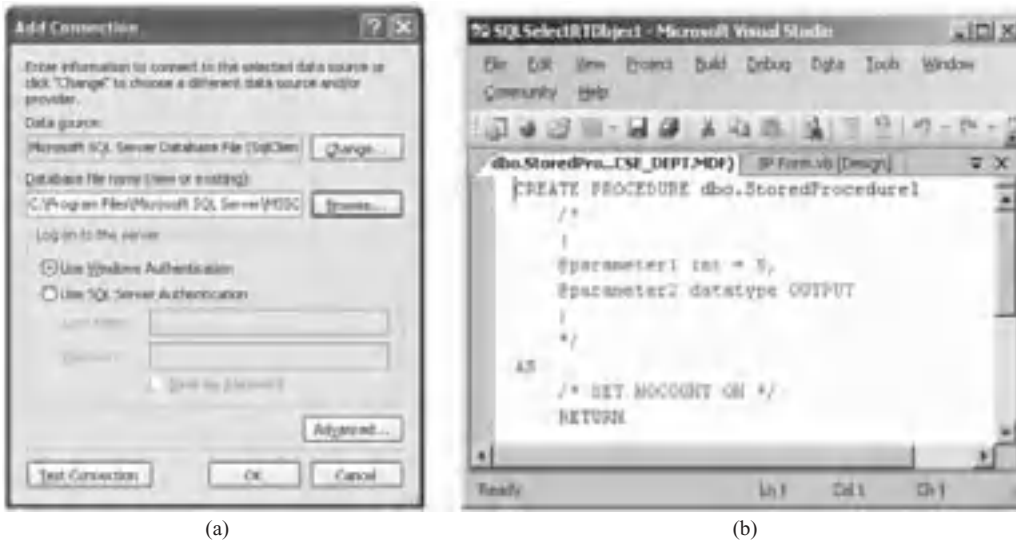


Figure 5.124. The Add Connection dialog box.

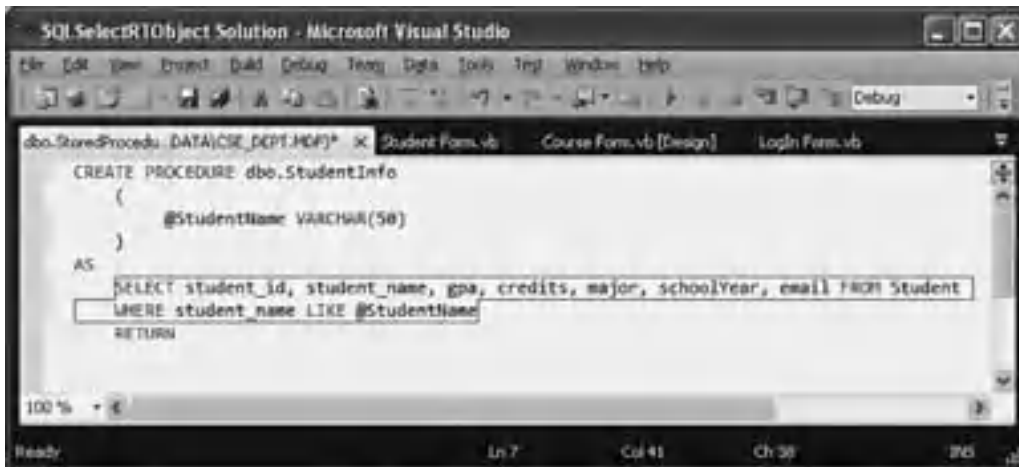


Figure 5.125. The first stored procedure—dbo.StudentInfo.

The default name for a new stored procedure is `dbo.StoredProcedure1`, which is located immediately after the keyword `CREATE PROCEDURE`. The top green-color codes, which are commented out by the comment symbol (`/* . . . */`), is used to create the parameters or a parameter list. The bottom green-color code is used to create the SQL statements. To create our first stored procedure, remove the comment-out symbols and enter the codes shown in Figure 5.125 into this procedure.

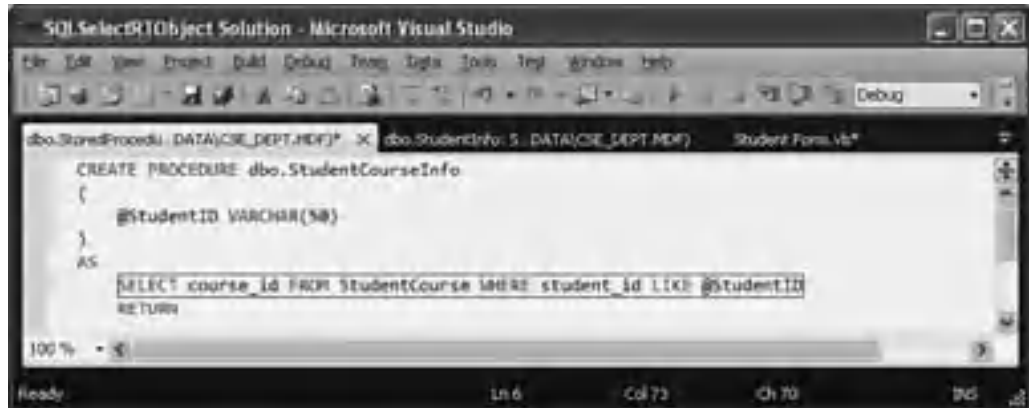


Figure 5.126. The second stored procedure—`dbo.StudentCourseInfo`.

The `@StudentName` is our only input parameter to this stored procedure, and this stored procedure will return seven 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 on the **FileSave StoredProcedure1** item to save our first stored procedure.

Similarly, we can create our second stored procedure `dbo.StudentCourseInfo`, which is shown in Figure 5.126.

The only input parameter to this stored procedure is `@StudentID`, and this stored procedure will return all courses (exactly `course_id`) taken by the selected student based on the input parameter `student_id`. Click on the **FileSave StoredProcedure2** to save our second stored procedure.

Ok, now we finished creating our two stored procedures. Next we need to develop the codes for our **Select** button Click event procedure in the Student form window to call these two stored procedures.

But wait for a moment. Before we can continue to develop our Visual Basic.NET codes 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-built stored procedure by using some pop-up menu items, which is shown in Figure 5.127a.

Open the Visual Studio.NET 2010 if it is not opened and connect to our database `CSE_DEPT` from the Server Explorer window. Expand the **Stored Procedure** folder and right click on any of our stored procedure. A pop-up menu will be displayed, which is shown in Figure 5.127a. The function 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.

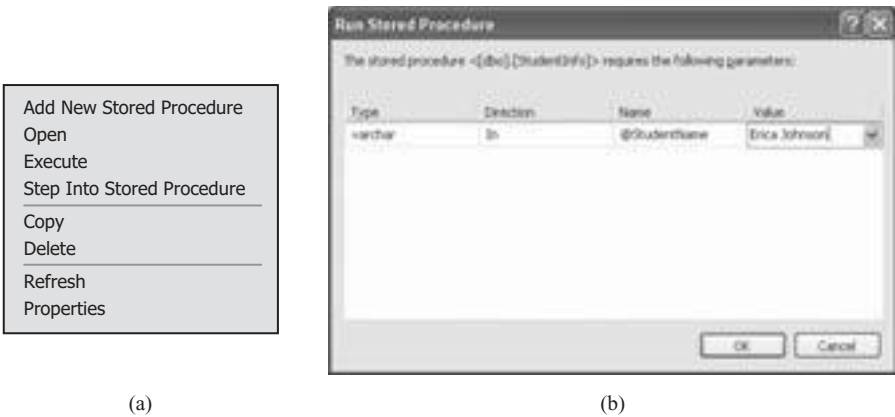


Figure 5.127. The pop-up menu and EXECUTE dialog box.

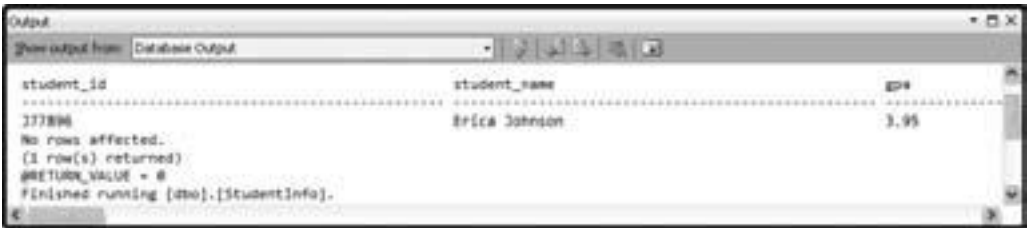


Figure 5.128. The testing result of our first stored procedure.

- 4. Step Into Stored Procedure:** Allow users to debug and run the developed stored procedure step by step.
- 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 on 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 the sample students' names, **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, which is shown in Figure 5.128.

In total, there is only one row with seven columns returned: **student_id**, **student_name**, **gpa**, **credits**, **major**, **schoolYear**, and **email**. Click on the right-arrow bar to view all columns.

In a similar way, you can try to run our second stored procedure. You need to enter a valid **student_id** as the input parameter to run it. Of course, you can use the **student_id**



Figure 5.129. The testing result for our second stored procedure.

we obtained from our first stored procedure, which is **J77896**. The testing result for our second stored procedure is shown in Figure 5.129.

Now that our two stored procedures have been tested successfully, it is time for us to develop our coding in Visual Basic.NET to call these two stored procedures.



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 to the server to open the database if you performed some tasks with the Server Explorer, such as creating stored procedures, because your server has been connected and the database is opening when you create stored procedures. An error message would be displayed if you try to do that since this Express version only allows one server instance to be connected at a time. You have to disconnect that connection first by rebooting your computer.

Open the GUI of the **Student** form and double-click on the **Select** button to open its event procedure, and enter the codes shown in Figure 5.130 into this event procedure.

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

- A.** Two stored procedures' names are assigned to two string variables **strStudent** and **strStudentCourse**, respectively. These two names must be identical to those we created in two stored procedures: **dbo.StudentInfo** and **dbo.StudentCourseInfo**.
- B.** All data components are declared and created in this section, which include two TableAdapters, two DataTables, two Command objects, and a local string variable **strName**.
- C.** The user-defined subroutine **FindName()** 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.** Two image properties, **SizeMode** and **Image**, are used to format and display the student's photo in the student's picture box.

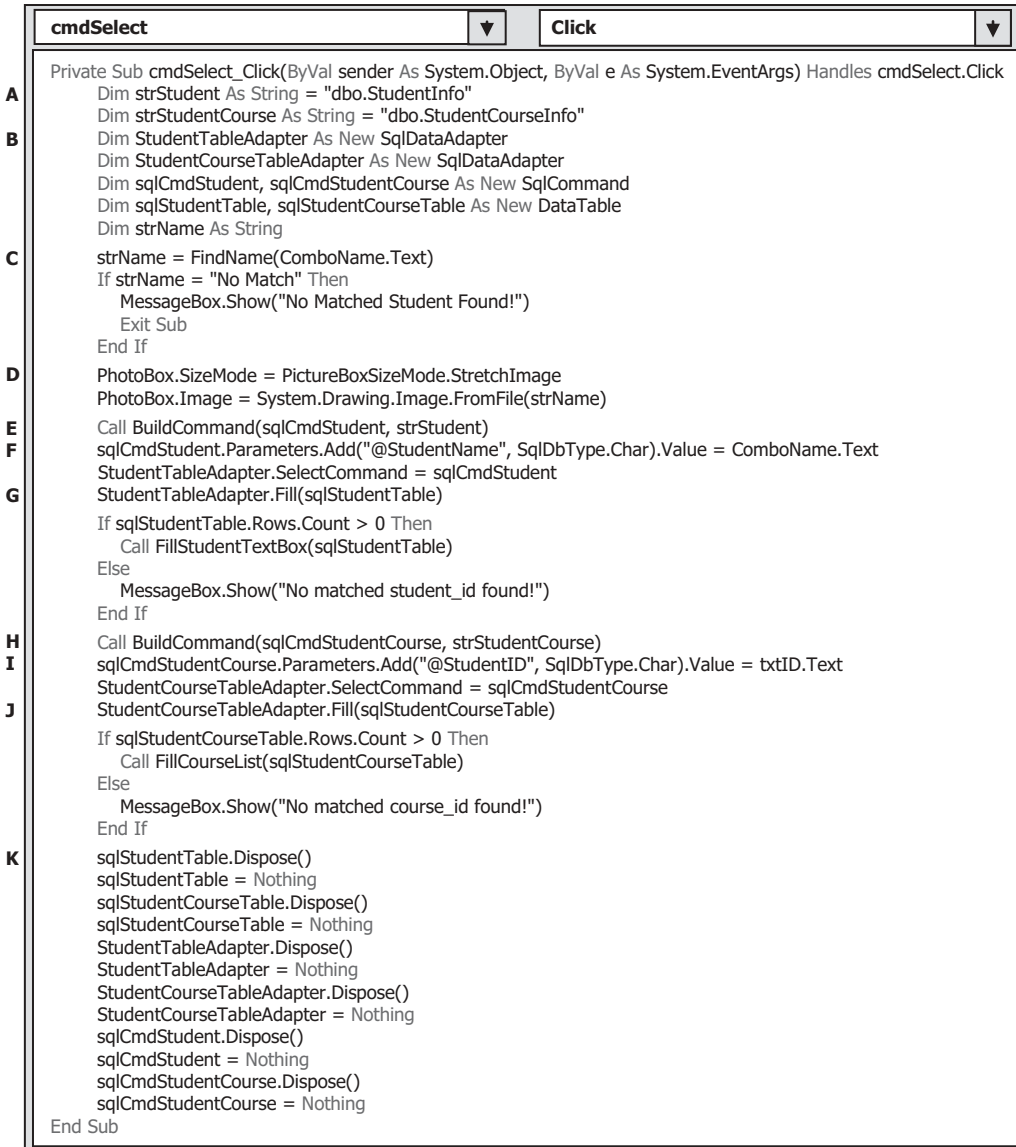


Figure 5.130. The codes for the Select button Click event procedure.

- E.** The subroutine `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 initialized 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.

- F. The unique input parameter to the stored procedure `dbo.StudentInfo` is the `StudentName`, which will be selected by the user from the student name combo box (`ComboName.Text`) as the project runs. 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 `sqlCmdStudent` is then assigned to the `SelectCommand` property of the `TableAdapter` to make it ready to be used in the next step.
- G. The `Fill()` method of the `TableAdapter` is called to fill the `Student` table, which is to call our first stored procedure to fill the `Student` table. If this calling is successful, the `Count` property should be greater than 0, which means that at least one row has been filled into the `Student` table, and the other user-defined subroutine `FillStudentTextBox()` is called to fill seven textboxes in the `Student` form with seven pieces of retrieved columns from the stored procedure. Otherwise, an error message is displayed if this fill has failed.
- H. The subroutine `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 is 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 `TableAdapter` to make it ready to be used in the next step.
- J. The `Fill()` method of the `TableAdapter` is called to fill the `StudentCourse` table, which is to call our second stored procedure to fill the `StudentCourse` table. If this calling is successful, the `Count` property should be greater than 0, which means that at least one row has been filled into the `StudentCourse` table, and the subroutine `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.
- K. The cleaning jobs are performed to release all data objects used in this event procedure.

The codes for the `BuildCommand()` subroutine are shown in Figure 5.131. The modifications to this user-defined subroutine procedure are:

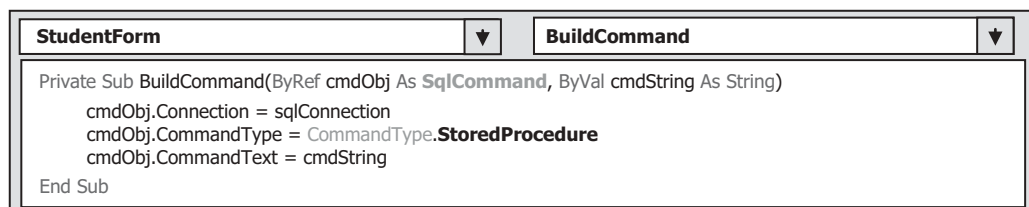


Figure 5.131. The codes for the subroutine `BuildCommand`.

1. Change the data type of the first passed argument `cmdObj` from the `OleDbCommand` to `SqlCommand` since we are using data components related to the SQL Server Data Provider to perform these queries.
2. Change the `CommandType` property to `CommandType.StoredProcedure` since we need to call the stored procedure to perform the student data query in this procedure.

The codes for all other four user-defined subroutine procedures, `FindName()`, `FillStudentTextBox()`, `MapStudentTextBox()`, and `FillCourseList()`, are identical with those we developed in the last project `AccessSelectRTOject`.

The codes for the **Back** button Click event procedure is simple; open that event procedure and just enter `Me.Close()` into that procedure.

One point to be noted is that in order to pick up the correct student's image file from the subroutine `FindName()`, you must store all the students' image files in the folder in which your Visual Basic.NET project's executable file is located. In our application, this folder is `C:\Chapter 5\SQLSelectRTOject\bin\Debug`. If you place those students' image files in other folders, 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 subroutine.

Now we can begin to run this project to call those two stored procedures from our Visual Basic.NET project. Click on the Start Debugging button to run our project, enter the username and password, and select the **Student Information** item to open the Student form window, which is shown in Figure 5.132.

Select a student, such as **Ashly Jade**, from the Student Name combo box and click on the **Select** button. All information related to this student and the courses are displayed in seven textboxes and the CourseList box, which is shown in Figure 5.132.

Our project to call two stored procedures is very successful!

Some readers may find that these two stored procedures are relatively simple, and each procedure only contains one SQL statement. Ok, let's dig a little deeper and develop

The screenshot shows a Windows application titled "CSE DEPT Student Form". It has a tabbed interface with the "Student Name Method" tab selected. The "Student Name" dropdown menu shows "Ashly Jade". The "Query Method" dropdown menu shows "DataAdapter Method". Below these is a photo of a young woman. To the right, the "Course Selected" list box contains the following text: CSC-132B, CSC-234A, CSC-331, CSC-306. Below the photo and course list is the "Student Information" section, which contains the following fields: Student ID (A50950), Student Name (Ashly Jade), School Year (Junior), GPA (3.57), Major (Information Systems Engineering), Credits (118), and Email (jade@college.edu). At the bottom of the form are three buttons: Select, Insert, and Back.

Figure 5.132. The running status of the Student form.

some sophisticated stored procedures and try to call them from our Visual Basic.NET project. Next, we will develop a stored procedure that contains more SQL statements.

5.19.8.4 Query Data Using More Complicated Stored Procedures

In this section, we want to get all courses (exactly all `course_id`) taken by the selected student based on the 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 regularly perform two queries to complete this data-retrieving operation. Now, we try to combine these two queries into a 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, and click 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 sample 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 5.133 into this new procedure.

Let's give a detailed discussion about this piece of codes.

- A. The stored procedure is named `dbo.StudentCourseINTO`.
- B. The input parameter is the student name, `@stdtName`, which is a varying-char variable with the maximum characters of 50. All parameters, no matter if they are input or output, must be declared inside the braces.

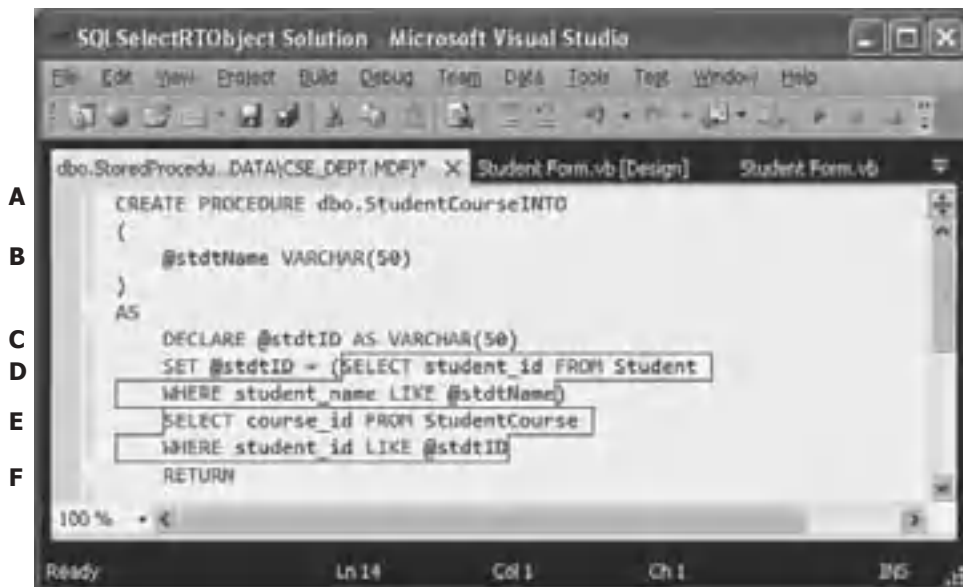


Figure 5.133. The newly stored procedure—StudentCourseINTO.

- C. The local variable `@stdtID` 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 `@stdtName`. A `SET` command must be used to assign the returned result from the first SQL query to the local variable (or intermediate variable) `@stdtID`. The first SQL statement must be covered by the parenthesis to indicate that this whole query will return a single data item.
- 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` (`@stdtID`) that is obtained from the first query.
- F. Finally, the queried result, all courses or `course_id`, are 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 newly 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 our procedure. The running result is shown in Figure 5.134.

We need to develop a Visual Basic.NET project to call this stored procedure to test the functionality of the stored procedure. To save time and space, we add a new form window into this project and named it as **SPForm**. Perform the following operations to create the codes for this new form:

1. Open our project `SQLSelectRTOject` and select **Project|Add Windows Form** item.
2. Enter `SP Form.vb` into the Name box and click on the **Add** button to add this new form into our project.
3. Enter `SPForm` into the Name property as the name for this form.
4. 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 this 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.



Figure 5.134. The running result of the stored procedure.

5. To save time, we need to copy most codes from the Student code window to our new SP form code window. The only exceptions are the codes for the **Select** button Click event procedure, `cmdSelect_Click()`, and the codes for the subroutine `BuildCommand()`. Don't copy these two pieces of codes since we need to develop new codes to test our stored procedure later.
6. To copy all other codes, open the code window of the Student form, select those codes, except the codes for the `cmdSelect_Click()` event procedure and subroutine `BuildCommand()`, and then paste them to our new SP form code window.
7. Now let's develop our codes for the **Select** button event procedure. Open the **Select** button click event procedure and enter the codes that are shown in Figure 5.135 into this event procedure.

Let's have a closer look at this piece of codes 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 our stored procedure in the Server Explorer window.
- B. A Command object and a DataReader object are declared here since we need to use them for our data query operation.
- C. The subroutine `FindName()` is called to get the matched student image file, and the returned image file is stored in the local string variable `strName`.

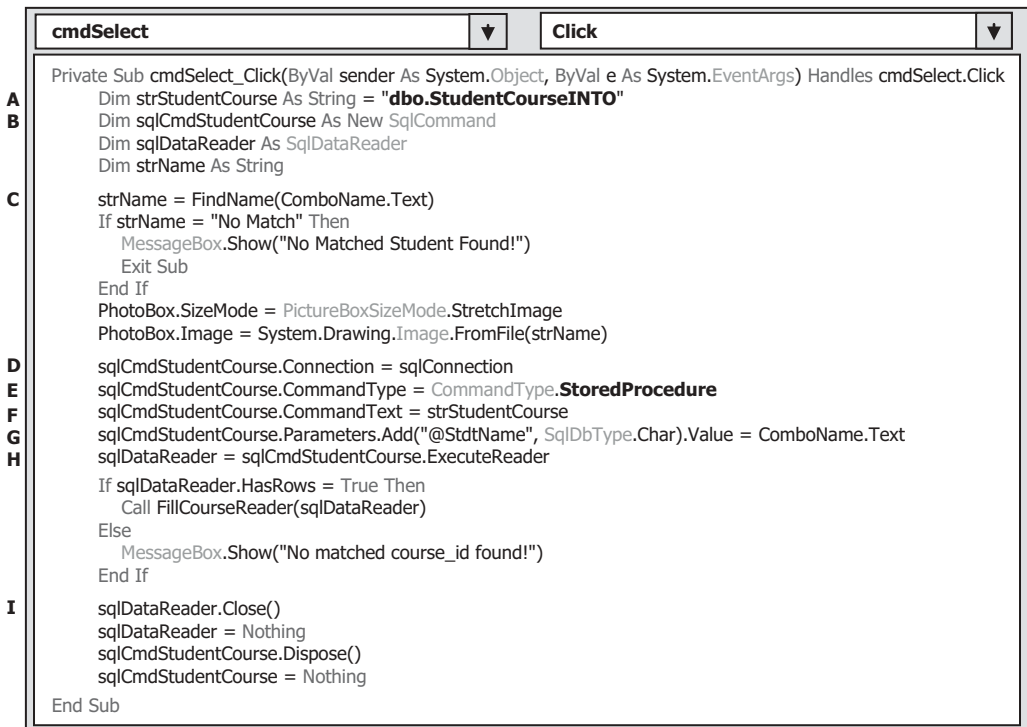


Figure 5.135. The codes for the Select button event procedure.

- D. The Command object is initialized with suitable properties: The first one is the Connection object.
- E. The CommandType property must be `StoredProcedure` to indicate that this query is to execute a stored procedure.
- F. The CommandText should be equal to the name of our stored procedure, `dbo.StudentCourseINTO`, which is stored in a string variable `strStudentCourse`.
- G. The input parameter is the student name, which is obtained from the student combo box, 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 parameter name we defined in the parameter braces in our stored procedure. The real parameter is entered by the user as the project runs.
- H. The `ExecuteReader()` method is executed to invoke the `DataReader` to call our stored procedure. If this call is successful, the queried result should be stored in the `DataReader` with returned rows. The user-defined subroutine `FillCourseReader()` is executed to fill the returned `course_id` into the `CourseList` box in this form. Otherwise, an error message is displayed if this call has failed.
- I. A cleaning job is performed to release all objects used in this data query operation.

For the detailed codes in the user-defined subroutine procedure `FillCourseReader()`, refer to Figure 5.117 in Section 5.19.6. You can copy the entire subroutine and paste it into this code window.

All other codes are identical to those we developed for the Student form, including all user-defined subroutines.

Before you can test this piece of codes, you need to add one more item into the Selection form code window to enable it to browse to our SP Form. Open the code window of the Selection form and enter the following codes into the `SelectForm_Load()` event procedure:

```
ComboSelection.Items.Add("SP Information")
```

Then add the following codes into the `OKButton_Click()` event procedure:

```
Dim spform As New SPForm
.....
ElseIf ComboSelection.Text = "SP Information" Then spform.Show()
```

Now run the project, 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 combo box and click on the **Select** button. All courses taken by selected student will be displayed in the `CourseList` box, which is shown in Figure 5.136.

In this project, we only used `DataReader` as the tool to call the stored procedure to retrieve our desired data from the database. As an option, you can consider to use the `Fill()` method of the `TableAdapter` class to fulfill the same functionality as the `DataReader` did in this project. We prefer to leave this job for students as their homework for this chapter.

At this point, we finished developing data-driven projects using the real-time object for the SQL Server database. 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.



Figure 5.136. The running status of calling stored procedure `dbo.StudentCourseINTO`.

5.20 QUERY DATA FROM ORACLE DATABASE USING RUNTIME OBJECT

To make it simple, in this section, we only installed the Oracle Database 11g Express Edition server in our local computer. It would be no difference whether the Oracle server is installed in the local or a remote computer for this sample project. The Oracle database used in this sample project is Oracle Database 11g Express Edition that was developed in Chapter 2.

5.20.1 Install and Configure the Oracle Database 11g Express Edition

In this section, we use the Oracle Database 11g Express Edition for our database provider. Refer to Appendix B for the detailed procedures to download, install, and configure this software on your computer.

Oracle Database 11g Express Edition (Oracle Database XE) 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 ISVs)

Oracle Database XE is built using the same code base as Oracle Database 11g 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 that exist for the Oracle Database 11g XE, such as up to 4 GB upper bound of 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 11g XE 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 11g Standard Edition One, Standard Edition, or Enterprise Edition, and the application development investment are guaranteed.
- With Oracle Database XE, ISVs have the industry's leading database technology to power their applications. Distributing Oracle Database XE in their applications or products without additional cost offers even greater value to their customers.
- Oracle Database XE can be freely distributed as a standalone 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 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 Studio.NET applications.

5.20.2 Configure the 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 11g XE. 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 as 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\11.2.0\server\NETWORK\ADMIN` after the Oracle Database 11g XE is installed. A sample file is shown in Figure 5.137. You can open this file using any text editor, such as Notepad, WordPad, or MS Word.

The database alias for our application is `XE`, and the top block of this file is the definition of the database alias (refer to Fig. 5.137).

Close this file, and now, let's create our connection string for the Oracle database 11g XE using the database alias `XE`.

The connection string can be defined as:

```
Dim oraString As String = "Data Source=XE;" + _
    "User ID=CSE_DEPT;" + _
    "Password=reback"
```

where the User ID `CSE_DEPT` is the name of our sample Oracle database, and the password `reback` is the password we used when we login to the APEX workspace for the Oracle Database 11g XE in our computer.

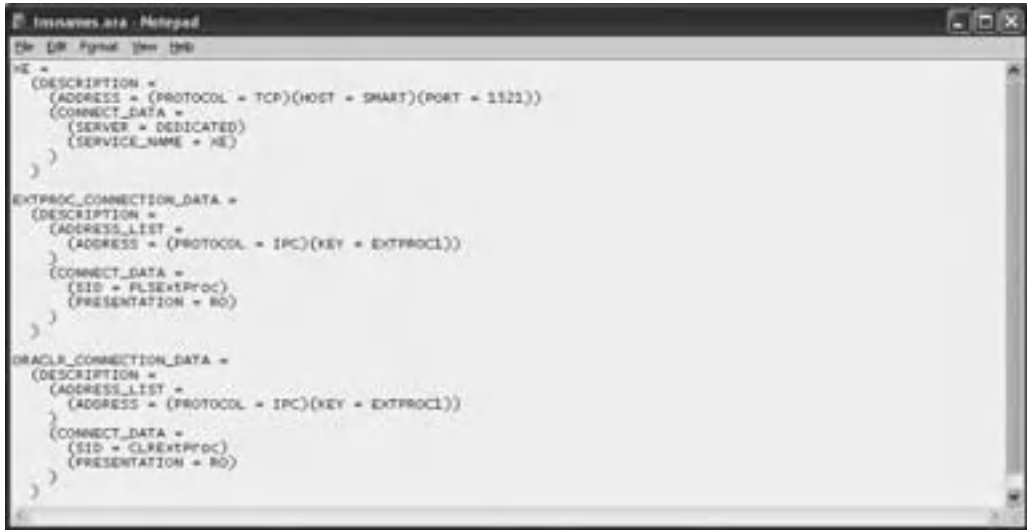


Figure 5.137. The sample of the file tnsnames.ora.

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:

```

Dim oraString As String = "Data Source=(DESCRIPTION=" + _
    "(ADDRESS=(PROTOCOL=TCP)(HOST=smart)(PORT=1521))"+_
    "(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME = XE));" + _
    "User ID=CSE_DEPT;Password=reback;"

```

In the following sample project, we will use the first way to create our connection string. With the connection string ready, now we can start to develop our sample project.

5.20.3 Query Data Using Runtime Objects for the Login Form

Open Visual Studio.NET 2010 and create a new Windows-based project named OracleSelectRTOject. As we did for the last section, delete the default form Form1, and perform the following operations to add five form windows and ConnModule into this project:

1. Go to the folder VB Forms\Window located at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1) to find all five form windows.
2. Click on the Project/Add Existing Item, and browse to the VB Forms\Window located at the Wiley ftp site (you can download and temporarily save those forms in one of your local folders, such as Temp).
3. Press and hold the Ctrl key on your keyboard and click on five forms one by one: Login Form.vb, Selection Form.vb, Faculty Form.vb, Course Form.vb, and Student Form.vb

and `ConnModule.vb`. Click on the **Add** button to add these forms and module into our current project.

Refer to Section 5.19.2 to modify the codes for the file `Application.Designer.vb`. Make sure that the `LogIn` form is the start form in this project by checking the `Project\OracleSelectRTOObject Properties` window.

Now let's add some Oracle Data Provider references to our project. Perform the following operations to complete this addition operation:

1. Right-click on the `OracleSelectRTOObject` from the Solution Explorer window and select the **Add Reference** item from the pop-up menu to open the Add reference window.
2. With the `.NET` tab selected, scroll down the list until you find the items `Devart.Data` and `Devart.Data.Oracle`, click on both to select them, and click on the **OK** button to add these two references 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 `AccessSelectRTOObject` or `SQLSelectRTOObject`. The reason for that is because

- Starting from .NET Framework 4.0, Microsoft no longer support Oracle database related operations. Therefore, we need to use an Oracle database driver provided by a third-party vendor.
- The namespaces for SQL Server and Microsoft Access Data Providers are default namespaces, and all components related to those Data Providers have been added automatically by the Visual Studio.NET 2010 as you open a new project.

5.20.3.1 Declare the Runtime Objects and Modify the `ConnModule`

To access the Oracle database, you need to use an Oracle database driver provided by a third-party vendor. You must first declare the namespace for that driver at the top line of your code window to allow Visual Basic.NET 2010 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 shown in Figure 5.138 to the top of this code window.

A new instance of the `OracleConnection` class is created with the **Public** access mode, which means that we can use this connection object for our whole project.

The first job you need to do is to connect your project with the database you selected after a new instance of the data connection object is declared.

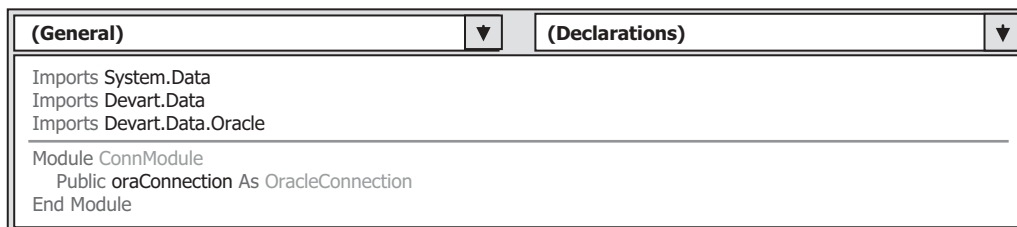


Figure 5.138. The declaration of the namespace for the Oracle Data Provider.

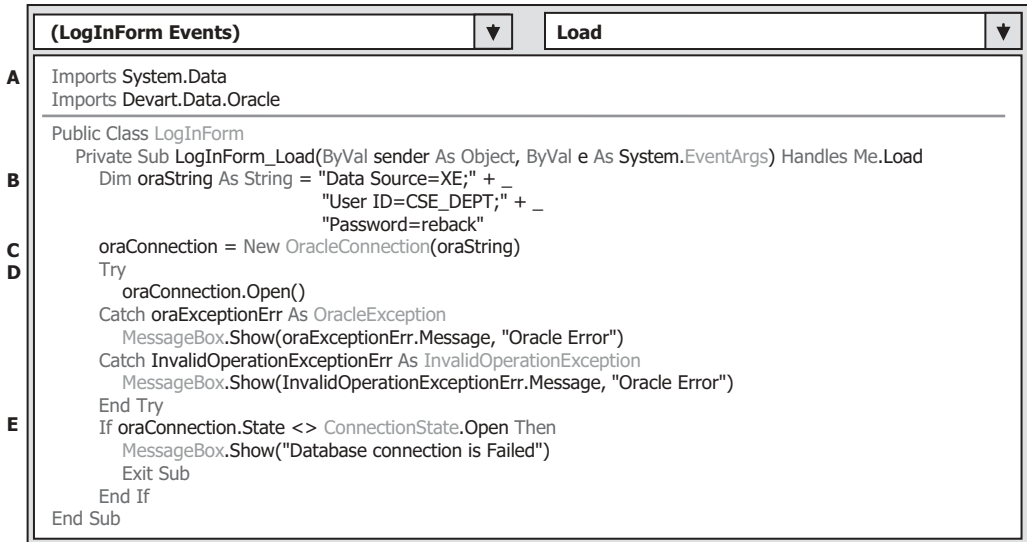


Figure 5.139. The codes for the LogInForm_Load() event procedure.

5.20.3.2 Connect to the Data Source with the Runtime Object

Since 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 LogInForm_Load() event procedure, to allow the connection to be made first as your project runs.

In the code window, click on the drop-down arrow in the Class Name combo box and select the (LogInForm Events). Then go to the Method Name combo box and click the drop-down arrow to select the Load method to open the LogInForm_Load() event procedure. Enter the codes shown in Figure 5.139 into this event procedure.

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

- A.** First, the namespaces for general data components and Oracle data provider are declared, since we need to use some components related to Oracle data provider.
- B.** The Oracle database connection string is defined first. Refer to Section 5.19.1 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 Oracle database.
- C.** A new Oracle Connection instance is created with the name oraConnection. This connection object is a Public variable, which means that it can be accessed by all event procedures in all forms defined in the current project.
- D.** A Try . . . Catch block is utilized here to try to catch up any mistake caused by opening this connection. The advantage of using this kind of strategy is avoiding unnecessary system debug process and simplifying 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, next, we need to use this connection to access the Oracle database to perform our data query job. As we did for the previous

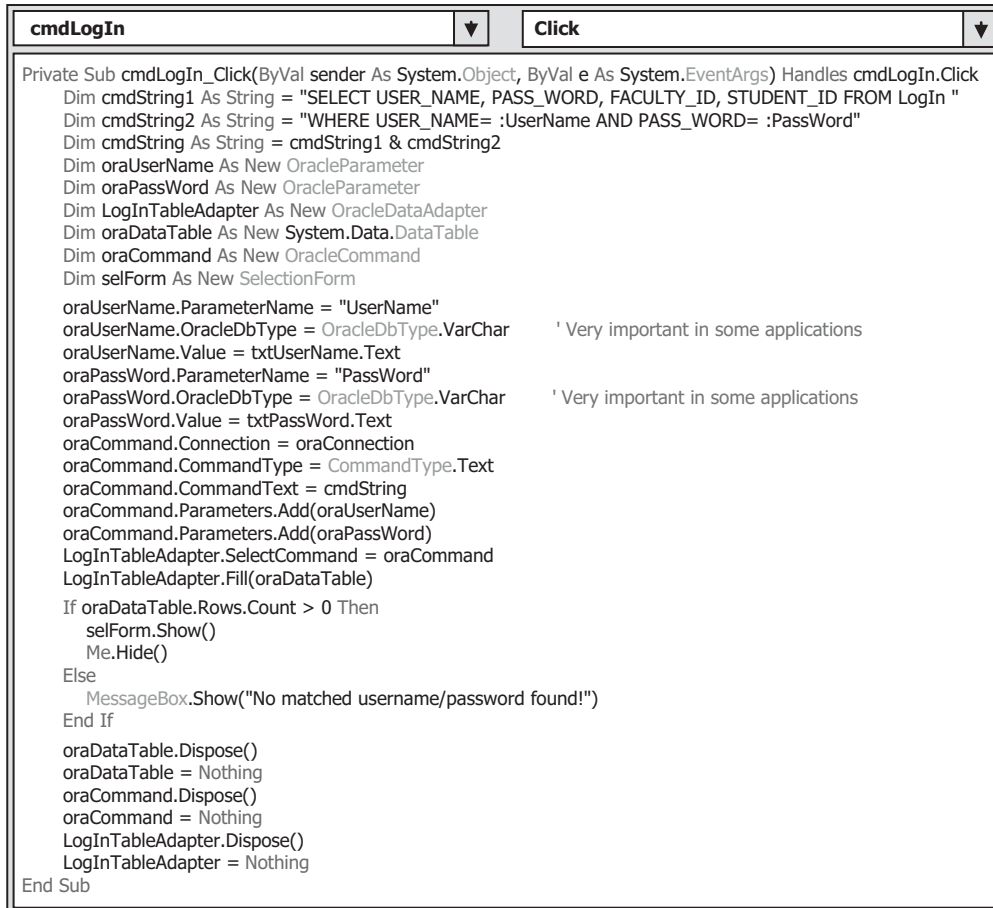


Figure 5.140. The codes for the LogIn button event procedure.

projects, we will use two methods to perform the data query: TableAdapter method and DataReader method.

5.20.3.3 Coding for Method 1: Using the TableAdapter to Query Data

Open the LogIn form window by clicking on the View Designer button, and then double-click on the LogIn button to open its event procedure. Enter the codes shown in Figure 5.140 into this event procedure.

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

- A.** The SELECT statement for Oracle is basically the same as for that used in the SQL Server database, but little difference is exist between them. The difference is the format of assigning the parameter in the WHERE clause. In both Microsoft Access and SQL Server, an @ symbol is prefixed before the parameter, and an equal symbol or a LIKE word is used to assign a parameter to the column. In Oracle, an equal symbol is still 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.
- 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 dynamic parameter in the SQL statement string. The parameter's type (`OracleType.VarChar`) 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. Two parameter objects are added into the Parameters collection that is the property of the Command object using the `Add()` method, and the command object is ready to be used. It is then assigned to the method `SelectCommand` of the `TableAdapter`.
- E. The `Fill()` method is called with the `oraDataTable` as the argument to fill the `LogIn` table.



The `SELECT` statement used for the Oracle database is different from that used for SQL Server and Microsoft Access. The difference is the format of assigning parameters to the columns in the `WHERE` clause. A colon must be prefixed before the parameter to be assigned to the column.

- F. 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. Otherwise, an error message will be displayed.
- G. A cleaning job is performed to release all data objects used for this data query.

Next, let's handle the coding for the second method.

5.20.3.4 Coding for Method 2: Using the `DataReader` to Query Data

To use this method, you need to add one more buttons named `cmdReadLogIn` with the Text of `ReadLogIn`. 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 event procedure. Enter the codes shown in Figure 5.141 into this procedure.

Most codes in the top section are identical with those codes in the `LogIn` button's event procedure with two exceptions. First, a `DataReader` object is created to replace the `TableAdapter` to perform the data query, and second, the `DataTable` is removed from this event procedure since we do not need it for our data query in this method.

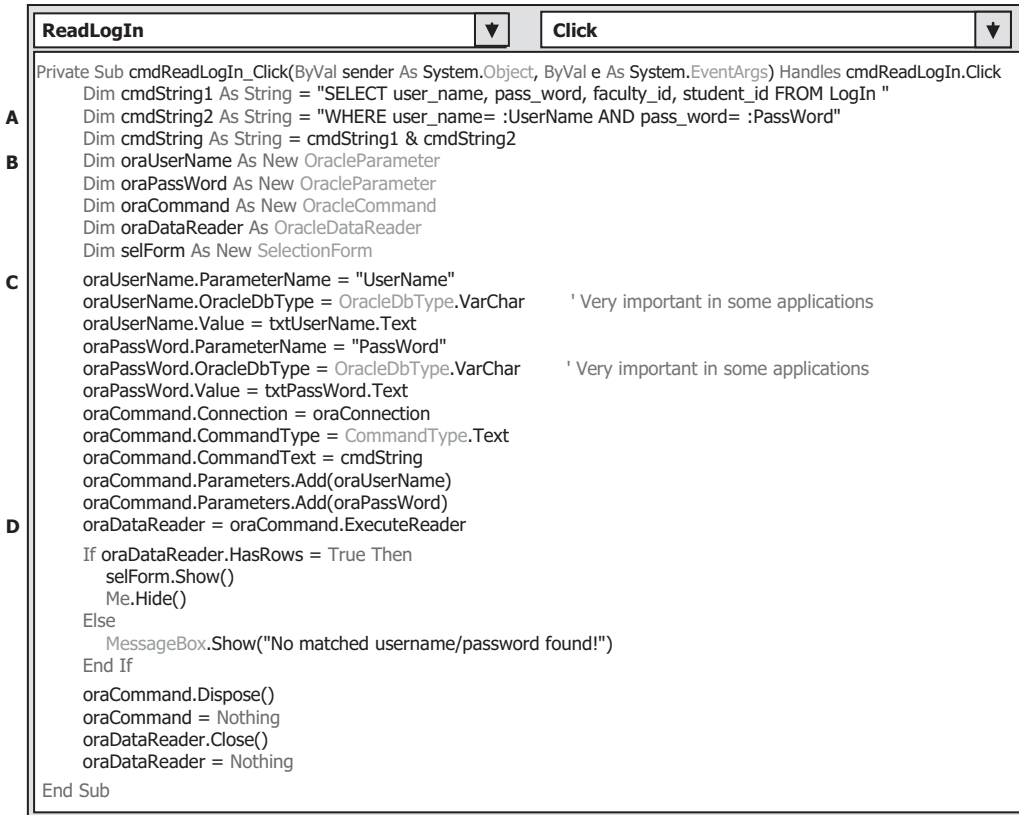


Figure 5.141. The codes for the ReadLogIn button event procedure.

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

- A.** As we did 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.
- D.** The `ExecuteReader()` method is called to perform the data query, and the returned data should be filled in the `DataReader`.

The rest codes are identical with those we did in the last section. The only issue you need to note is that all prefixes of the objects used in this part should be replaced by `ora`, such as `oraCommand`, `oraDataReader`, and so on, since we are using an Oracle database for this section.

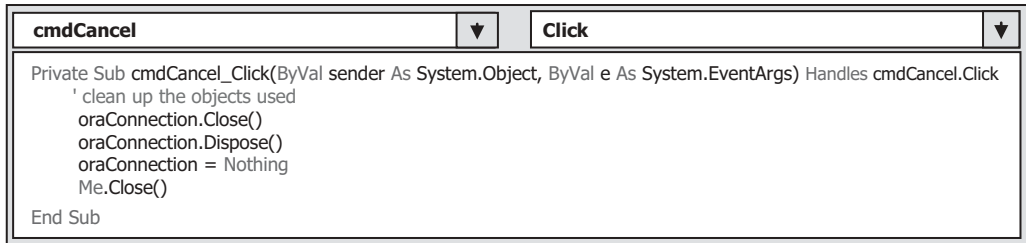


Figure 5.142. The codes for the Cancel button Click event procedure.

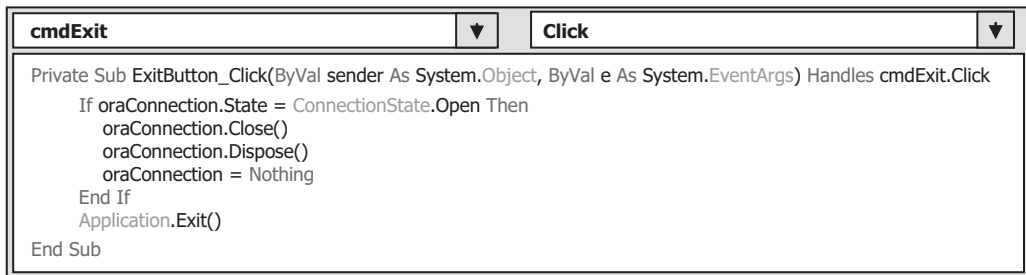


Figure 5.143. The codes for the Exit button event procedure.

The codes for the Cancel command button event procedure is basically identical with the codes we did in the last section, and the only modification is to change the prefix of the connection object from acc to ora, as shown in Figure 5.142.

If the login process is successful, the next form will be the Selection form that allows users to select different information to view.

5.20.4 The Coding for the 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, 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 so the connection object should be preceded by a prefix ora. When the Exit button is clicked, we need to check whether a valid connection is still exist in this project. If it is, we need to close this connection before we can exit the project. Open this event procedure and enter the codes shown in Figure 5.143 into this event procedure.

Next, let's handle the coding for the Faculty form to perform the data query from the Faculty table in our sample database.

5.20.5 Query Data Using Runtime Objects for the Faculty Form

In this section, we will develop three different query methods to perform the data query from the Faculty table in our sample database: DataAdapter, DataReader, and LINQ to

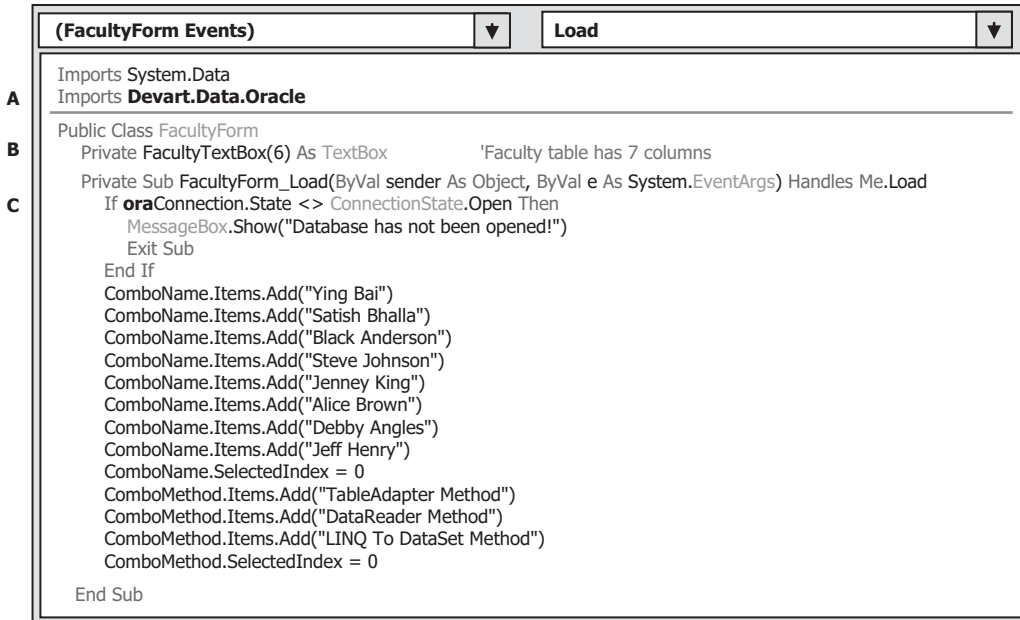


Figure 5.144. The codes for the FacultyForm_Load() event procedure.

DataSet method. First, let's take a look at the codes for the FacultyForm_Load() event procedure, which are shown in Figure 5.144.

The differences between this coding with the coding in the last project are:

- A.** The namespace of the Data Provider. The **Devart.Data.Oracle** is utilized for the namespace since we are using the Oracle database in this section; therefore, we need to declare this namespace in which all Oracle data components related to the Oracle Data Provider are included.
- B.** The FacultyTextBox object array used in this project is the same as that used in the last project.
- C.** The prefix of the connection object is changed to **ora** since an Oracle Data Provider is utilized in the project.

The next coding is for the **Select** button event procedure. Open the Faculty form window and the **Select** button Click event procedure. Enter the codes shown in Figure 5.145 into this event procedure.

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

- A.** The query string is declared first since we need query all seven columns from the Faculty table. 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.

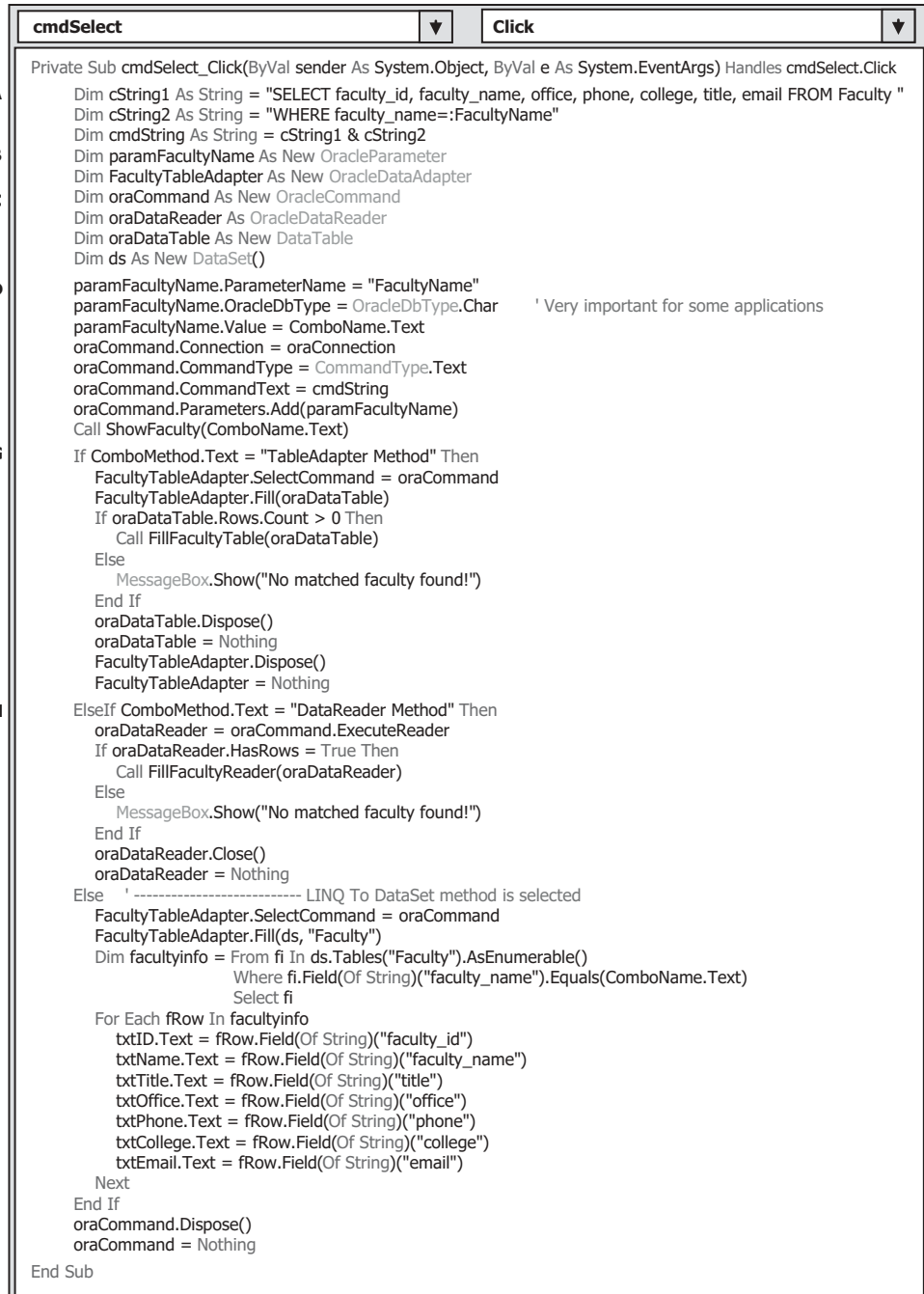


Figure 5.145. The codes for the Select button event procedure.

- C. Two Data Provider-dependent objects are created, and they are: `oraCommand` and `oraDataReader`. The `oraDataTable` is a Data Provider-independent object.
- D. The `OracleParameter` object is initialized by assigning it with the parameter's name, type, and parameter's value.
- E. The `OracleCommand` object is initialized by assigning it with three values.
- F. The initialized dynamic parameter `paramFacultyName` is assigned to the `Parameters` property of the `Command` object by using the `Add()` method.
- G. If the `TableAdapter Method` is selected by the users, the `Fill()` method is executed to fill the `Faculty` table.
- H. If the `DataReader Method` is selected, the `ExecuteReader()` method is called to read the `Faculty` data.
- I. If the `LINQ to DataSet Method` is selected, the `Faculty` table in the `DataSet` is filled by using the `Fill()` method. A `For Each` loop is used to fill the seven textbox controls in the `Faculty` form window with the queried seven data columns.

The rest of the codes are similar to those codes we developed in the same event procedure for the last project. You can copy those codes and paste them into this procedure. The only difference between this piece of codes and those in the last project is: the prefix `sql` preceded in front of each data object has been replaced by `ora`, such as `sqlCommand` by `oraCommand`, `sqlDataTable` by `oraDataTable`, and `sqlDataReader` by `oraDataReader`.

For three user-defined subroutine procedures, `FillFacultyTable()`, `ShowFaculty()`, and `MapFacultyTable()`, there is no modification at all, and you can copy them and paste them into this project. However, for the user-defined subroutine procedure, `FillFacultyReader()`, a small modification is needed, which is to change the data type of the passed argument `FacultyReader` from `OleDbDataReader` to `OracleDataReader`. Figure 5.146 (A) shows this modification.

The codes for the `Back` button `Click` event procedure is identical to those codes we did for the project `AccessSelectRTOObject`, with no modification. Just insert `Me.Close()` into this event procedure—yes, that is easy!

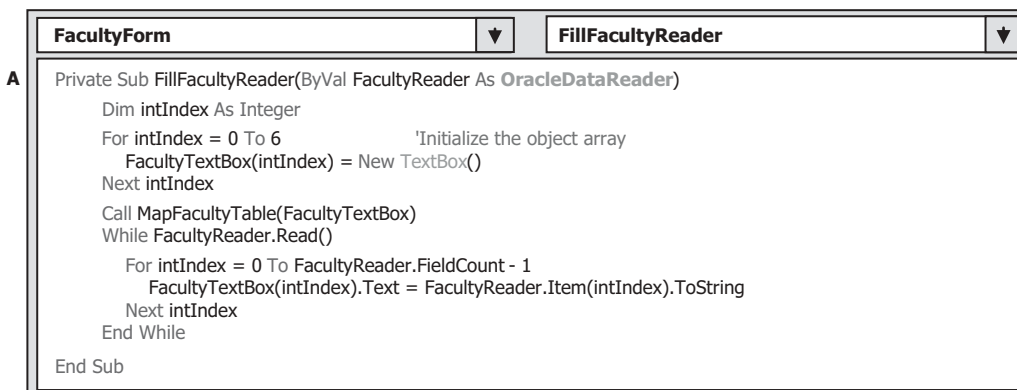


Figure 5.146. The modified codes for the subroutine `FillFacultyReader()`.

5.20.6 Query Data Using Runtime Objects and LINQ to DataSet for the 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 11g XE environment to replace the general data query commands to simplify the query structure and improve the query efficiency.

First, let's develop the codes for the **CourseForm_Load()** event procedure.

Basically, the codes of this event procedure are similar to those we did for the same event procedure in the last project. The only differences are (refer to Fig. 5.147):

- A.** Data Provider namespace modification. The **Devart.Data.Oracle** namespace is used to replace the original **System.Data.SqlClient** since we are using an Oracle data provider in this section.
- B.** The prefix of the Connection object has been changed from **sql** to **ora** since an Oracle Connection object is used in this project.
- C.** The third query method, **LINQ to DataSet**, has been added into this form to enable users to select this method to perform course data query.

The next coding job is for the **Select** button Click event procedure. After the user selected the desired data query method from the Method combo box and the faculty member from the Faculty Name combo box, the **Select** button is used to trigger its event procedure to retrieve all courses (**course_id**) taught by the selected faculty.

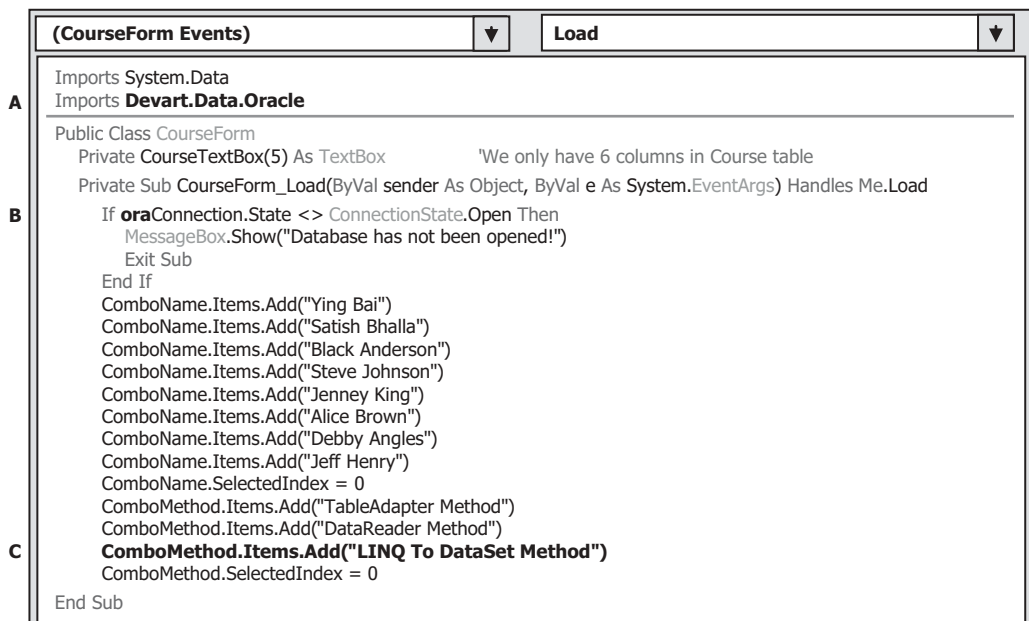


Figure 5.147. The modified codes for the CourseForm_Load() event procedure.

One point you need to note is that two queries are needed in this event procedure because there is no faculty name column available in the Course table, so 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 combo box in the Course form. Then we need to make the second query to find all courses (`course_id`) taught by the selected faculty from the Course table. The queried `course_id` are displayed in the CourseList box, and the detailed course information for each course can be displayed in six textboxes when the user clicks 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 as we discussed in Section 5.19.6, 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 in Section 5.19.8 for the data query operations with the Student table. In this section, we will use the stored procedure built in the Oracle database environment to perform these queries. You need to note that the stored procedures defined in the SQL Server database and the Oracle database are different, therefore, in the following section, we need first to provide a discussion about these two different kinds of stored procedures in two database environments.

5.20.7 The 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 2008, Microsoft provides the SQL Server Management Studio and the SQL Server Management Studio Express. For the Oracle database, Oracle provides Oracle Database 11g and Oracle Database 11g Express Edition. Different methods can be used to create stored procedures, for example, six methods are shown in Section 5.19.8.1 to create stored procedures for 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 SQL Workshop under the APEX Workspace in the Oracle Database 11g Express Edition to create stored procedures.

In Section 5.19.8.1, we discussed how to use the Server Explorer provided by the Visual Studio 2010 to create stored procedures. Similarly, Visual Studio 2010 also enables users to use Server Explorer to manage stored procedures built in the Oracle database, although Oracle has provided the Oracle development tools for the .NET.

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 2008, there is no difference between stored procedures using either an `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement, and all of these statements can be used by SQL Server stored procedures. However, in Oracle database, if a stored procedure needs to return data such that a stored procedure needs to execute an `SELECT` statement, 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 procedures, but 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.

5.20.7.1 The Syntax of Creating a Stored Procedure in the Oracle

The syntax of creating a stored procedure in the Oracle is shown in Figure 5.148.

The keyword **REPLACE** is used for the modified stored procedures. Recall that in the SQL Server 2008, 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 5.149.

The length of data type for each parameter is not necessary since this allows those parameters to have a varying-length value.

5.20.7.2 The Syntax of Creating a Package in the 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 5.150.

```
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 5.148. The 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);
END;
```

Figure 5.149. The syntax of creating a package in Oracle.

```

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 5.150. The 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 5.151. An example of creating a Faculty Package in Oracle.

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 the output parameters (cursor works as 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 be ended with a semicolon, including the command **END**.

An example of creating a **FacultyPackage** in the Oracle is shown in Figure 5.151.

The stored procedure is named **SelectFacultyID** with two parameters: the input parameter **FacultyName** and the output parameter **FacultyID**. The keywords **IN** and **OUT** that followed the associated parameter are used to indicate the input/output direction of the parameter. The length of the stored procedure name is limited to 30 letters in

the Oracle. Unlike the stored procedure name created in the SQL Server 2008, there is no prefix applied for each procedure's name.

Ok, 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—our Course form. 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 **course_id** 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 11g Express Edition (XE). In this application, we prefer to use the Object Browser page to do this job since it provides a GUI.

Unlike the SQL Server database, Visual Studio.NET 2010 does not provide a GUI 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 11g Express Edition did provide 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 11g XE to create our packages without installing and using the ODT.

5.20.8 Create the Faculty_Course Package for the Course Form

Open the Oracle Database 11g XE home page by going to the **start|All Programs|Oracle Database 11g Express Edition|Get Started** items. Perform the following operations to create this package:

1. Click on the **APEX** button to open the Login to APEX page.
2. Enter **SYSTEM** and reback into the Username and Password box to complete the login process for the APEX.
3. Since we have already created our sample database **CSE_DEPT** in Chapter 2, click on the **Already have an account? Login Here** button.
4. Enter reback into the Password box and click on the Login button.
5. Click on the **SQL Workshop** icon to open this workshop window.
6. Click on the **Object Browser** icon and click on the drop-down arrow on the **Create** button, and select the **Package** item to open the Create Package wizard, which is shown in Figure 5.152.

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 5.153.

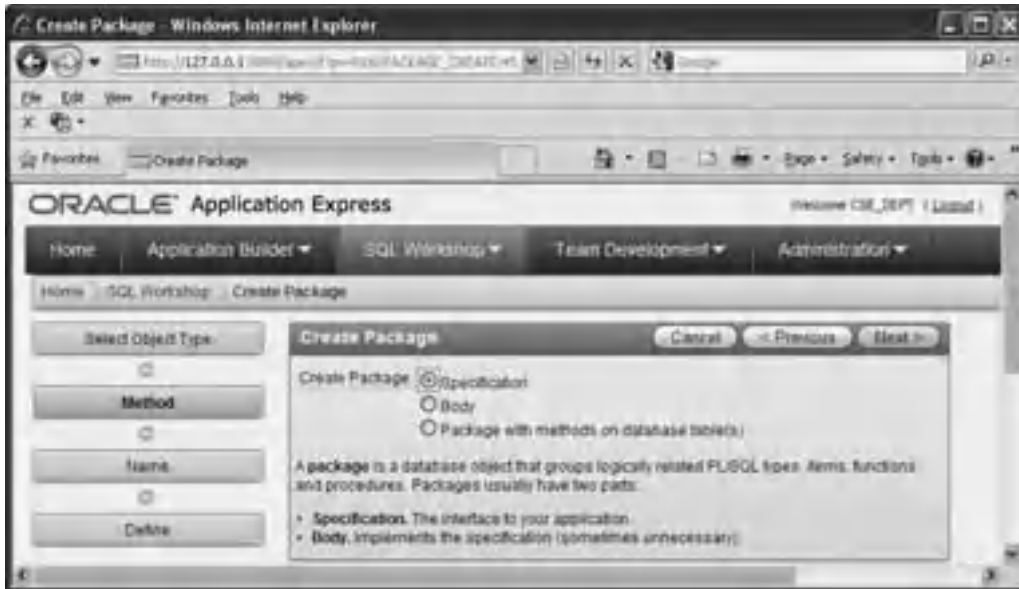


Figure 5.152. The opened Create Package wizard.

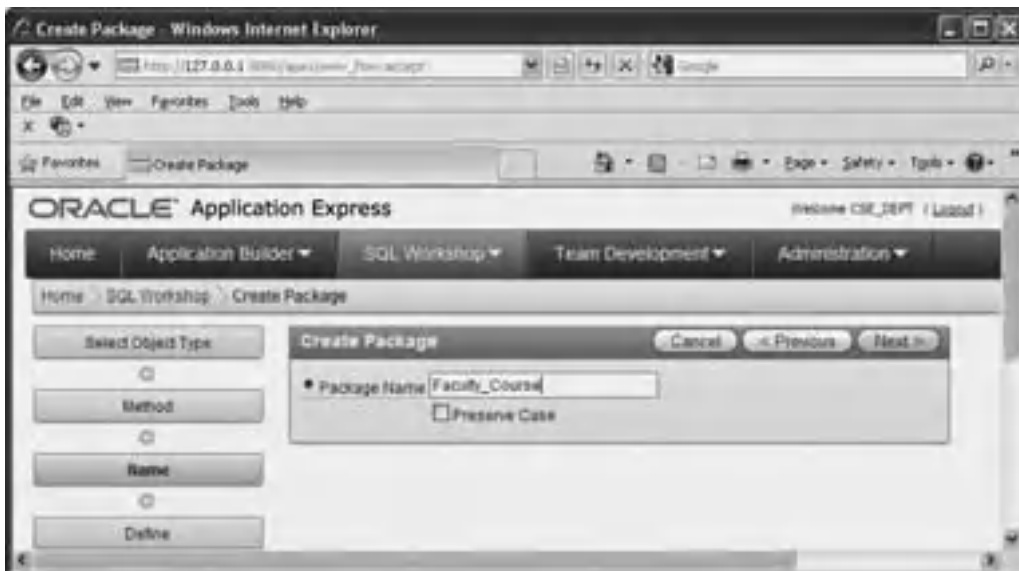


Figure 5.153. The Name page of the Package wizard.



Figure 5.154. The opened Specification page.

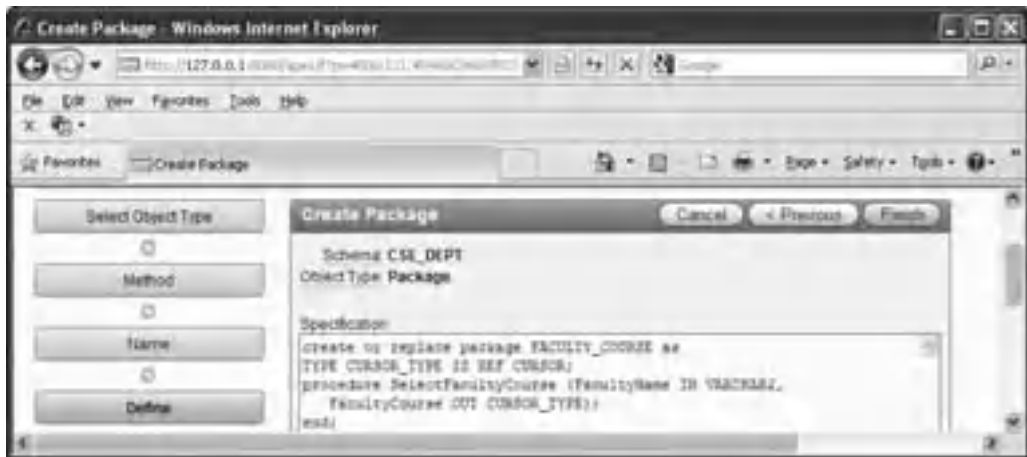


Figure 5.155. The codes for the Specification page.

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 5.154.

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 for our application, remove the default function prototype, and change the default procedure name from the `test` to our procedure name—`SelectFacultyCourse`. Your finished codes for the specification page should match the one that is shown in Figure 5.155.

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 you 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 `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 a keyword `OUT`. Each PL-SQL statement must be ended by a semi-colon, and this rule is also applied to the `END` statement.

Click on the **Finish** button to complete this step. To confirm this specification, you can click on the **Save & Compile** button to compile this specification block. A successful compiling page, as shown in Figure 5.156, should be displayed if this coding is fine.

Next, we need to create the body block of this package. Click on the **Body** tab that is next to the **Specification** tab located on the first row to open the Body page, which is shown in Figure 5.157.

Enter the PL-SQL codes shown in Figure 5.158 into this body.

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 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 that is located at line 6.

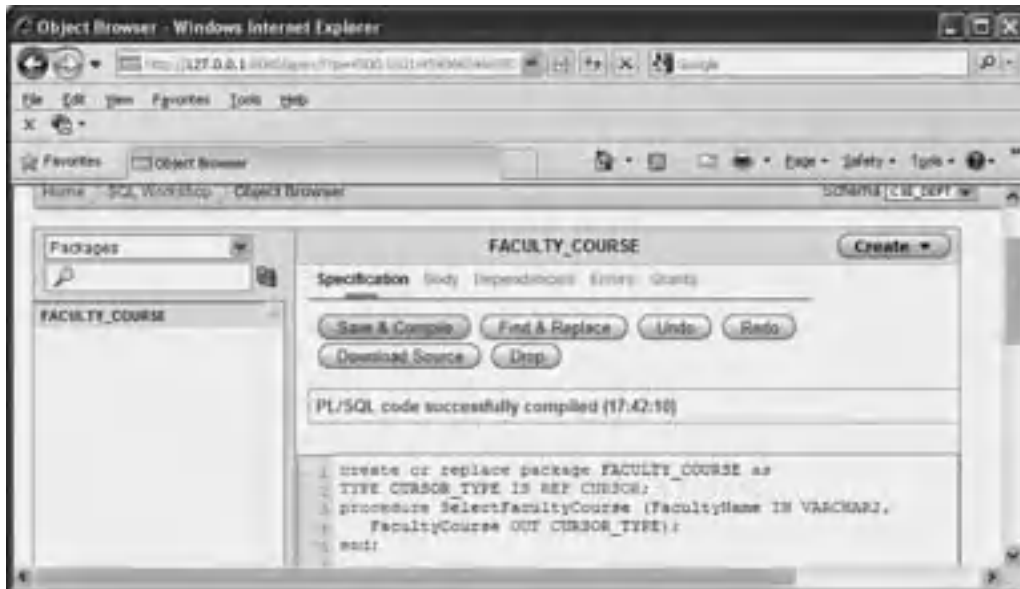


Figure 5.156. The successful compiling page.

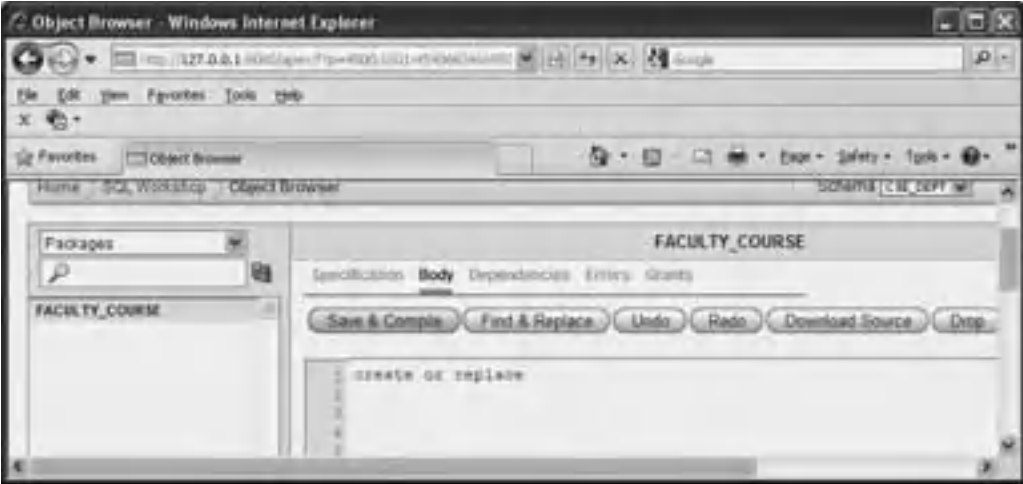


Figure 5.157. The opened Body page of the package.

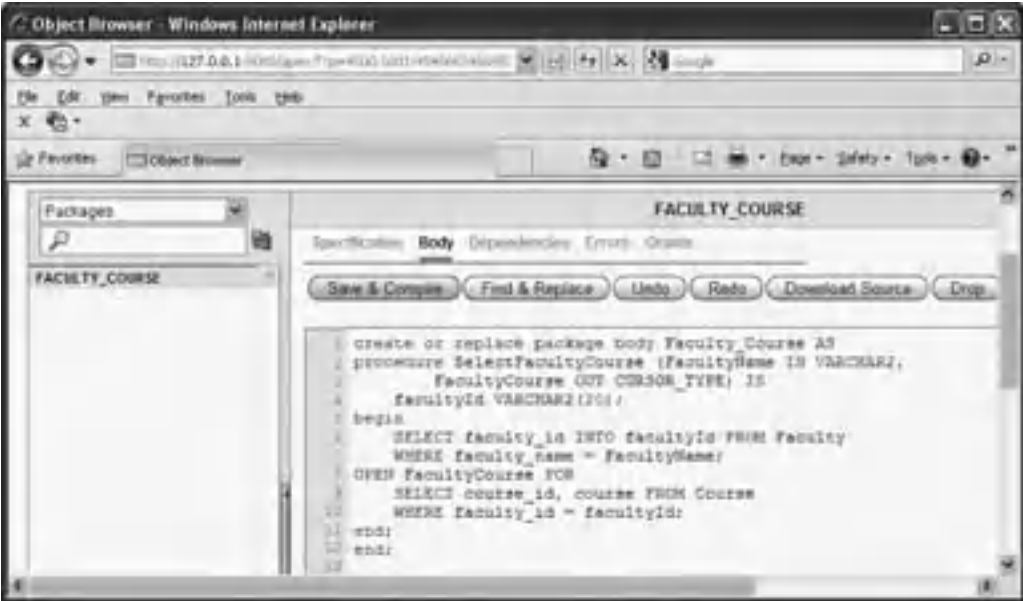


Figure 5.158. The codes for the Body part of the package.

Starting from **BEGIN**, our real SQL statements are placed 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. An **SELECT . . . INTO** statement is utilized to temporarily store the returned `faculty_id` into the intermediate variable `facultyId`.



Figure 5.159. The compiled codes 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 5.19.8.4. 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.

Ok, now let's compile our package by clicking on the Save & Compile button. A successful compiling message in green color

PL/SQL code successfully compiled (18:00:52)

will be displayed if this package is bug-free, which is shown in Figure 5.159.

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.

5.20.9 Query Data Using the Oracle Package For the Course Form

Open the Course form window and double-click on the Select button to open its event procedure and enter the codes that are shown in Figure 5.160 into this event procedure.

Let's take a look at this piece of codes to see how it works.

- A. The package query string is declared first, and this string contains both the package's name (Faculty_Course) and the stored procedure's name (Select FacultyCourse). All query

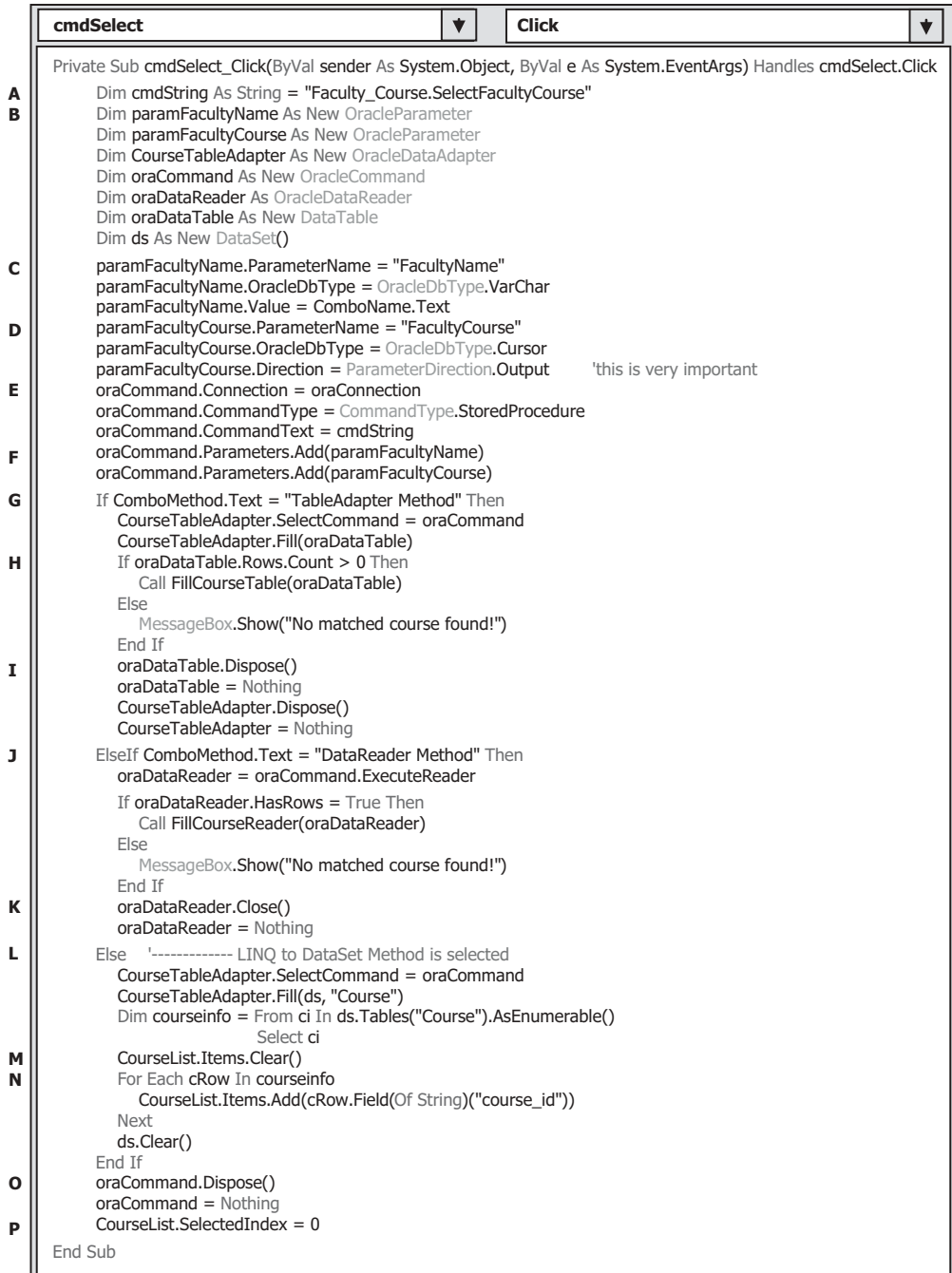


Figure 5.160. The codes for the Select button Click event procedure.

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 the Oracle Database 11g XE. Otherwise, your calling to this package would be failed.

- B.** All data components used to perform this query are declared and created here. First, two Oracle parameter objects are created, `paramFacultyName` and `paramFacultyCourse`, and these two parameter objects will be passed into the calling package, and they work as the input and the output parameter, respectively. Some other components, such as the `TableAdapter`, `Command`, `DataTable` and `Data Reader`, are also created here.
- C.** The first parameter object is initialized first. 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 11g XE. The parameter's value should be equal to the selected name from the faculty name combo box (`ComboName.Text`) in the Course form window in Visual Basic.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 an `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` should be `StoredProcedure` and the `CommandText` should be the query string we declared at the beginning of this event procedure (**A**).
- F.** Two initialized parameter objects are added into the `Command` object, that is, are added into the `Parameters` collection property of the `Command` class.
- G.** If the user selected the `TableAdapter` method, the initialized `Command` object is assigned to the `SelectCommand` property of the `TableAdapter`, and the `Fill()` method is executed to fill the Course table. Exactly, only at this moment, the Oracle package 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 subroutine `FillCourseTable()` is called to fill the queried courses 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 `HasRow` is true, which means that the `DataReader` did read back some rows, the subroutine `FillCourseReader()` is called to fill the `CourseList` box in the Course form window with the read rows. Otherwise, an error message is displayed.
- K.** Finally, another cleaning job is performed to release all components used for this query.
- L.** If the user selected the LINQ to `DataSet` method, the `Fill()` method is executed to fill the Course table in the created `DataSet ds`.
- M.** The `CourseList` box is cleaned up before it can be filled with queried `course_id`.

- N. The For Each loop is executed to pick up each queried column and add them into the CourseList one by one using the Add() method.
- O. The Command object is released after its function is done.
- P. This statement is very important, and it is used to select the first course_id in the CourseList box as the default one as the Course form is opened. More important, this command can work as a trigger event to trigger the CourseList box's SelectedIndexChanged event procedure to display the detailed information related to that default course_id in the seven textbox controls in the Course form window. The codes for this event procedure are our next job.

The user-defined subroutine procedure FillCourseTable() and the Back button Click event procedure have nothing to do with any object used in this project, so no coding modification is needed. The user-defined subroutine FillCourseReader() needs only a small modification, which is to change the data type of the nominal argument CourseReader from OleDbDataReader to OracleDataReader (refer to step A in Fig. 5.161) since now we are using an Oracle data provider. The codes for subroutines FillCourseTable() and FillCourseReader() are similar to the codes we did in Figure 5.91 in Section 5.18.4. For your convenience, we list this piece of codes again, which is shown in Figure 5.161.

Next, we need to take care of the coding for the CourseList_SelectedIndexChanged() event procedure.

The functionality of this event procedure is to display the detailed course information, such as the course id, course title, 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. Six textbox controls in the Course form are used to store and display six pieces of detailed course information.

Now let's begin to do our coding for this event procedure. Open the Course form window and double-click on the CourseList box (any place inside that list box) to open

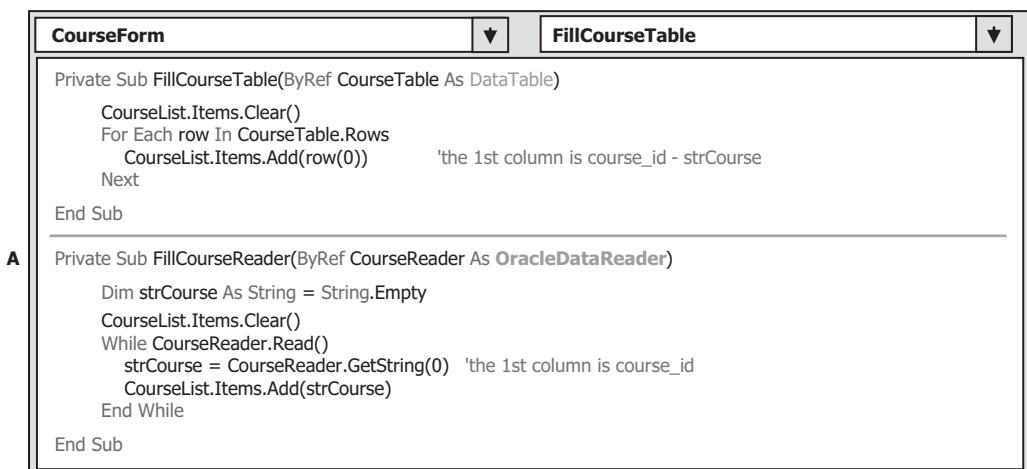


Figure 5.161. The codes for two user-defined subroutine procedures.



Figure 5.162. The codes for the SelectedIndexChanged event procedure.

this event procedure. Enter the codes that are shown in Figure 5.162 into this event procedure.

Let's have a closer look at this piece of codes to 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 event procedure. But in here, 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 TableAdapter, Command object, DataReader, and the DataTable objects. The keyword **Oracle** needs to

be prefixed before all classes related to the Oracle Data Reader, since we are using the Oracle data components to perform this query.

- C. The Command object is initialized here by assigning it with associated property, such as the Connection, Command Type, and CommandText.
- D. The dynamic parameter `courseid` is added into the Parameters collection that 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. If the TableAdapter method is selected by the user, the `SelectCommand` property of the TableAdapter 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 Course table, the user-defined subroutine procedure `FillCourseTextBox()` is called to fill six textbox controls with retrieved six columns. Otherwise, an error message is displayed to indicate that this fill has failed.
- G. Some cleaning job is performed here to release some data objects used for this fill table operation.
- H. If the `DataReader` method is selected by the user, the `ExecuteReader()` method is executed to invoke the `DataReader` to retrieve six data columns.
- I. If the `HasRows` property of the `DataReader` is true, which means that at least one row is collected, the subroutine `FillCourseReader()` is called to fill six textbox controls with retrieved columns. Otherwise, an error message is displayed to indicate that this reading has failed.
- J. Some other cleaning jobs are performed to release all objects used for this query.

The two subroutines: `FillCourseTextBox()` and `MapCourseTable()` have no relationship with any object used in this project; therefore, no coding modification is needed for them. The subroutine `FillCourseReaderTextBox()` needs a small modification, which is to change the data type of nominal argument `CourseReader` from `OleDbDataReader` to `OracleDataReader` since an Oracle data provider is utilized in this project. For the detailed line-by-line explanations of the subroutines `FillCourseTextBox()`, `FillCourseReaderTextBox()`, and `MapCourseTable()`, refer to Figures 5.93 and 5.94 in Sections 5.18.4 and 5.18.5.

Now let's start this project to test the codes we built 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 name from the Faculty Name combo box and click on the **Select** button to list all `course_id` taught by the selected faculty in the CourseList box. Then click on each `course_id` item from the CourseList box, and the detailed course information related to the selected `course_id` is displayed in six textbox controls in this form, which is shown in Figure 5.163.

At this point we finished all coding process for this project. As for the coding process for the Student form, we prefer to leave this job to students as their homework.

But do not forget to copy all faculty image files to the folder in which your Visual Basic executable file is located before you can run this project. In this application, it is the **Debug** folder of the project. In our case, this folder is located at: `C:\Chapter 5\Oracle-SelectRTOBJECT\bin\Debug`.

Figure 5.163. The running status of the Course form.

5.21 CHAPTER SUMMARY

The main topic of this chapter is to develop professional data-driven applications in Visual Basic.NET 2010 environment with three methods. The data query is the main task of this chapter.

The first method is to utilize the Wizards and Tools provided by Visual Studio.NET 2010 and ADO.NET to build simple but powerful data query projects, and the second is to use the runtime object method to build the portable projects. The LINQ to DataSet is introduced as the third method to query data from the different data tables.

Comparably, the first method is simple, and it is easy to be understood and learned by students who are beginner to Visual Basic.NET and databases. This method utilizes a lot of powerful tools and wizards provided by Visual Studio.NET and ADO.NET to simplify the coding process, and most codes are auto-generated by the .NET Framework and Visual Studio.NET 2010 as the user uses these tools and wizards to perform data operations, such as adding new a data source, making data binding, and connecting 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, so it is hard to enable users to have a clear picture about what is really happened behind those tools and wizards. The most codes are generated by the system automatically in the specific locations, so it is not easy to translate and execute those codes in other platforms.

The runtime objects are utilized in the second method. This method 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 codes, it is very easy to translate and execute this kind of projects in other platforms. This method provides a clear view for the users and enables them to have a global and detail picture in how to control the direction of the project with the codes based on the users' idea and feeling.

The shortcoming of this method is that a lot of codes make the project complicated and hard to be accepted by the beginners.

The LINQ to DataSet method is an updated method provided by .NET Framework. With this method, a general query object can be produced and executed in higher efficiency to disregard what kind of language the user is using. In other words, this method is a language- or query-independent method, and this makes this method simple and more efficient compared with another two methods discussed above.

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 two or three different data query methods: TableAdapter method, runtime object method, and LINQ to DataSet method. A line-by-line illustration is 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 2010 and ADO.NET to develop the simple but powerful data-driven applications to perform data query to Microsoft Access, SQL Server 2008, and Oracle 11g XE databases.
- Use the OleDbConnection, SqlConnection, or OracleConnection class to connect to Microsoft Access, SQL Server 2008 Express, and Oracle 11g 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.
- 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 query to Microsoft Access, SQL Server 2008, and Oracle 11g XE databases.
- Use the OleDbConnection, SqlConnection, and OracleConnection class to dynamically connect to Microsoft Access, SQL Server 2008 Express, and Oracle 11g 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 query and process data with three kinds of databases.
- Set properties for the OleDbCommand, SqlCommand, and OracleCommand objects dynamically to construct a desired query string for three kinds of databases.
- Use LINQ to DataSet method to perform data query for three kinds of databases.
- Use the Server Explorer to create, debug, and test stored procedures in Visual Studio.NET environment.

- Use SQL stored procedure to perform the data query from Visual Basic.NET.
- Use Object Browser in Oracle Database 11g XE to create, debug, and test stored procedures and packages.
- Use the Oracle stored procedures and packages to perform the data query from Visual Basic.NET.

In Chapter 6, we will discuss the data inserting technique with three kinds of databases. Both methods are introduced in two parts: Part I: Using the tools and wizards provided by Visual Studio.NET 2010 to develop data inserting query, and Part II: Using the Runtime objects and LINQ to DataSet 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. The Fill() method belongs to the TableAdapter class.
- ___ 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 ExecuteReader() method.
- ___ 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 exactly to fill 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 Basic.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 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 an SELECT statement in the SQL Server database, a @ symbol must be prefixed before the nominal variable.
- ___ 16. The name of the dynamic parameter in an SELECT statement in the SQL Server database may be different with the name of the nominal parameter that is assigned to the Parameters collection of the Command object.

- ___17. To assign a dynamic parameter in an 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: Object Browser page and SQL Command page in Oracle Database 11g XE.
- ___19. Two popular ways to query data from any database are: using Fill() method that belongs to the TableAdapter class, or calling ExecuteReader method that belongs to the Command class.
- ___20. A DataTable can be considered as a collection of DataRowCollection and DataColumnCollection, and the latter contain 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, exactly it executes _____ object with suitable parameters.

- a. DataAdapter
 - b. Connection
 - c. DataReader
 - d. Command
8. To fill a list box or combo box 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 run-time data query to 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 run time 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 are existed in an Oracle Package and they are _____ and _____
- a. Specification, body
 - b. Definition, specifications
 - c. Body, specification
 - d. Specification, execution

III. Exercises

1. Using 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 is located at the folder **DBProjects\Chapter 5** at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).
2. Using LINQ to DataSet method to build the data query for the **Select** button Click event procedure in the Course form in the **AccessSelectRTOObject** project. The project is located at the folder **DBProjects\Chapter 5** at the Wiley ftp site (refer to Fig. 1.2 in Chapter 1).
3. Develop a method by adding some codes into the **cmdLogin_Click()** event procedure of the project **OracleSelectRTOObject** to allow users to try the login process only 3 times. A warning message should be displayed and the project should be exited after 3 times of trying to login but all of them are failed.
4. Using PL-SQL to create a Package in the Object Browser page of Oracle Database 11g XE. The Package contains two stored procedures; one is used for query the **student_id** from the Student table based on the input student name, and the second is to query all **course_id** 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 SQL Server or Oracle data provider for the **SQLSelectRTOObject** or the **OracleSelectRTOObject** project to perform the similar data query jobs for the Faculty form.
6. Using the TableAdapter method (**Fill()**) to perform the data query to Student and StudentCourse tables for the Student form in **SQLSelectRTOObject** project. For your reference, a sample project can be found at the folder **DBProjects\Chapter 5** located at the Wiley ftp site (refer to Fig. 1.2 in 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 11g XE. Either TableAdapter or DataReader method can be used for this query. The desired way is to use an Oracle Package to build this query.

Chapter 6

Data Inserting with Visual Basic.NET

We spent a lot of time in discussion and explanation of 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 is only to insert data into the mapping of the data table in the DataSet, and this insertion 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 is to insert the data into the data tables in the real database.

As you know, ADO.NET provided 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, and 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.

In this chapter, we will provide two parts to show readers how to insert data into the database: inserting data into the database using the Visual Studio.NET design tools and wizards is discussed in the first part, and inserting data to the database using the run-time object method is shown in the second part.

When you finish this chapter, you will:

- Understand the working principle and structure on inserting data to the database using the Visual Studio.NET design tools and wizards
- Understand the procedures in 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 accessing the database

- Understand the working principle and structure on inserting data to the database using the run-time object 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 the Fundamentals of Databases, which was introduced in Chapter 2, and ADO.NET, which was discussed in Chapter 3. 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 the repeatability, we will use the project **SelectWizard** we developed in the last chapter. 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 procedures related to those buttons, will be developed and built 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 discuss how to insert data into the database by first adding new records into the DataSet, and then updating those 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 SQL Server 2008 Express database, `CSE_DEPT.mdf`, which was developed in Chapter 2 and located in the folder `Database\SQLServer` at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). You can try to use any other databases, such as Microsoft Access 2007 or Oracle Database 11g 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 Basic.NET data-driven applications.

6.1 INSERT DATA INTO A DATABASE

Generally, there are many different ways to insert data into the database in Visual Studio.NET. Regularly, three methods are widely utilized:

1. Using the TableAdapter's DBDirect methods, specifically such as the `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 combined with the `ExecuteNonQuery()` method

When using method 1, one can directly access the database and execute commands, such as the `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 choice is to try 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 those data modifications occur in the real database, the `TableAdapter`'s `Update()` method is needed to update those modifications in the database.

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 those data manipulations, which means that both methods need to execute the `Command` object, more precisely, the `ExecuteNonQuery()` method of the `Command` object to finish those data operations again 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, and then we discuss 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 completed coding related to the runtime objects.

6.1.1 Insert New Records into a Database Using the `TableAdapter.Insert` Method

When you use this `TableAdapter` `DBDirect` method to perform data manipulations to a database, the main query must provide enough information in order for the `DBDirect` methods to be created correctly. The so-called main query is the default or original query methods, such as `Fill()` and `GetData()`, when you open for the first time 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 in 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

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 dataset 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.

DataSet to the database based on the associated properties of the TableAdapter, such as the InsertCommand, UpdateCommand, and DeleteCommand.

In this chapter, we only take care of the inserting data, so 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 the TableAdapter.Update Method

To use this method to insert data into the database, one needs to perform the following two 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 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 THE SQL SERVER DATABASE USING A SAMPLE PROJECT INSERTWIZARD

We have provided a very detailed introduction about 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 those staff to develop our data-inserting sample project based on the SelectWizard 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 creat-

ing our new project in this way is that you don't need to redo the data source connection and configuration since those jobs have been performed in the last chapter.

6.2.1 Create New Project Based on the SelectWizard Project

Open the Windows Explorer and create a new folder such as **Chapter 6**, and then browse to our project **SelectWizard Solution**, which was developed in the last chapter and is located at the folder **DBProjects\Chapter 5** at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy this solution with the project to our new folder **Chapter 6**. Change the folder name of the solution and the project from **SelectWizard Solution** to **InsertWizard Solution**, from **SelectWizard project** to **InsertWizard project**, respectively. Also, change the names of the file **SelectWizard.vbproj** to **InsertWizard.vbproj**, and **SelectWizard.vbproj.user** to **InsertWizard.vbproj.user**. Then, double-click on the **InsertWizard.vbproj** to open this project.

On the opened project, perform the following modifications to get our desired project:

- Select one form window, such as the **LogIn Form.vb**, by clicking on it from the **Solution Explorer** window. Then go to the **Project\InsertWizard Properties** menu item to open the project's property window. Change the **Assembly name** from **SelectWizard** to **InsertWizard**, and the **Root namespace** from **SelectWizard** to **InsertWizard**, respectively.
- Click on the **Assembly Information** button to open the **Assembly Information** dialog box, and change the **Title** and the **Product** to **InsertWizard**. Click on the **OK** button to close this dialog box.

Go to **File\Save All** to save those modifications. Now we are ready to develop our graphic user interfaces based on the **SampleWizards Project** we developed in the last chapter.

6.2.2 Application User Interfaces

As you know from the last chapter, five form windows work as the user interfaces for the **SelectWizard** project: **LogIn**, **Selection**, **Faculty**, **Course**, and **Student**. Of all these five form windows, only three of them contain the **Insert** command button, and they are: **Faculty**, **Course**, and **Student**. Therefore, we only need to work on these three forms to perform the data insertion to our database. In fact, we do not need to build any other new form to perform this data insertion operation; instead, we can use the **Insert** button defined in those three forms to do this function. First, let's concentrate on the **Faculty** form to perform the data insertion into our **Faculty** table in the database.

First, let's concentrate on the data validation before the data can be inserted into the database.

6.2.3 Validate Data Before the 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 is to make sure that each datum 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, an NULL must be entered. To make this validation simple, we develop a control collection and add all of those textboxes into this collection. This way, we don't need to check each textbox, but instead, we can use the For Each . . . Next loop to scan the whole collection to find the empty textbox.

6.2.3.1 Visual Basic Collection and .NET Framework Collection Classes

There are two kinds of collection classes available for the Visual Basic.NET applications: one is Visual Basic collection class, and the other is the .NET Framework collection class. One of the most important differences between these two collection classes is the starting value of the index. The index of the Visual Basic collection class is 1-based, which means that the index starts from 1. The index value in the .NET Framework collection class is 0-based, which means that the index starts from 0. The namespace for the Visual Basic collection class is `Microsoft.VisualBasic`, and the namespace for the .NET Framework 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 Microsoft Visual Basic collection object `newVBCollection`, one can use the following declarations:

```
Dim newVBCollection As New Microsoft.VisualBasic.Collection()
```

or

```
Dim newVBCollection As New Collection()
```

The first declaration uses the full name of the collection class, which means that both the class name and the namespace are included. The second declaration uses only the collection class name with the default namespace.

To create a .NET Framework collection object `newNETCollection`, the following declaration can be used:

```
Dim newNETCollection As New System.Collections.Generic.Dictionary(Of String, String)
```

or

```
Dim newNETCollection As New Dictionary(Of String, String)
```

The first declaration uses the full class name and the second one only uses the class name with the default namespace. Both declarations work well for the Visual Basic.NET applications. The newly created collection object contains two arguments, the item key and the item content, and both are in the string format.

Now, let's begin to develop the codes for this data validation using this collection component for our application.

6.2.3.2 Validate Data Using the Generic Collection

First, we need to create the generic collection object for our Faculty form. Since this collection will be used by the different procedures in this form, a form-level or a model-level

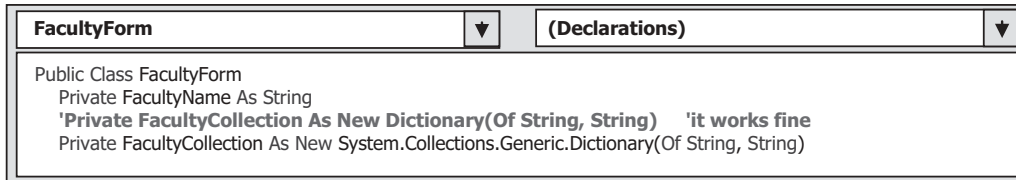


Figure 6.1. The form-level collection object.

object should be created. Open the Code Window of the Faculty form by clicking on the View Code button from the Solution Explorer window, and enter the codes that are shown in Figure 6.1 into the form's declaration section.

The so-called form's declaration section, which is located just under the class header, is used to create all form-level variables or objects. First, we create a form-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 event procedure to validate the newly inserted faculty data. Second, the generic collection object, **FacultyCollection**, is created with two arguments: item key and the item content. The code in bold, 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. The following two points should be noted:

1. First, we need to emphasize the order to perform this validation check. As the project starts, all textboxes are blank. The user needs to enter all pieces of 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 finished 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 by one time.
2. Another point to be noted 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 two user-defined subroutines to perform this adding and removing textboxes from the collection object, respectively.

Open the graphical user interface window of the Faculty form and then double-click on the **Insert** button to open its event procedure. Enter the codes that are shown in Figure 6.2 into this event procedure.

Let's take a closer look at this piece of codes to see how it works.

- A. First, we need to create an instance of the **KeyValuePair** structure, and this structure instance contains two arguments, the **Key** and the **Value**, which are related to a collection component. In **B**, it can be found that both a key and the content of the associated textbox are added into the collection **FacultyCollection** when the user-defined subroutine **CreateFacultyCollection()** is called. We need both the key and the value of a textbox to

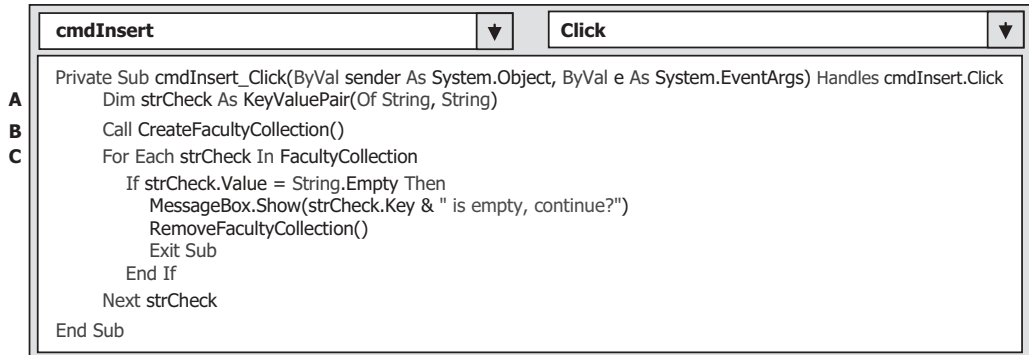


Figure 6.2. The codes for the Insert button event procedure.

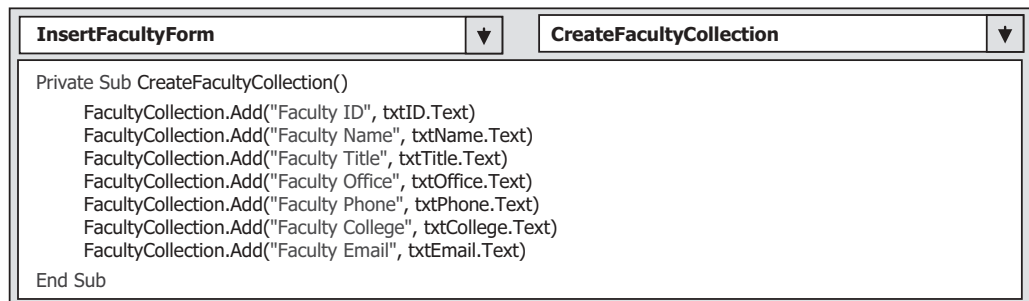


Figure 6.3. The codes for the subroutine CreateFacultyCollection.

validate the data for each textbox, which is to check whether a textbox is empty. The Value of that textbox is used to identify the emptied textbox and the Key of the textbox is used to display the emptied textbox.

- B.** Next, we need to call the subroutine `CreateFacultyCollection()` to add all textboxes into the collection `FacultyCollection`. Refer to Figure 6.3 for the detailed codes for this subroutine later.
- C.** A `For Each` loop is utilized to scan all textboxes to check and identify if any textbox is empty from the `FacultyCollection`. If any textbox is empty by checking its Value, that textbox will be identified by its Key, and a message box is used to ask users to fill some information into it. Then, the subroutine `RemoveFacultyCollection()` is called to remove all textboxes that have been added into the collection in the subroutine `CreateFacultyCollection()` since the collection only allows those textboxes to be added only once. The project will exit if this situation happens. The detailed codes for the user-defined subroutine `RemoveFacultyCollection()` are shown in Figure 6.4.

Now let's take care of the codes for the subroutine `CreateFacultyCollection()`, which are shown in Figure 6.3.

The codes are very simple and straightforward. Each textbox is added into the collection by using the `Add()` method with two parameters: the first one is the so-called Key parameter represented in a string format, and the second is the content of each textbox, which is considered as the Value parameter. In this way, each textbox can be identified

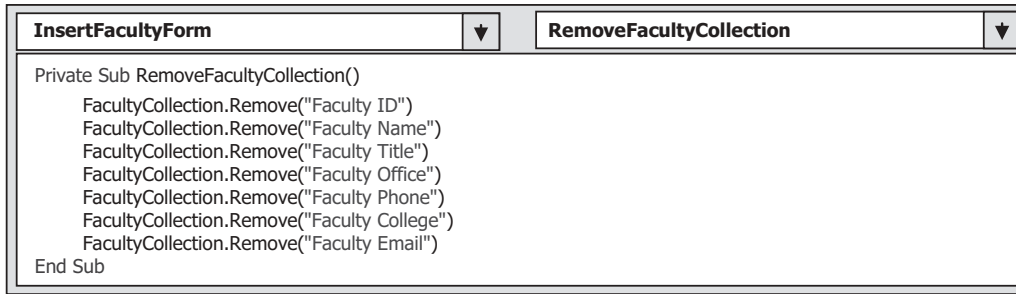


Figure 6.4. The codes for the subroutine RemoveFacultyCollection.

by its Key. Of course, each textbox can also be identified by the index, but remember that the index starts from 0, not 1 since it is a .NET Framework collection instead of a Visual Basic collection.

To remove all textboxes from the collection, another user-defined subroutine procedure RemoveFacultyCollection() should be called, and the codes for this subroutine are shown in Figure 6.4.

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.

At this point, we have completed the coding process for the data validation. Next, we need to handle some initialization coding jobs for the data insertion.

6.2.4 Initialization Coding for the Data Insertion

In this section, we need to handle the coding for the following event procedure:

- Coding for the Form_Load event procedure to add two more insertion methods, TableAdapter.Insert() and TableAdapter.Update(), into the combo box comboMethod to display two data insertion methods.

This coding process is for the combo box comboMethod. As the project runs and the Faculty form window is shown up, two more 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. Open the Faculty form and its Form_Load event procedure, and add the codes that are in bold and shown in Figure 6.5 into this event procedure.

The codes are straightforward and easy to be understood. Two methods are added into the combo box by using the Add() method, and the first method is selected as the default one by setting up the SelectedIndex property to zero.

Now we need to take care of the coding process for the data insertion. Because we are using the design tools to perform this job, first, we need to configure the TableAdapter and build the insert query using the TableAdapter Query Configuration Wizard.

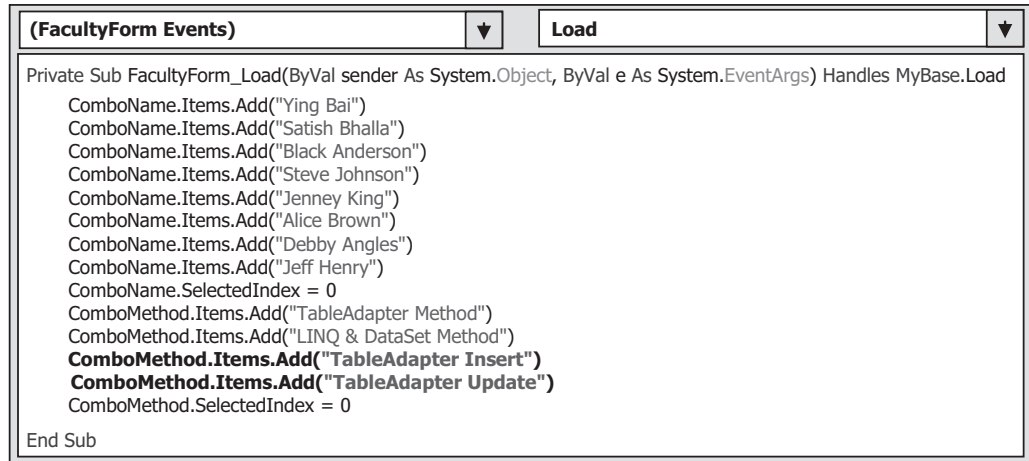


Figure 6.5. The modified codes for the Form_Load event procedure.

6.2.5 Build the 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 concentrate on the first method.

6.2.5.1 Configure the TableAdapter and Build the Data Inserting 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. Perform the following operations to build this data insertion query:

1. Open the Data Source window by going to the **Data>Show Data Sources** menu item.
2. On the opened window, click on the **Edit the DataSet with Designer** button that is located at the second left on the toolbar in the Data Source window to open this Designer.
3. 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.
4. Keep the default selection **Use SQL statements** unchanged and click on the **Next** button to go to the next wizard.
5. Select and check the **INSERT** item from this wizard since we need to perform an inserting new records query, and then click on the **Next** button again to continue.
6. Click on the **Query Builder** button since we want to build our insert query. The opened Query Builder wizard is shown in Figure 6.6.
7. The default Insert query statement is matched to our requirement since we want to add a new faculty record that contains all 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 wizard.
8. Click on the **Next** button to confirm this query and continue to the next step.

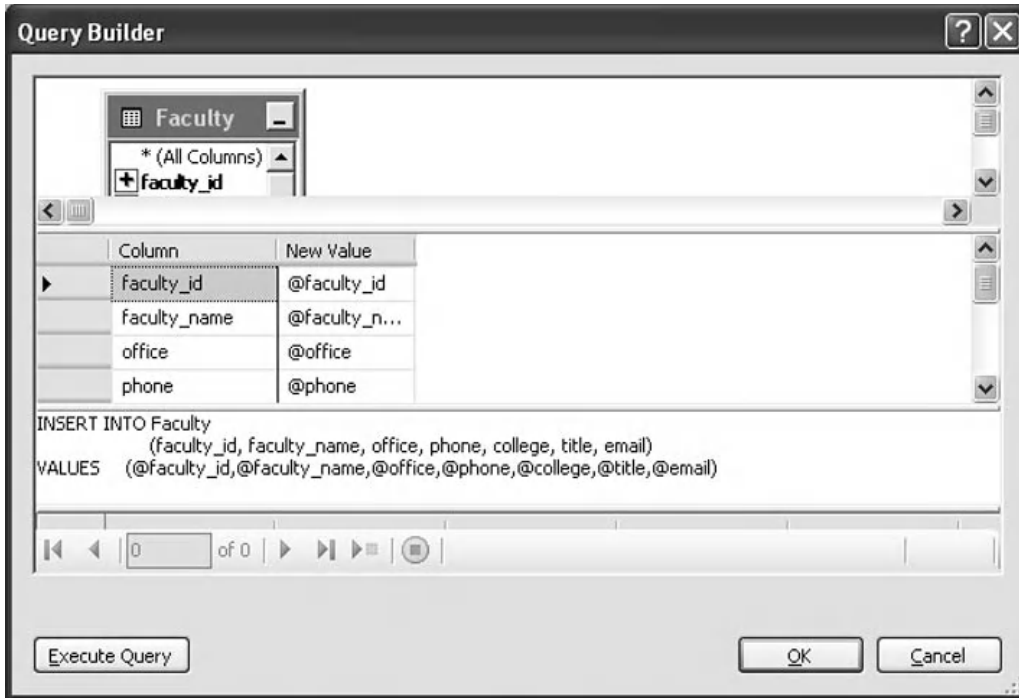


Figure 6.6. The opened Query Builder wizard.

9. Modify the query function name from the default one to the InsertFaculty and click on the Next button to go to the last wizard.
10. 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.

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 insertion query. We need to develop the codes for the first method—using the TableAdapter DBDirect method, TableAdapter.Insert().

6.2.6 Develop Codes to Insert Data Using the TableAdapter.Insert Method

Open the graphical user interface of the Faculty Form by clicking on the View Designer button from the Solution Explorer window, and then double-click the Insert button to open its Click event procedure. Then add the codes that are shown in Figure 6.7 into this event procedure.

Recall that we have created some codes for this event procedure in Section 6.2.3.2 to perform the data validation, so the old codes are highlighted with a gray background.

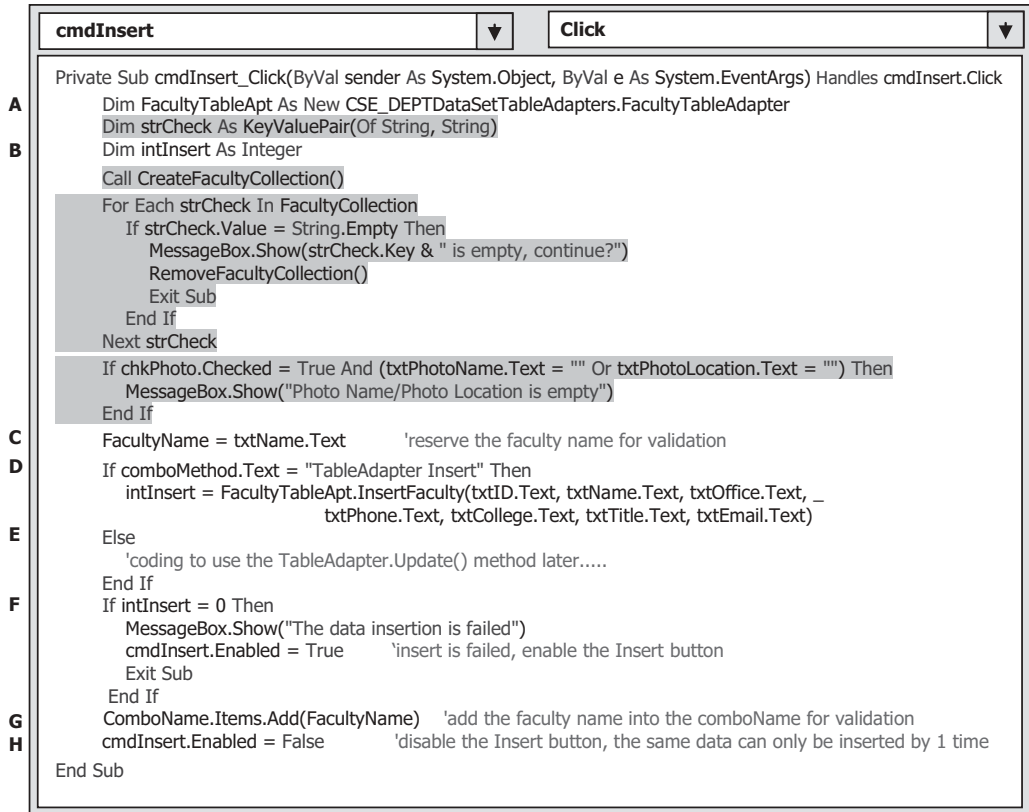


Figure 6.7. The modified codes for the Insert button event procedure.

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

- A.** First, we need to create a TableAdapter object for the FacultyTableAdapter class since we need this object to perform inserting data directly into the database. As we know, all TableAdapters in this application are located at the namespace CSE_DEPTDataSetTableAdapters, so this namespace must be prefixed before the desired TableAdapter class.
- B.** 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 has been added or affected to the database; in other words, this insertion has failed.
- C.** The form-level variable FacultyName is used to temporarily hold the faculty name entered by the user from the textbox txtName since we need this name to validate this insertion later.
- D.** 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 job. Seven pieces

of new information, which is about the newly inserted faculty and entered by the user into seven textboxes, will be inserted to the Faculty table in the database.

- E.** 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.
- F.** If this data insertion is successful, the returned integer will reflect the number of records that have been inserted into the database correctly. As we mentioned in **B**, a returned value of zero indicates that this insertion is failed. A message box will be displayed with a warning message, and the program will be exited. One point we need to emphasize is that when performing a data insertion, the same data can only be inserted into the database by one time, and the database does not allow multiple insertions of the same data item. To avoid multiple insertions, in this application (generally in most popular applications), we will disable the Insert button after one record is inserted successfully (refer to step **H** below). If the insertion has failed, we need to recover or reenale 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 only once. In other words, no duplicated record can be added or can exist in the database. Each record in the database must be unique. The popular way to avoid this situation from happening is to disable the Insert button after one insertion is done.

- G.** As we mentioned before, when we perform the data validation to validate this data insertion, an `SELECT` statement will be executed to retrieve the newly inserted record from the database. The faculty name will work as the dynamic parameter for the `WHERE` clause in that `SELECT` statement, so we need to add this faculty name into the combo box `comboBoxName` for the validation to be performed later.
- H.** As we mentioned in **E**, the database does not allow the multiple insertions of the same data item into the database. So after the data insertion is successful, we need to disable the Insert button to protect it from being clicked again.

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 has been successfully completed. A question arises: When and how can this button be enabled again to allow us to insert new, different records if we want to do that later? The solution to this question is to develop another event procedure to handle this issue. Try to think about it, the time when we want to insert a new, different data item into the database; first, we must enter each piece of new information into each associated textbox, such as `txtID`, `txtName`, `txtOffice`, `txtPhone`, `txtTitle`, `txtCollege`, and `txtEmail`. In other words, any time as long as the content of a textbox is changed, which means that a new, different record will be inserted, we should enable the Insert button at that moment to allow users to perform this new insertion. Visual Basic.NET did provide an event called `TextChanged` and an associated event procedure for the textbox control. So we need to use this event procedure to enable the Insert button as long as a

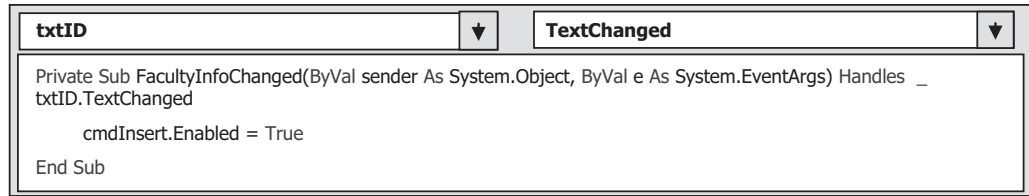


Figure 6.8. The codes for the txtID TextChanged event procedure.

TextChanged event occurs. Another question arises: with which textbox's **TextChanged** event occurring, we should trigger the associated event procedure to enable the **Insert** button to allow users to insert a new record? Is any textbox's **TextChanged** event? To answer these questions, we need to review the data issue in the database. As you know, in our sampling database **CSE_DEPT** (i.e., 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 questions is: only the content of the textbox that stores the primary key—in our case, it is the **txtID** that equals to the **faculty_id**—is changed; it means that a new record will be inserted, and as this happened, that textbox's **TextChanged** event procedure should be triggered to enable the **Insert** button.

To open the **TextChanged** event procedure for the textbox **txtID**, open the graphical user interface of the **Faculty** form window by clicking on the **View Designer** button from the **Solution Explorer** window, and then double-click on the **Faculty ID** textbox to open its **TextChanged** event procedure. Change the event procedure's name from the **txtID_TextChanged** to **FacultyInfoChanged** and enter the codes shown in Figure 6.8 into this event procedure.

The codes for this event procedure are 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 codes for the first data insertion method, let's continue to do our coding process for the second method.

6.2.7 Develop Codes to Insert Data Using the **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. First, 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. Then 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 the above two steps to insert data using this method.

Open the graphical user interface of the Faculty form window and double-click on the **Insert** button to open its event procedure. We have already developed most codes for this procedure 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 6.7), and enter the codes that are shown in Figure 6.9 into this block.

In order to distinguish between the newly added codes and the old codes that have been developed before, all old codes are highlighted with a gray background.

Let's take a closer look at this piece of newly inserted codes 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, as we need to create a row in the Faculty data table, the `FacultyRow` is selected as the `DataRow` class.

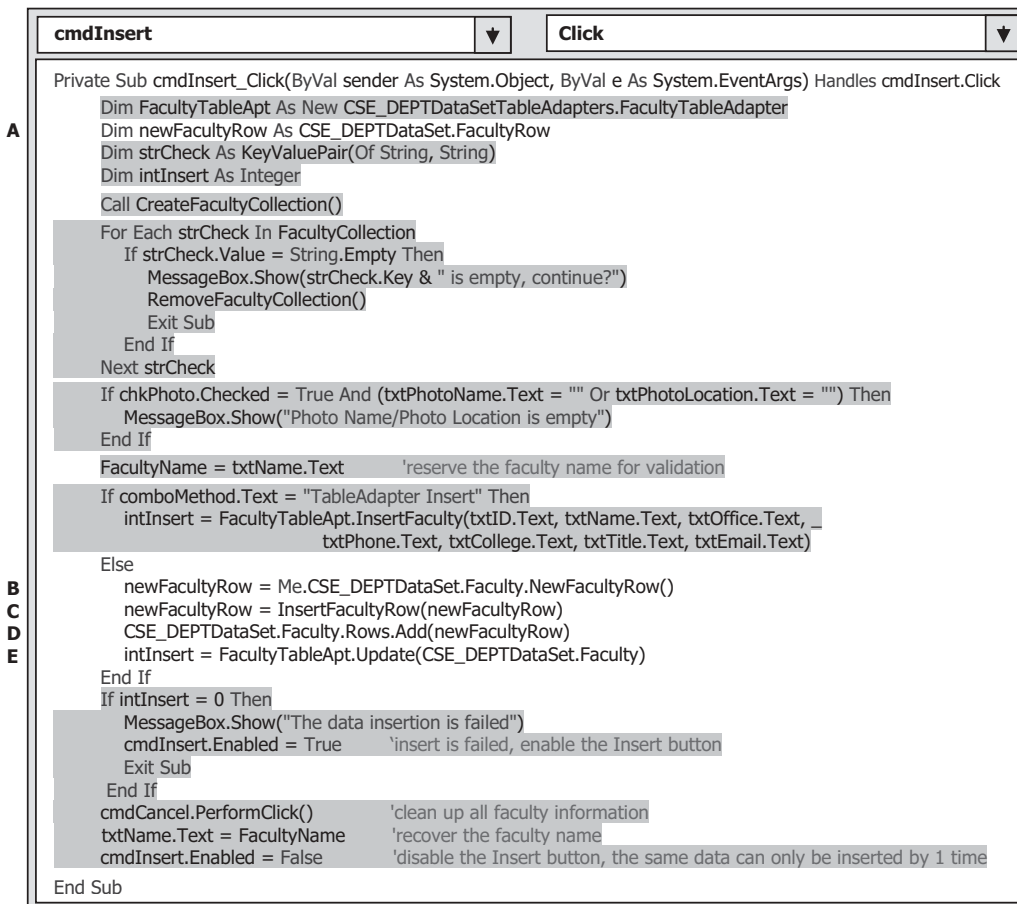


Figure 6.9. The codes for the second data insertion method.

- B.** Next, we need to create a new object of the `NewFacultyRow` class.
- C.** A user-defined function `InsertFacultyRow()` is called to add all pieces of information about the newly inserted faculty, which is stored in seven textboxes, into this newly created `DataRow` object. The detailed codes and the function of this user-defined function will be explained below. This function returns a completed `DataRow` in which all pieces of information about the new record has been added.
- D.** The completed `DataRow` is added into the `Faculty` table in our `DataSet` object. One point to be noted is that adding a new record into the data table in the `DataSet` has 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 method `Update()` is executed to add this new record 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 `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 inserted 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 codes for the user-defined function `InsertFacultyRow()`. Open the code window and enter the codes that are shown in Figure 6.10 into this function.

Let's have a closer look at this piece of codes to see how this function works.

- A.** In Visual Basic.NET, unlike C/C++ or Java, the subroutines and functions are different. A procedure that returns data is called a function, but a procedure that does not return any data is called a subroutine. The function `InsertFacultyRow()` needs to return a completed `DataRow` object, and the returned data type is indicated at the end of the function header after the keyword `As`. The argument is also a `DataRow` object, but it is a newly 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 procedure.

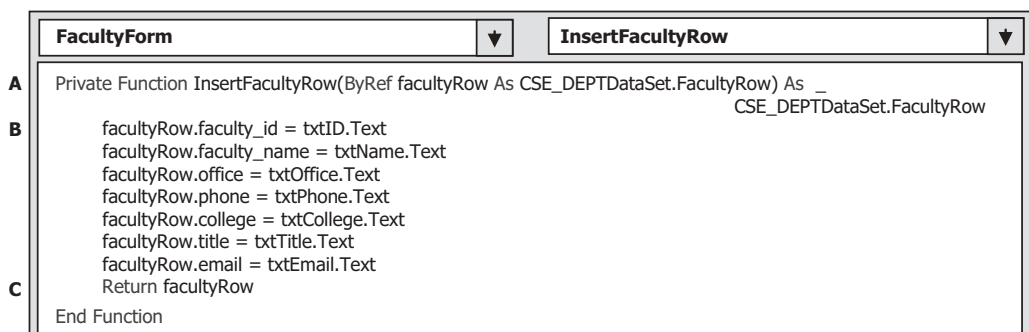


Figure 6.10. The codes for the user-defined function `InsertFacultyRow`.

- B.** Seven pieces of new information stored in the associated textboxes are added into this new DataRow object, that is, added into a new row of the faculty table in the DataSet.
- C.** Finally, this completed DataRow object is returned to the calling procedure. Another advantage of using this reference mode 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.

At this point, we have completed the coding process for our data insertion by using two methods. Before we can run the project to test the function of the codes we developed, we need to find a way to confirm this data insertion. To confirm this data insertion, we can use the **Select** button Click event procedure to do this job. However, we need to do some modifications to the codes inside the **Select** button Click event procedure, since we may insert a new photo for the inserted faculty member.

Open the **Select** button Click event procedure and modify the codes just below the function FindName(), as shown in Figure 6.11. The modified codes have been highlighted in bold.

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

- A.** If no matched faculty image can be found, there are two possibilities: (1) no matched faculty image file or no inserted faculty image exist, or (2) the image is a new faculty image to be inserted, and its name is located at the Faculty Image textbox. If the first situation happened, a default faculty image file is assigned and used.
- B.** If the second case occurred, the image file name stored in the Faculty Image textbox, txtImage, is assigned and used.

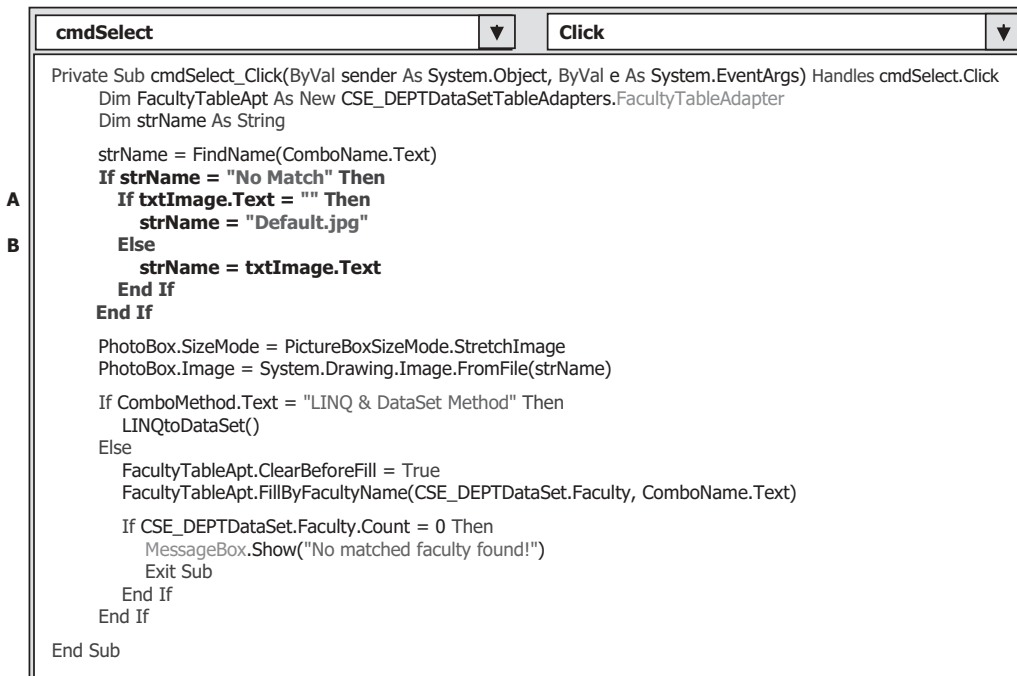


Figure 6.11. The modified codes for the Select button Click event procedure.

Now let's test our codes by running our project. You have two ways to test the project. One way 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. The second way, which is more flexible, is to directly starting from the Faculty form.

To run the project in the second way, one must confirm that the **Startup** form in the Project Properties window is the **Faculty** form window. To do that, go to the Project\InsertWizard Properties menu item to open the Project property window. Keep the default tab **Application** selected, and make sure that the content of the **Startup** form box is the **FacultyForm**. Of course, if you want to run the project in the first way, you need to confirm that the Startup form is the **LogInForm**.

Now you can run the project in either way. We prefer to run it in the first way. Make sure that the Startup form is **LogInForm**, 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 to open the Faculty form.

First, we want to test the first method, **TableAdapter.Insert()**, to add a new faculty record into the database, so select this method from the combo box. Enter seven pieces of new information for this newly inserted faculty member into the associated textbox, as shown in Figure 6.12.

Faculty ID: A56789
 Name: Williams Tom
 Title: Associate Professor
 Office: MTC-222
 Phone: 750-330-1660
 College: University of Miami
 Email: wtom@college.edu

The screenshot shows a Windows application window titled "CSE DEPT Faculty Form". It contains two main sections. The "Faculty Name_Query Method" section has two dropdown menus: "Faculty Name" with "Ying Bai" selected, and "Query Method" with "TableAdapter Insert" selected. The "Faculty Information" section contains seven text input fields with the following values: "Faculty ID" (A56789), "Name" (Williams Tom), "Title" (Associate Professor), "Office" (MTC-222), "Phone" (750-330-1660), "College" (University of Miami), and "Email" (wtom@college.edu). At the bottom of the window are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 6.12. The running status of inserting a new faculty.

The screenshot shows a Windows application window titled "CSE DEPT Faculty Form". It contains several input fields and buttons. On the left, there is a "Faculty Image" section with a small rectangular box above a larger square placeholder image of a person. Below the image placeholder are two buttons: "Select" and "Insert". On the right, there is a "Faculty Name_Query Method" section with two dropdown menus: "Faculty Name" (showing "Williams Tom") and "Query Method" (showing "TableAdapter Insert"). Below this is a "Faculty Information" section with seven textboxes: "Faculty ID" (A56789), "Name" (Williams Tom), "Title" (Associate Professor), "Office" (MTC-222), "Phone" (750-330-1660), "College" (University of Miami), and "Email" (wtom@college.edu). At the bottom of the form are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 6.13. The confirmation result of the newly inserted faculty member.

After all 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, there will be no message box with a warning message to be displayed, and the **Insert** button is disabled to avoid the same data to be added into the database more than one time.

Now let's confirm this data insertion by retrieving this inserted faculty from our database using the **Select** button Click event procedure. Go to the Faculty Name combo box and scroll down this box, and you can find that the newly inserted faculty member Williams Tom has been added into this box. Select this member and click on the **Select** button. All inserted seven pieces of information related to this faculty member Williams Tom are retrieved and displayed in seven textboxes, as shown in Figure 6.13. A default faculty image is used since we did not enter any image name into the Faculty Image textbox.

Click on the **Back** and the **Exit** buttons to terminate the project.

You can try to use the second method, `TableAdapter.Update()`, to insert a new faculty record into our sample database.

Next, we want to discuss how to insert a new record using the stored procedure.

6.2.8 Insert Data into the Database Using the 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 will use the Course Form window to discuss 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 modify the codes for the `Form_Load` event procedure of the Course Form

and develop the codes for the Insert button's Click event procedure. The code modifications and code developments include:

1. Add one more item **Stored Procedure Insert** into the combo box **ComboMethod** in the **Form_Load** event procedure to allow users to select this method to perform the data insertion using this method.
2. Add one If block in the Insert button Click event procedure to allow users to select the method, **Stored Procedure Insert**, to perform the data insertion.
3. Add the associated codes in the If block in the Insert button Click event procedure to call the built stored procedure to perform the data insertion.

Let's first to create a stored procedure under the Course table using the TableAdapter Query Configuration Wizard.

6.2.8.1 Create the Stored Procedure Using the TableAdapter Query Configuration Wizard

Perform the following operations to create this stored procedure:

1. Open the **Data Source** window by clicking on the **DataShow 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 popup menu to open this wizard.
2. Right-click on the last item from the Course table and select the **AddQuery** item from the pop-up menu to open the TableAdapter Query Configuration Wizard.
3. Check the **Create new stored procedure** radio button since we want to create a new stored procedure to do the data insertion. Then click on the **Next** button to go to the next wizard.
4. Check the **INSERT** radio button and click on the **Next** button to continue.
5. Click on the **Query Builder** button on the opened wizard since we need to build a new query. The Query Builder wizard is opened and shown in Figure 6.14.

Make sure that the order of the inserted columns in the **INSERT INTO** statement is identical with the order shown in Figure 6.14, since this order is very important and it must be identical with the order of the columns in the query stored procedure that will be called from the Insert button's Click event procedure later. Click on the **OK** button to go to the next wizard to confirm our built query function, which is shown in Figure 6.15.

Since we only need to insert a record into the database, highlight the second **SELECT** statement and delete it by pressing the **Delete** key from your keyboard. Click on the **Next** button again, and enter **InsertCourseSP** as the name of this query's stored procedure into the name box. Click on the **Next** and the **Finish** buttons to close this process.

6.2.8.2 Modify the Codes to Perform the Data Insertion Using the Stored Procedure

The first modification is to add one more item **Stored Procedure Insert** into the combo box **ComboMethod** in the **Form_Load** event procedure to allow users to select it to perform the data insertion. Open the **Form_Load** event procedure by first selecting the item (**CourseForm Events**) from the **Class Name** combo box, and then selecting the item

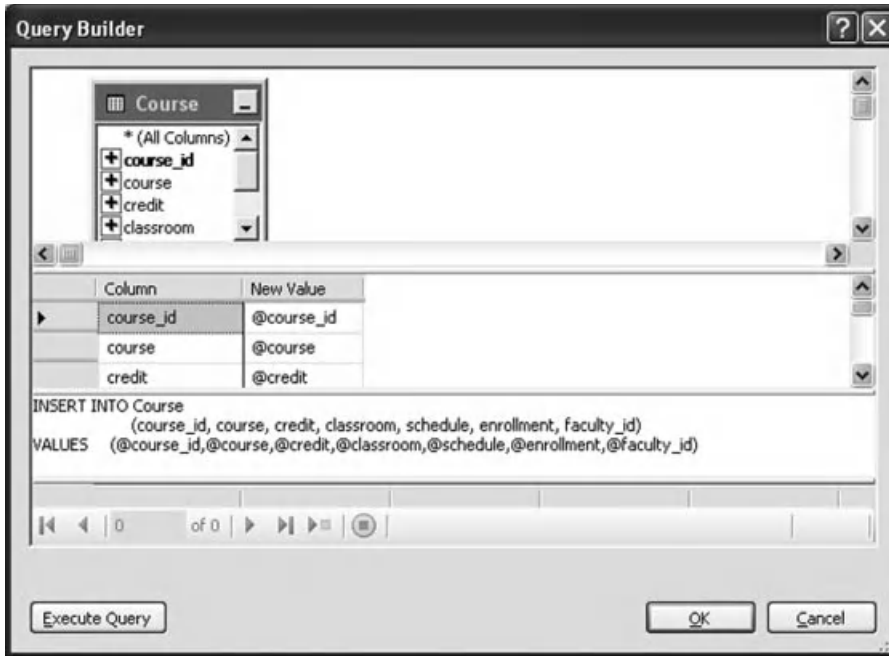


Figure 6.14. The opened Query Builder wizard.

Load from the Method Name combo box in the code window of the Course Form window. Add one more line of the code, which is highlighted in bold and shown in Figure 6.16, into this event procedure. The codes we developed before are highlighted with a gray background.

Now, let's perform the second and the third code developments and modifications. Both code developments and modifications are performed inside the **Insert** button's Click event procedure. Open the **Insert** button's Click event procedure and enter the codes that are shown in Figure 6.17 into this event procedure.

To make this piece of codes simple, we did not develop any code to perform the checking or validation of input data before the data insertion. In fact, this checking and validation is necessary in most actual applications. We leave this coding process as a homework to the readers, and one can refer to Section 6.2.3 to build this data validation function.

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

- A.** Some data action components, such as TableAdapters and local variables used for this procedure, are declared and created first. The string variable **strFacultyID** is used to store the queried **faculty_id** from the execution of the first query, and the integer variable **intInsert** is used to hold and check the running result of the data insertion operation.
- B.** Since there is no **faculty_name** column available in the Course table, and the only available column in that table is the **faculty_id**, therefore, two queries are needed for inserting a new course record into the Course table: (1) query to the Faculty table to get the matched **faculty_id** based on the faculty name selected by the user, and (2) query to the Course

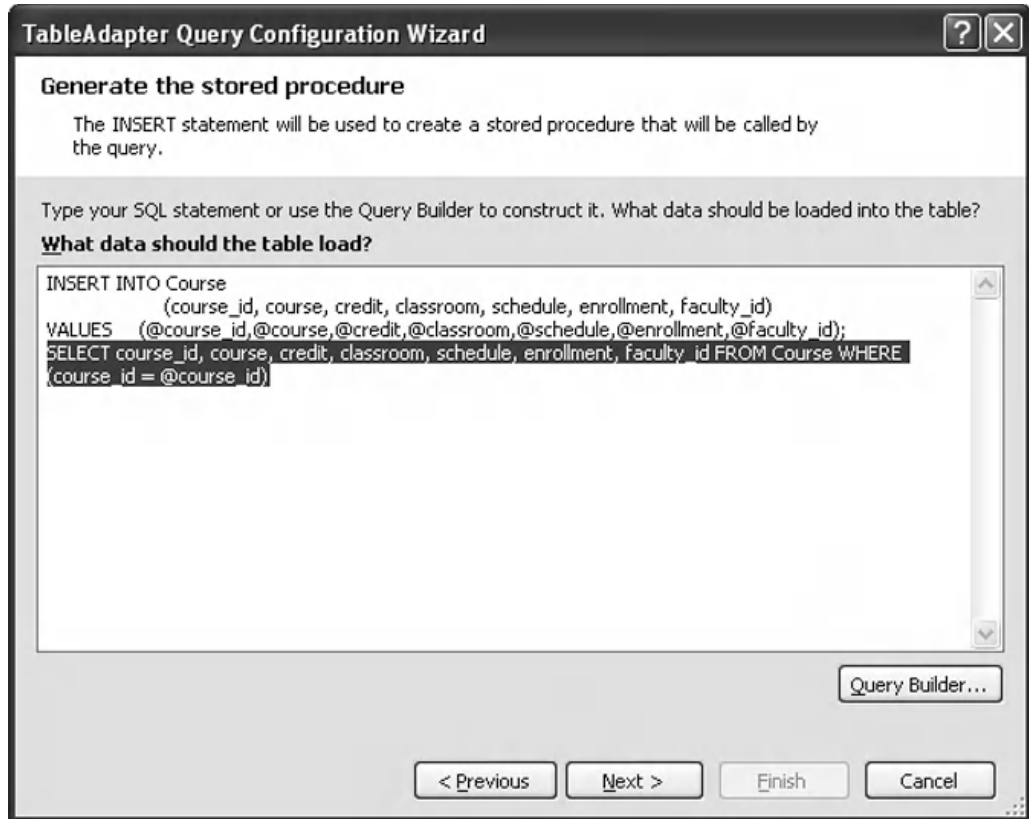


Figure 6.15. The confirmation wizard.

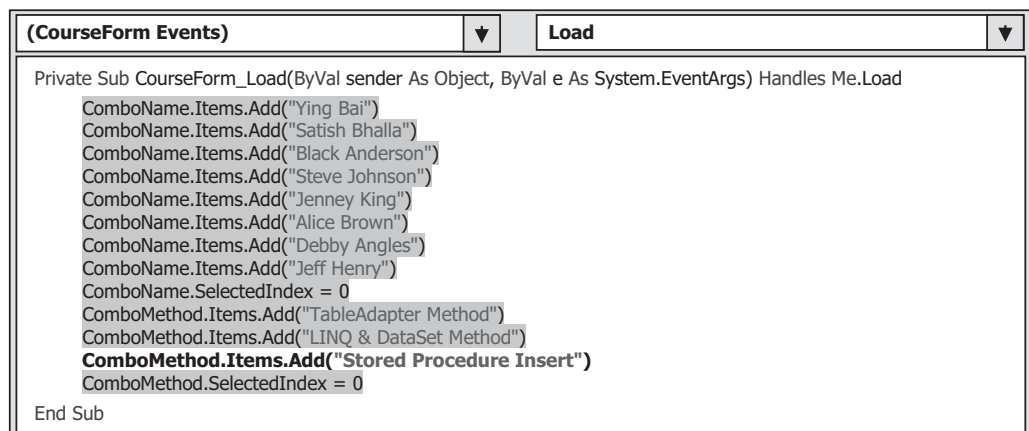


Figure 6.16. The code modification to the Form_Load event procedure.

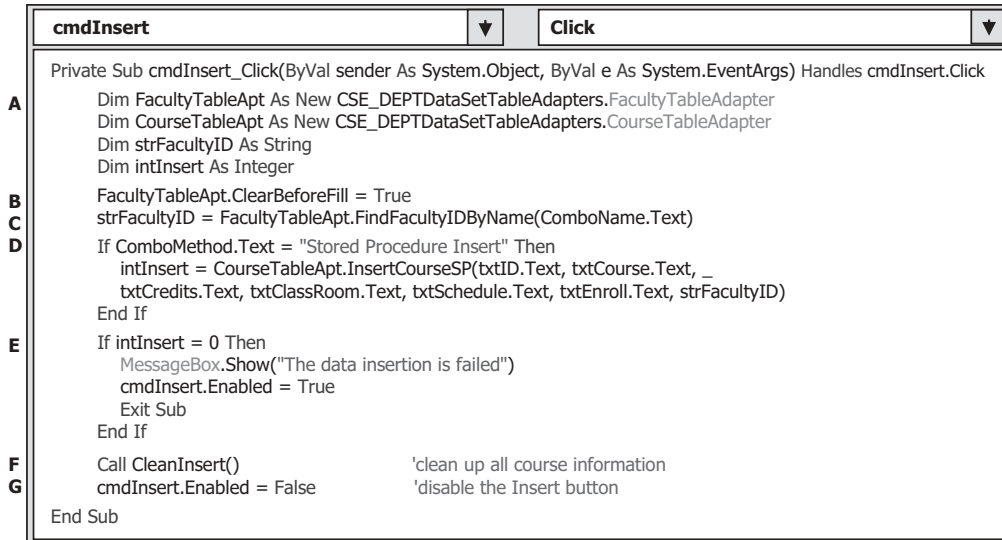


Figure 6.17. The code developments for the Insert button's Click event procedure.

table to insert a new course based on the `faculty_id` obtained from the first query. Prior to performing the first query, the Faculty TableAdapter is cleaned up first.

- C. The first query is executed by calling the query function `FindFacultyIDByName()` we built in Section 5.14.1 in Chapter 5 to get the matched `faculty_id` based on the faculty name selected by the user from the `ComboName` box.
- D. If the user selected the **Stored Procedure Insert** method to perform this course data insertion, the stored procedure `InsertCourseSP()` we built in the last section is called to perform this new course record insertion operation. One point to be noted is that the order of the inserting columns in this calling stored procedure must be identical with that order in the stored procedure we built in the last section. Otherwise, you may encounter a mismatching error during the project runs.
- E. By checking the returned value that stored in the local integer variable `intInsert`, we can determine whether this data insertion is successful or not. If this stored procedure returns a zero, which means that no any record has been inserted into the Course table, a warning message is displayed to indicate this situation, and the project is exited.
- F. Otherwise, the data insertion is successful. A user-defined subroutine procedure `CleanInsert()` is executed to clean up all six pieces of new course information from six textboxes in this form window.
- G. To avoid multiple insertions for the same record, the **Insert** button is disabled after this successful data insertion.

The detailed codes for the user-defined subroutine `CleanInsert()` is shown in Figure 6.18. The codes are used to clean up all six textboxes in the Course Form window.

From the codes shown in Figure 6.17, it can be found that 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 exactly a function or a collection of functions to perform some special functionality or

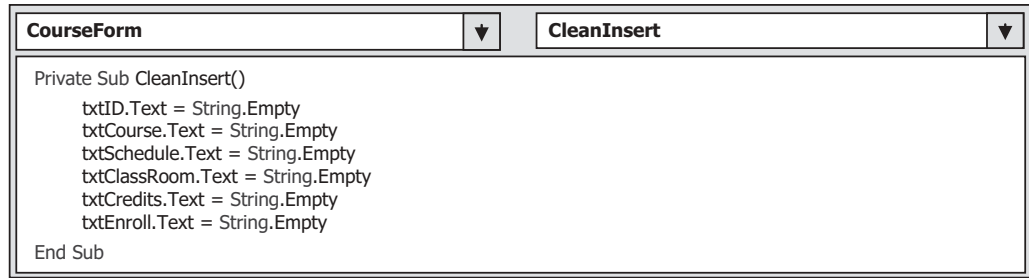


Figure 6.18. The codes for the subroutine CleanInsert().

functionalities. One point we need to note is that by using the TableAdapter Query Configuration Wizard, we cannot create a stored procedure that can perform multiple data actions to the multiple different data tables since each TableAdapter can only access the associated data table. However, by using the runtime object method to insert data into the database, which we will discuss in Part II, one stored procedure can access multiple different data tables and fulfill multiple different data manipulation operations.

At this point, we have finished developing our sample project to insert data into the SQL Server database. Now we can run our project to test inserting new course records using the stored procedure method. Click on the **Start Debugging** button to run the project. Complete the login process and select the **Course Information** from the Selection Form to open the Course Form window.

Keep the default faculty member Ying Bai in the Faculty Name combo box unchanged, and select the **Stored Procedure Insert** method from the Query Method combo box. Then enter the following six pieces of new course information into the associated textbox:

- Course ID: CSE-566
- Course: Introduction to Fuzzy Logic
- Schedule: TH: 1:30–2:45 PM
- Classroom: TC-309
- Credits: 3
- Enrollment: 20

Click on the **Insert** button to try to insert this new course into the Course table in our sample database. Immediately, you can find that all six textboxes are cleaned up after this data insertion and the Insert button is disabled.

To confirm and validate this data insertion, just keep the default faculty member Ying Bai selected from the Faculty Name box, and click on the **Select** button to try to retrieve all courses taught by this faculty. You can find that the newly inserted course CSE-566 has been added into the CourseList box. To get the details of that course, just click on that **course_id** (CSE-566) from the CourseList box, and you can find the detailed information for that course is displayed in six textboxes, as shown in Figure 6.19.

Click on the **Back** and **Exit** buttons to terminate our project.

A completed project InsertWizard can be found in a folder DBProjects\Chapter 6 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Figure 6.19. The running result of calling the stored procedure.

Next, we will discuss how to insert data into the Oracle database using the Visual Studio.NET design tools and wizards.

6.3 INSERT DATA INTO THE ORACLE DATABASE USING A SAMPLE PROJECT INSERTWIZARDORACLE

Because of the similarity between the SQL Server database and the Oracle database, all of codes we developed in the last section can be used to access the Oracle database to perform data insertion functionality. The only difference between these two databases is the connection string when the Oracle database is connected to the Visual Basic.NET applications. In order to save the space and the time, we will not duplicate those codes in this section. Refer to Section 5.16.3 in Chapter 5 for more detailed information in how to add and connect the Oracle database with your Visual Basic.NET applications using the Design Tools and Wizards. Refer to Appendix C to get a clear picture in how to use the sample Oracle 11g XE database CSE_DEPT. 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 simple way to do this job is to copy a project `SelectWizardOracle` we built in Chapter 5 and paste it into our new folder Chapter 6. Refer to Section 6.2.1 to change this project's name from `SelectWizardOracle` to `InsertWizardOracle`.

You need to consider and perform the following modifications to make this project works:

1. Modify the codes inside the `Select` button Click event procedure in the Faculty Form window. Refer to steps **A** and **B** in Figure 6.11 in Section 6.2.7 to complete this modification.

2. Refer to Section 6.2.8.2 to complete the following developments:
 - Add one more item **Stored Procedure Insert** into the combo box **ComboMethod** in the **Form_Load** event procedure in the **Course Form** window.
 - Develop the codes for the **Insert** button **Click** event procedure in the **Course Form** window.
 - Build the codes for the user-defined subroutine procedure **CleanInsert()**.
 - Add **cmdInsert.Enabled=True** into the **Course ID TextChanged()** event procedure.
3. Build the Oracle 11g XE stored procedure **InsertCourseSP()** for the **Course Form**. One point is that after you created this Oracle stored procedure, you need to add it into our project **InsertWizardOracle** using the **DataSet Configuration Wizard**. Refer to Appendix E for this addition operation.

A complete data insertion project named **OracleInsertWizard** can be found in a folder **DBProjects\Chapter 6** located at the Wiley ftp site (refer to Figure 1.2 in 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 job in Visual Basic.NET environment. Compared with the method we discussed in Part I, in which Visual Studio.NET design tools and wizards are utilized to insert data into the database, the runtime objects method provides more sophisticated techniques to do this job efficiently and conveniently even if a more complicated coding job is needed. Relatively speaking, the methods we discussed in the first part are easy to learn and code, but some limitations exist for 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 times that you can modify that query function is either before the project runs or after the project runs. In other words, you cannot modify that query function during the project runs.

To overcome those shortcomings, we will discuss how to insert data using the runtime object method in this part. Three sections are covered in this part: inserting data using the general runtime object method is discussed first. 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 against 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 those data manipulations if you developed your projects using the runtime objects method.

6.4 THE GENERAL 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 methodology of this method, as well as some keynotes in using this method to perform the data actions again the databases.

As you know, ADO.NET provides different classes to help users to develop professional data-driven applications by using different methods to perform specific data actions, such as inserting data, updating data, and deleting data. For the 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 method ExecuteNonQuery() 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, and 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 relative 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 continue to provide any discussion for this method in this part.

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, and then build a query statement and attach some useful parameter into that query for the newly 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 object 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, and then illustrate the coding differences between these databases by using the real codes for the rest of two sample projects.

Now let's first develop the sample project `SQLInsertRTOObject` to insert data into the SQL Server database using the runtime object method. Recall that in Sections 5.18.3–5.18.5 in Chapter 5, we discussed how to select data for the Faculty, Course, and Student Form windows using the runtime object 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.

In order to avoid the duplication on the coding process, we will modify an existing project named `SQLSelectRTOObject` we developed in Chapter 5 to create our new project `SQLInsertRTOObject` used in this section.

6.5 INSERT DATA INTO THE SQL SERVER DATABASE USING THE RUNTIME OBJECT METHOD

Open the Windows Explorer and create a new folder such as `Chapter 6`, and then browse to the folder `DBProjects\Chapter 5` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy the project `SQLSelectRTOObject` to the new folder, such as `C:\Chapter 6`. Change the name of the project folder from `SQLSelectRTOObject` to `SQLInsertRTOObject`, and change the name of the project `SQLSelectRTOObject.vbproj` to `SQLInsertRTOObject.vbproj`. Then double-click on the `SQLInsertRTOObject.vbproj` to open this project. On the opened project, perform the following modifications to get our desired project:

- Go to `Project\SQLInsertRTOObject Properties` menu item to open the project's property window. Change the `Assembly name` and the `Root namespace` from `SQLSelectRTOObject` to `SQLInsertRTOObject`, respectively.
- Click on the `Assembly Information` button to open the `Assembly Information` wizard. Change the `Title` and the `Product` to `SQLInsertRTOObject`. Click on the `OK` button to close this wizard.

Go to `File\Save All` and `Build\Rebuild SQLInsertRTOObject` to save those modifications and rebuild the project. Now we are ready to develop our graphic user interfaces based on our new project `SQLInsertRTOObject`.

6.5.1 Insert Data into the Faculty Table for the SQL Server Database

Let's first discuss inserting data into the Faculty table for the SQL Server database. To insert data into the Faculty data table, we can use the Faculty Form window we built in the last section.

6.5.1.1 Develop the Codes to Insert Data into the Faculty Table

The codes for this data insertion are divided into three steps the data validation before the data insertion, the data insertion using the runtime object method, and the data vali-

dation 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 can be read back and displayed in the Faculty form window. Let's begin with the coding process for the first step now.

6.5.1.1.1 Validate Data Before the Data Insertion First, let's handle the coding development for the data validation before the data insertion.

This data validation can be performed by calling one user-defined subroutine and one user-defined function. The subroutine is named `InitFacultyInfo()`, and it is used to set up a mapping relationship between each item in the string array `FacultyTextBox()` and each textbox. The function is named `CheckFacultyInfo()` and it is used to scan and check all textboxes to make sure that no one of them is empty.

Open the code window of the Faculty Form window, and enter the codes shown in Figure 6.20 into this window to create a user-defined subroutine `InitFacultyInfo()`.

The `FacultyTextBox()` is a zero-based string array, and it 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 of them is empty later.

Open the code window of the Faculty Form window and enter the codes shown in Figure 6.21 into this window to create a user-defined function `CheckFacultyInfo()`.

The functionality of this function is:

- A. A For loop is used to scan each textbox in the `FacultyTextBox()` string array to check whether any of them is empty. A message will be displayed if this situation happens, and the function exists to allow user to fill all textboxes.
- B. If the Faculty Image box is empty, which means that the user wants to use a default faculty photo with this new data insertion, then we need to display a message to indicate this situation, and a default faculty image will be used. The function returns a zero to indicate that this validation is successful.

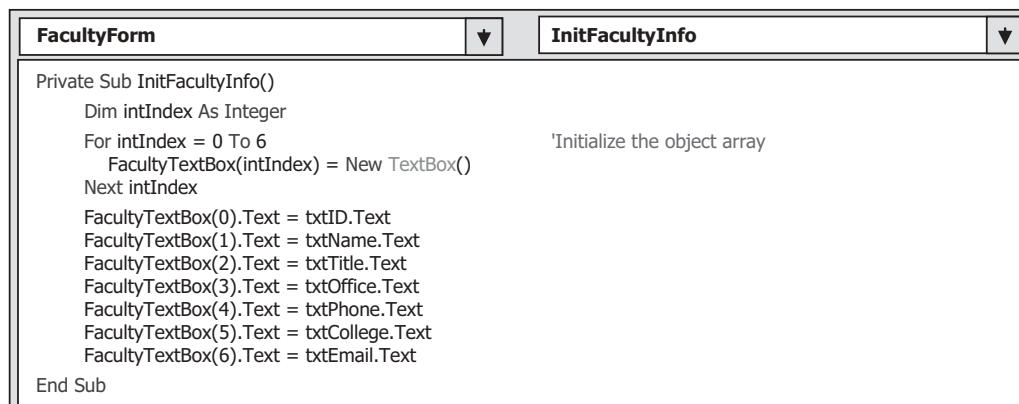


Figure 6.20. The codes for the user-defined subroutine `InitFacultyInfo()`.

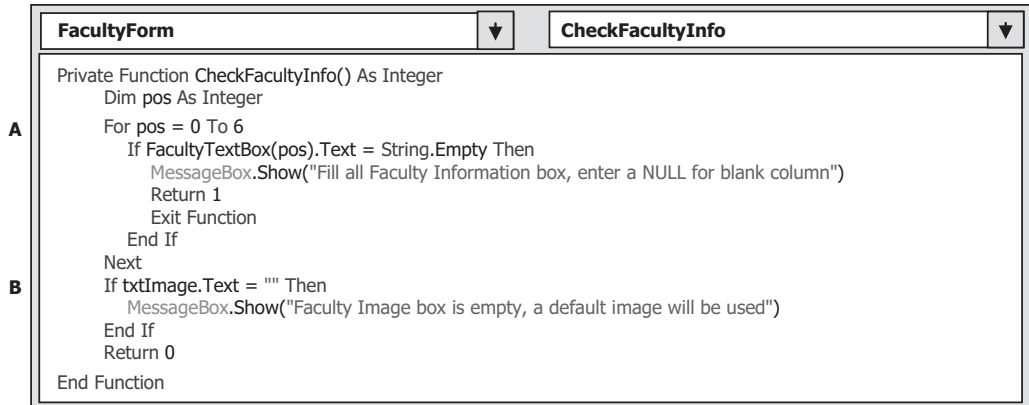


Figure 6.21. The codes for the function CheckFacultyInfo().

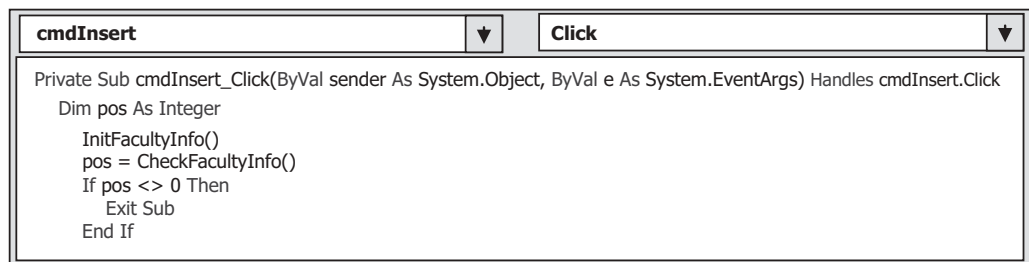


Figure 6.22. The first piece of codes for the Insert button's Click event procedure.

Now let's develop the code for the Insert button's Click event procedure to call the subroutine and the function we built above to perform the data validation before the data insertion. Open the Insert button's Click event procedure, and enter the codes that are shown in Figure 6.22 into this event procedure.

The function of this piece of codes is straightforward and easy to be understood. First, the user-defined subroutine InitFacultyInfo() is called to set up the mapping relationship between each item in the string array FacultyTextBox() and each textbox. Then the user-defined function CheckFacultyInfo() is executed to check and make sure that no textbox is empty. If any of textboxes is empty, the function returns a nonzero value, and the procedure is exited to allow users to reenter information to the associated textboxes until all of them are filled with the desired information.

At this point, we completed the coding process for the data validation before the data insertion. Now let's do our coding process for the data insertion.

6.5.1.1.2 Insert Data into the Faculty Table The main coding job is performed inside the Insert button's Click event procedure. We have already developed some codes at the beginning of this procedure in the last section. Now let's continue to complete this coding process.

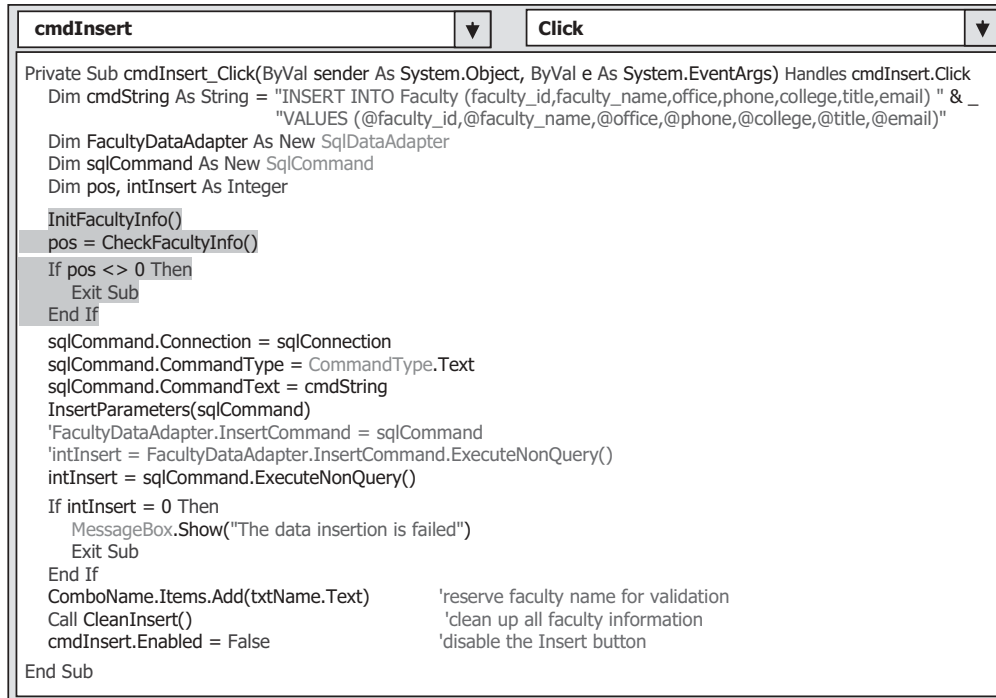


Figure 6.23. The second piece of codes for the Insert button's event procedure.

Open the Insert button Click event procedure and enter the codes that are shown in Figure 6.23 into this event procedure. The codes we developed before for this event procedure are indicated with a gray background.

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

- A.** The SQL INSERT statement is declared first, and it contains seven parameters followed by the command 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. Two local integer variables, pos and intInsert, are also declared at this part. The pos is used to hold the returned value of the calling the function CheckFacultyInfo(), and the intInsert is used to hold the returned value of executing the ExecuteNonQuery() method of the Command class.
- C.** The Command instance is initialized with the Connection, CommandType, and CommandText properties of the Command class.
- D.** Another user-defined subroutine InsertParameters() is called to fill parameters to the Parameters collection of the Command instance. Figure 6.24 shows the detailed codes for this subroutine later. Another way to execute this insert action, which has been commented out, is to call the FacultyDataAdapter with its property of the InsertCommand.
- E.** After the Command instance is initialized, the ExecuteNonQuery() method of the Command class is called to insert the new record into the Faculty table in the database.

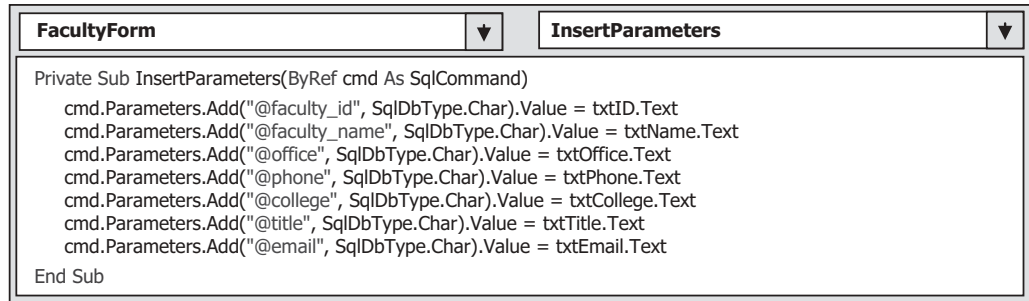


Figure 6.24. The codes for the user-defined subroutine `InsertParameters()`.

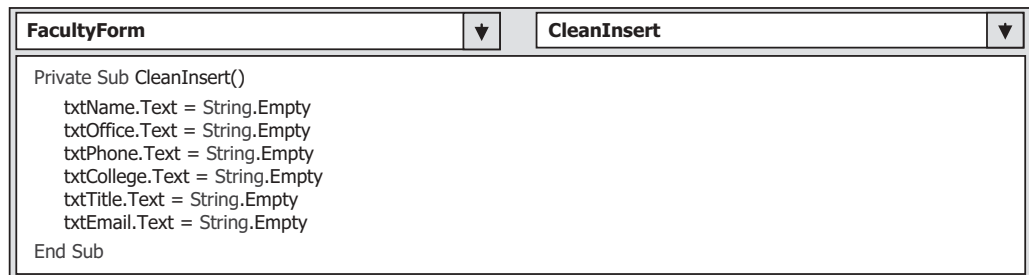


Figure 6.25. The codes for the user-defined subroutine `CleanInsert()`.

- F.** The `ExecuteNonQuery()` method returns an integer as the feedback to indicate whether this insertion is successful or not. The value of this returned integer equals the number of newly inserted records into the Faculty data table. A returned zero means that no new record has been inserted into the Faculty table and this insertion has failed. A warning message would be displayed, and the procedure is exited if this situation happened.
- G.** The newly inserted faculty name is added into the Faculty Name combo box for the validation purpose after this data insertion.
- H.** A cleaning job is performed to clean up the contents of all textboxes that contain the newly inserted faculty information, except the Faculty ID.
- I.** The Insert button is disabled after this data insertion to avoid the 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 codes for the user-defined subroutine `InsertParameters()` are shown in Figure 6.24.

This piece of codes is easy; each piece of faculty-related information stored on the associated textbox is assigned to each matched parameter by using the `Add()` method. One point to be noted is that the `@` symbol must be prefixed before each parameter since this is the requirement of the SQL Server database operations.

The codes for the user-defined subroutine `CleanInsert()` are shown in Figure 6.25. The function of this piece of codes is simple; just clean up the contents of all textboxes

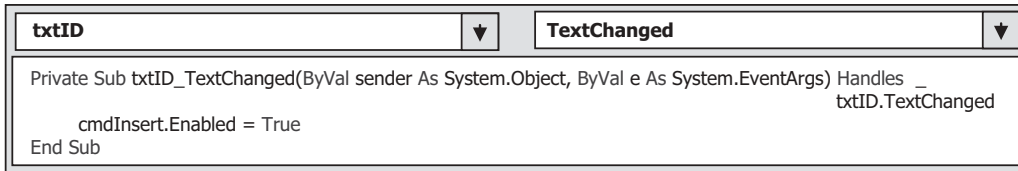


Figure 6.26. The codes for the Faculty ID TextChanged event procedure.

that stored seven pieces of the newly inserted faculty information, except the textbox FacultyID.

Another coding job is for the Faculty ID textbox, exactly for the TextChanged event procedure of the Faculty ID textbox. As we mentioned, in order to avoid multiple insertions of the same data, the Insert button should be disabled after one data is inserted into the database. This Insert button will be enabled again when the content of the Faculty ID textbox is changed, which means that a different new record is ready to be inserted into the database. The codes for that event procedure are shown in Figure 6.26.

At this point, we have finished all coding development for this data insertion action for the Faculty Form window. Before we can run the project to test the function of this data insertion, we need first to develop the codes to perform the validation for this data insertion.

6.5.1.1.3 Validate Data After the Data Insertion To validate the newly inserted faculty record, the same Faculty form window is used and the function of this validation is to read back the inserted data 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 event procedure in Chapter 5 to perform this data query, and, also, we need to make a little modification to the user-defined subroutine ShowFaculty() to complete this data validation.

Open the user-defined subroutine procedure ShowFaculty() and add the codes shown in steps **A** and **B** in Figure 6.27 into this subroutine.

The codes we developed before are indicated with a gray background. Let's have a closer look at this piece of attached codes to see how it works.

- A.** If no matched faculty image can be found, three possibilities exist: first, a faculty data query is being executed and no matched faculty image can be found. Second, a new faculty record is being inserted into the Faculty table and the user does not want to use any faculty image for that inserted faculty record. Third, a new faculty record is being inserted into the Faculty table and the user wants to use a new faculty image for that inserted faculty record, and the name of that new faculty image file has been entered into the Faculty Image textbox. For both the first and the second situations, a default faculty image Default.jpg is used.
- B.** For the third case, the name of new faculty image file is assigned to the FacultyImage variable, which will be displayed later.

Now we have finished all coding process for both data insertion and the data validation after the data insertion. Let's run the project to test the functionalities of the codes we developed 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.



Figure 6.27. The added codes for the subroutine ShowFaculty().

For this test, we want to display a faculty photo named Mhamed.jpg, and we have stored this file in our default folder, which is C:\Chapter 6\SQLInsertRTOObject\bin\Debug.

Now starts 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. Then enter the following data into this form as a new faculty record:

- M99875 Faculty ID textbox
- Mhamed Jones Faculty Name textbox
- Professor Title textbox
- MTC-225 Office textbox
- 750-330-5587 Phone textbox
- University of Chicago College textbox
- mjones@college.edu Email textbox

Then enter Mhamed.jpg into the Faculty Image textbox as the photo file name. Your finished new faculty information is shown in Figure 6.28.

Now click on the Insert button to insert this new faculty record into the database. Immediately, all textboxes, except the Faculty ID, are cleaned up, and the Insert button is disabled after this insertion.

CSE DEPT Faculty Form

Faculty Name_Query Method

Faculty Name: Ying Bai

Query Method: TableAdapter Method

Faculty Information

Faculty ID: M99875

Name: Mhamed Jones

Title: Professor

Office: MTC-225

Phone: 750-330-5587

College: University of Chicago

Email: mjones@college.edu

Faculty Image: Mhamed.jpg

Select Insert Update Delete Back

Figure 6.28. The newly inserted faculty record in the Faculty Form window.

CSE DEPT Faculty Form

Faculty Name_Query Method

Faculty Name: Mhamed Jones

Query Method: TableAdapter Method

Faculty Information

Faculty ID: M99875

Name: Mhamed Jones

Title: Professor

Office: MTC-225

Phone: 750-330-5587

College: University of Chicago

Email: mjones@college.edu

Faculty Image: Mhamed.jpg

Select Insert Update Delete Back

Figure 6.29. An example of the data validation result.

To check and validate this faculty record insertion, click on the drop-down arrow of the Faculty Name combo box, and you can find that the newly inserted faculty name Mhamed Jones is in there. Click that name to select it, and then click on the **Select** button to try to read back that newly inserted record from the database and display it in this Faculty form window.

Immediately, you can find that all pieces of information about that newly inserted faculty record, including the faculty image, are displayed in the associated textboxes, which is shown in Figure 6.29. Our data insertion is successful.

One potential bug that exists in this data validation is that each time when you enter a new piece of faculty information into the database, the faculty name must not be identical. Some readers may argue with me for this: the different faculty member is identified by the faculty ID, not by name, and the faculty ID is the primary key in the Faculty table. Yes, that is true. But the issue is that in this application, we use the faculty name, not 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 will be returned if many faculty members who have the same name even they have the different faculty IDs in this application.

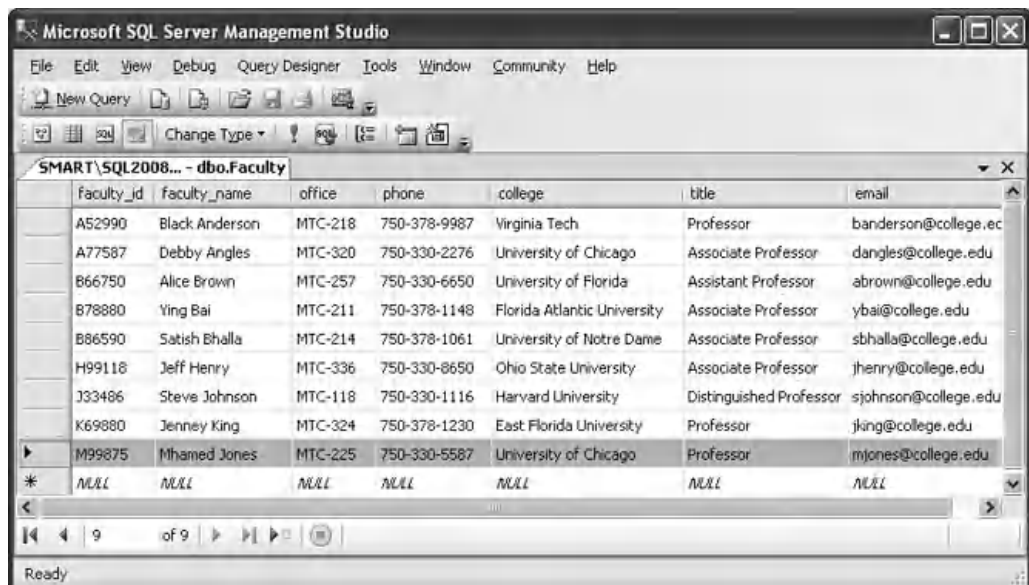
Click on the **Back** and then **Exit** buttons to terminate our project.

Another way to confirm this faculty record insertion is to open our SQL Server 2008 Express sample database using the Microsoft SQL Server Management Studio. To do that, go to **start|All Programs|Microsoft SQL Server 2008|SQL Server Management Studio**. On the opened sample database **CSE_DEPT**, expand to the **Tables** folder and the **dbo.Faculty** table. Right-click on this table and select **Edit Top 200 Rows** item to open this table. You can find that a new faculty member **Mhamed Jones** has been inserted into this table, as shown in Figure 6.30.

A completed project **SQLInsertRTOObject** can be found in the folder **DBProjects\Chapter 6** that is located at the Wiley ftp site (refer to Figure 1.2 in 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 query strings, including the Connection string and the SELECT query string used in the LogIn form.
- SELECT query string, INSERT query string, and Parameter strings used in the Faculty form.



The screenshot shows the Microsoft SQL Server Management Studio interface. The table 'dbo.Faculty' is open, displaying 10 rows of data. The columns are: faculty_id, faculty_name, office, phone, college, title, and email. The row for 'Mhamed Jones' is highlighted with a mouse cursor.

faculty_id	faculty_name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	banderson@college.ec
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
M99875	Mhamed Jones	MTC-225	750-330-5587	University of Chicago	Professor	mjones@college.edu
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 6.30. The newly inserted faculty member Mhamed Jones.

All other codes are identical. We will show those differences and discuss how to insert data into the Microsoft Access and Oracle database in Sections 6.6 and 6.7.



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

6.6 INSERT DATA INTO THE MICROSOFT ACCESS DATABASE USING THE 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 coding parts are identical without modifications. So we can use all 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 database.

First, let's modify the project `SQLInsertRTOObject`. Open the Windows Explorer to create a new folder, such as `Chapter 6`, and then copy the project `SQLInsertRTOObject` from the folder `DBProjects\Chapter 6` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1) to our new folder `Chapter 6`. Change the name of the solution and the project folders to `AccessInsertRTOObject`. Also, change the name for the following files:

1. `AccessInsertRTOObject.sln`
2. `AccessInsertRTOObject.vbproj`
3. `AccessInsertRTOObject.vbproj.user`

Refer to Section 6.2.1 to complete this rename operation.

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 `SQLInsertRTOObject` to get a new project `AccessInsertRTOObject` to perform our data insertion job using the runtime object method. Basically, we need to perform the following modifications:

1. Imports the `OleDb` namespace—all `OleDb` data components are defined here.
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 strings—select and insert the correct faculty information.
5. Modifications to other forms—change the connection object.

Modifications 2 and 3 are included in the `LogIn` form window with the `LogIn` data table. Modification 4 is performed in the Faculty form with the Faculty data table in the Microsoft Access database. Let's do these modifications one by one now.

6.6.1 Modify the Imports Commands and the ConnModule

First, let's open the code window of the LogIn form by clicking on the View Code button from the Solution Explorer window. On the opened code window, move your cursor to the top and modify two Imports commands to:

```
Imports System.Data
Imports System.Data.OleDb
```

In this way, we finished the modification for the Imports commands in the LogIn form window. Make same modifications to the rest of form windows:

- ConnModule
- Faculty Form
- Course Form

Since we will not use the Student form in this application, therefore, it does not matter for this modification. Now let's modify the Connection string for the LogIn form.

Now open the code window of the ConnModule and change the global connection object from the `sqlConnection` to `accConnection`, class from `SqlConnection` to `OleDbConnection`, respectively, as shown in Figure 6.31.

6.6.2 Modify the 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 Form_Load event procedure of the LogIn form since the connection string is defined in there.

Open the Form_Load event procedure of the LogIn form window. Change the name and the content of the Connection string, as shown in Figure 6.32.

Let's have a closer look at these modifications to see how they work.

- The modifications to the Imports commands are shown here.
- Change the prefix of the Connection object from `sql` to `acc`, and change the prefix of the Connection class from `Sql` to `OleDb` since we need to use the Access database and OleDb data provider in this project.
- Change the name and the content of the connection string as shown in step C.

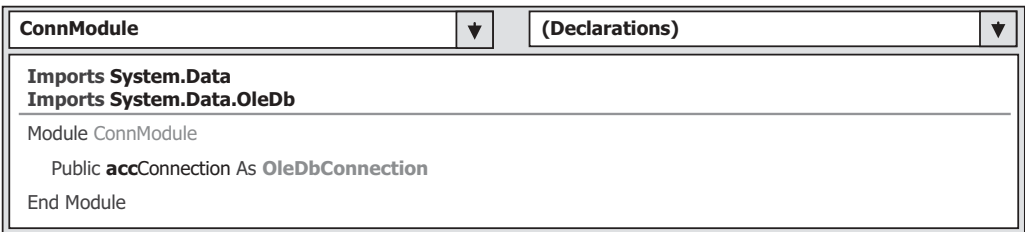


Figure 6.31. The modified codes for the ConnModule.

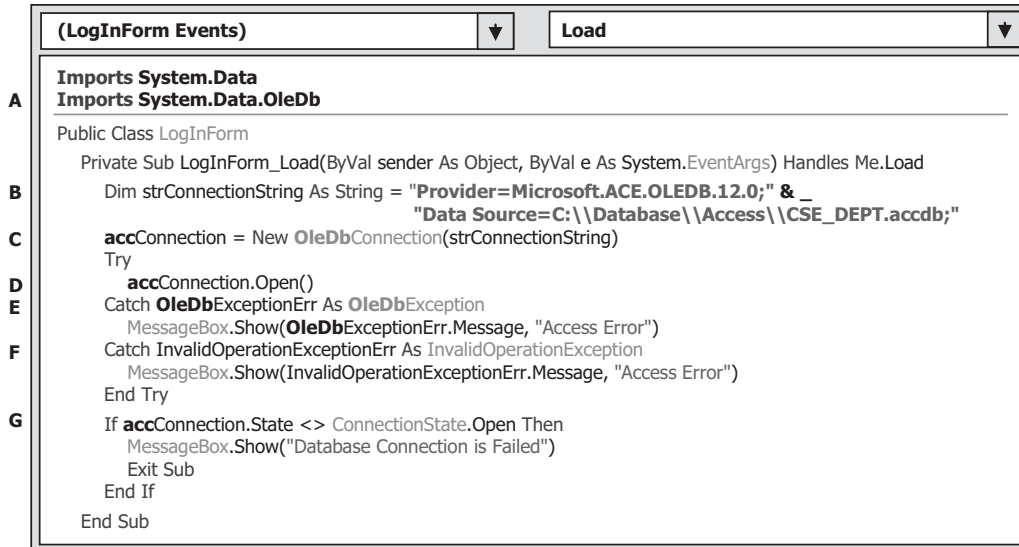


Figure 6.32. The modified codes to the Connection String.

Make the modifications to the following items, which are shown in Figure 6.32, from steps **D** to **G**; all modifications have been highlighted in bold.

Go to the File|Save All to save those modifications. Next, let's continue to modify the login query strings in the LogIn form.

6.6.3 Modify the LogIn Query Strings

In this application, two LogIn buttons are used for this form since two login methods are utilized. To save time and space, we only modify one method: TableAdapter Method. Open this event procedure by double-clicking on the TabLogIn button from the LogIn form window, and do the modifications that are shown in Figure 6.33 for this event procedure.

Let's take a closer look at these modifications to see how they work.

- A.** Most parts of this query string are working fine with the Microsoft Access database, and the only modification is the LIKE operator used in the WHERE clause. Change these two LIKE operators to the comparison operator = before two parameters @Param1 and @Param2, respectively. This is the syntax used in the Microsoft Access database.
- B.** Starting from step **B**, change the prefix for all Data Provider classes used in this event procedure from Sql to OleDb. All modifications have been highlighted in bold. Steps involved in these modifications are from **B** to **D**.
- E.** Starting from step **E**, change the prefix for all Data Provider objects used in this event procedure from sql to acc. All modifications have been highlighted in bold. Steps involved in these modifications are **E**, and from steps **G** to **R**.
- F.** Change the prefix for both Data Provider classes and objects from sql to acc, and from Sql to OleDb, respectively. The step involved in this modification is **F** only.

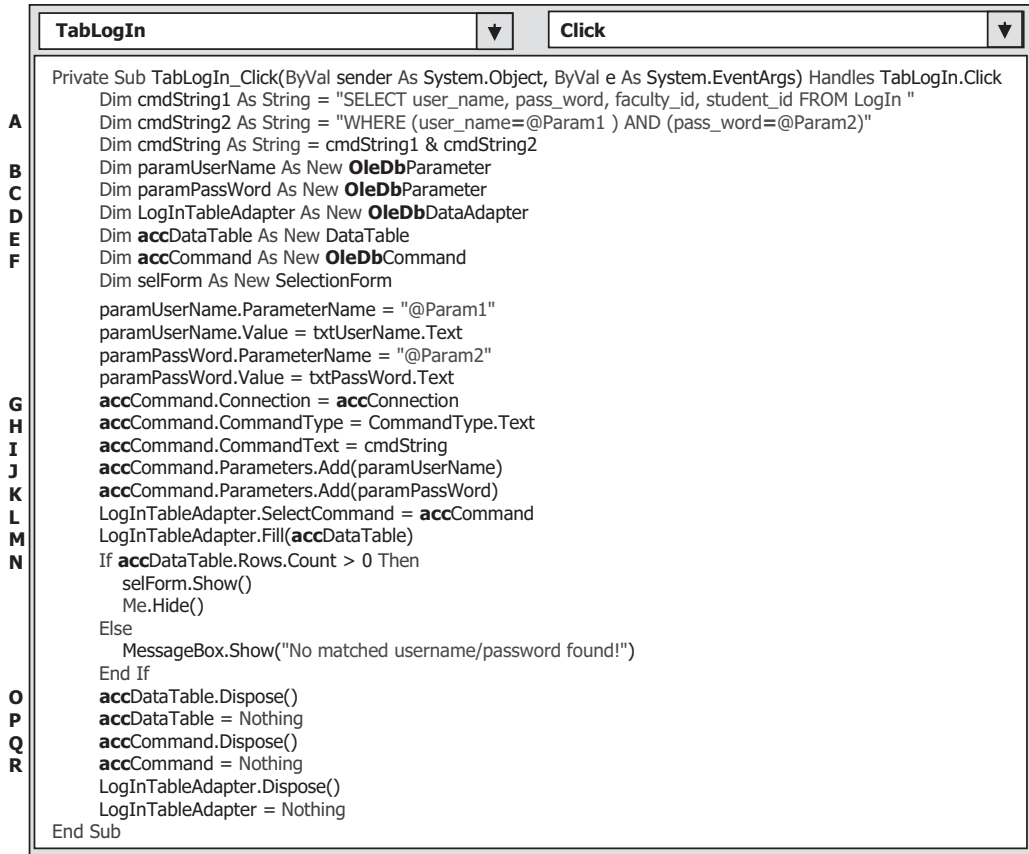


Figure 6.33. The modifications to the LogIn query string.

You can perform similar modifications to the codes in the **ReadLogIn** button Click event procedure and the **Cancel** button's Click event procedures.

Now let's go to the Faculty form to modify the Faculty table query string.

6.6.4 Modify the Faculty Query String

First, make sure that the Imports commands that are located at the top of this form are modified as we did in Section 6.6.1. Then open the **Form_Load** event procedure and change the Connection object, which is located at the first line, from the **sqlConnection** to the **accConnection**, as shown below:

```
If accConnection.State <> ConnectionState.Open Then
```

Now open the **Select** button's Click event procedure by double-clicking on this button from the Faculty form window, and perform the modifications that are shown in Figure 6.34 to this event procedure.

Let's have a closer 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 database and the only modification is to change

	cmdSelect	▼	Click	▼
	Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click			
	Dim cmdString1 As String = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty "			
A	Dim cmdString2 As String = "WHERE faculty_name=@facultyName"			
B	Dim cmdString As String = cmdString1 & cmdString2			
C	Dim paramFacultyName As New OleDbParameter			
D	Dim FacultyTableAdapter As New OleDbDataAdapter			
E	Dim accCommand As New OleDbCommand			
F	Dim accDataReader As OleDbDataReader			
	Dim accDataTable As New DataTable			
	Dim ds As New DataSet()			
	paramFacultyName.ParameterName = "@facultyName"			
	paramFacultyName.Value = ComboName.Text			
G	accCommand.Connection = accConnection			
H	accCommand.CommandType = CommandType.Text			
I	accCommand.CommandText = cmdString			
J	accCommand.Parameters.Add(paramFacultyName)			
	Call ShowFaculty(ComboName.Text)			
	If ComboMethod.Text = "TableAdapter Method" Then			
K	FacultyTableAdapter.SelectCommand = accCommand			
L	FacultyTableAdapter.Fill(accDataTable)			
M	If accDataTable.Rows.Count > 0 Then			
N	Call FillFacultyTable(accDataTable)			
	Else			
	MessageBox.Show("No matched faculty found!")			
	End If			
O	accDataTable.Dispose()			
P	accDataTable = Nothing			
	FacultyTableAdapter.Dispose()			
	FacultyTableAdapter = Nothing			
	ElseIf ComboMethod.Text = "DataReader Method" Then			
Q	accDataReader = accCommand.ExecuteReader			
R	If accDataReader.HasRows = True Then			
S	Call FillFacultyReader(accDataReader)			
	Else			
	MessageBox.Show("No matched faculty found!")			
	End If			
T	accDataReader.Close()			
U	accDataReader = Nothing			
	Else '----- LINQ To DataSet method is selected			
V	FacultyTableAdapter.SelectCommand = accCommand			
	FacultyTableAdapter.Fill(ds, "Faculty")			
	Dim facultyinfo = From fi In ds.Tables("Faculty").AsEnumerable()			
	Where fi.Field(Of String)("faculty_name").Equals(ComboName.Text) Select fi			
	For Each fRow In facultyinfo			
	txtID.Text = fRow.Field(Of String)("faculty_id")			
	txtName.Text = fRow.Field(Of String)("faculty_name")			
	txtTitle.Text = fRow.Field(Of String)("title")			
	txtOffice.Text = fRow.Field(Of String)("office")			
	txtPhone.Text = fRow.Field(Of String)("phone")			
	txtCollege.Text = fRow.Field(Of String)("college")			
	txtEmail.Text = fRow.Field(Of String)("email")			
	Next			
	End If			
W	accCommand.Dispose()			
	accCommand = Nothing			
	End Sub			

Figure 6.34. Modifications to the Faculty query string.

the LIKE operator, which is inside the cmdString2 and located before the dynamic parameter @facultyName, to the comparison operator = since this is the requirement of the Microsoft Access database.

- B.** Change the prefix of all Data Provider-related classes from the **Sql** to the **OleDb**. Steps involved in these modifications are **B** and **C**. All modifications have been highlighted in bold.
- D.** Change the prefix of all Data Provider related classes and objects from the **Sql** to the **OleDb**, and from the **sql** to the **acc**. Steps involved in these modifications are **D** and **E**. All modifications have been highlighted in bold.
- F.** Change the prefix of all Data Provider related objects from the **sql** to the **acc**. Steps involved in these modifications are from **F** to **W**.

Another modification is for the user-defined subroutine FillFacultyReader(). The data type of the argument FacultyReader should be changed from SqlDataReader to OleDbDataReader.

6.6.5 Modify the Faculty Insert String

Open the Insert button Click event procedure and perform the modifications shown in Figure 6.35 to the codes in this event procedure.

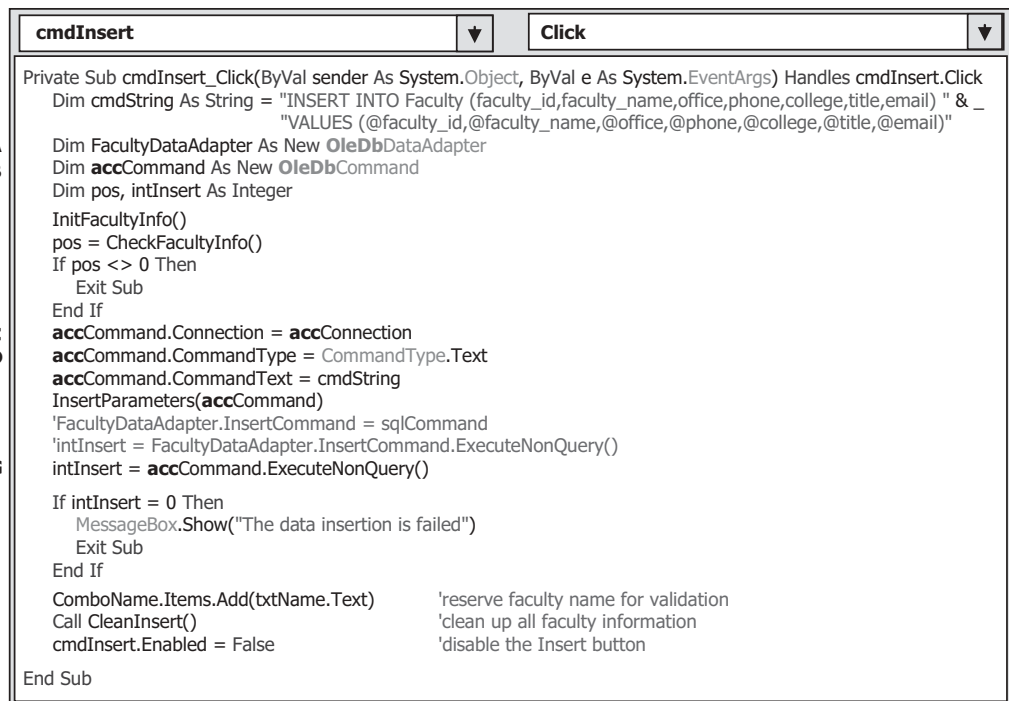


Figure 6.35. Modified codes for the Insert button Click event procedure.

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

- A.** Change the prefix for all Data Provider-related classes from `Sql` to `OleDb`. Steps involved in this modification are **A** and **B**.
- C.** Change the prefix for Data Provider-related objects from `sql` to `acc`. Steps involved in this modification are **C** through **G**.

Another modification is to the user-defined subroutine procedure `InsertParameters()`. The first modification is to change the data type of the passed command object `cmd` from `SqlCommand` to `OleDbCommand`. The second modification is to change the data type of all seven columns passed into the `Faculty` table from `SqlDbType` to `OleDbType`.

Before we can run the project to insert data into the database, we need to finish the rest of modifications to other forms. Basically, this modification is to change the connection object for all other forms to match to the Microsoft Access database connection.

6.6.6 Modifications to Other Forms

The following four forms contain this connection object: `Selection`, `Course`, `SP`, and `Student`. In this project, we only need to use the `Selection` and `Course` Form, so we only need to modify the connection object for those two forms.

Perform the following operations to complete these modifications:

1. Open the code window of the `Course` Form, and then open the `Form_Load` event procedure of that form, and change the connection object name from the `sqlConnection` to the `accConnection`.
2. Open the `Exit` button's `Click` event procedure of the `Selection` form to change the connection object, too. Your finished modification for this connection object should match the one that is shown below:

```
If accConnection.State <> ConnectionState.Open Then
    accConnection.Close()
End If
```

Besides the connection object, perform the following operations to complete the modifications to the Data Provider-related classes and objects in the `Course` form:

1. Open the `Select` button `Click` event procedure, and change the `LIKE` operator to the comparison operator `=` in the query string `cString2`.
2. Change the data type of the argument `CourseReader`, which is located in the user-defined subroutine `FillCourseReader()`, from `SqlDataReader` to `OleDbDataReader`.
3. Change the data type of the argument `CourseReader`, which is located in the user-defined subroutine `FillCourseReaderTextBox()`, from `SqlDataReader` to `OleDbDataReader`.
4. Change the prefix of all Data Provider-related classes from `Sql` to `OleDb`, and the prefix of all Data Provider-related objects from `sql` to `acc`, for the codes in the `Select` button `Click` event procedure and the `CourseList` `SelectedIndexChanged` event procedure.

Since we will not use the `Student` form for this project, but you can modify the connection object from `sqlConnection` to `accConnection` in the `Form_Load` event

Figure 6.36. The running status of the Faculty Form window.

procedure in the Student and the SP Forms, and comment out the `SqlConnection` object in the subroutine `BuildCommand()` in the Student form and in the `Select` button Click event procedure in the SP Form to enable us to build and run this project.

Now let's run the project to test our data insertion functionality. 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, which is shown in Figure 6.36.

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
- MTC-118 Office textbox
- 750-330-7788 Phone textbox
- University of Miami College textbox
- mkims@college.edu Email textbox

Keep the `Faculty Image` textbox empty since we do not want to include a photo for this faculty record. Your finished new faculty information window should match the one that is shown in Figure 6.36.

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 and all textboxes, except the `Faculty ID`, becomes empty.

To confirm this data insertion, click on the drop-down arrow of the `Faculty Name` combo box from the Faculty form, and you can find that the name of the newly inserted faculty `Mattin Kims` has been in this box. Click on it to select this faculty and then click on the `Select` button to try to retrieve this newly inserted record from the database and

CSE DEPT Faculty Form

Faculty Image

Faculty Name_Query Method

Faculty Name: Maltin Kims

Query Method: TableAdapter Method

Faculty Information

Faculty ID: M39558

Name: Maltin Kims

Title: Associate Professor

Office: MTC-118

Phone: 750-330-7788

College: University of Miami

Email: mkims@college.edu

Select Insert Update Delete Back

Figure 6.37. The insert data validation process.

display it in this form. Immediately, you can find that all seven pieces of information about this new faculty is shown in this form, which is shown in Figure 6.37.

Another way to confirm this data insertion is to open the Faculty table in our sample database `CSE_DEPT.accdb`, which is located at `C:\Database` in your computer, and you can find that a new faculty record has been inserted into this table.

This is the evidence that our data insertion into the Microsoft Access database is successful! Click on the **Back** and then the **Exit** buttons to close the project.

You can remove some newly added records from this database to keep your table neat if you like. To do that, open the database and the associated data table, and you can do whatever you want.

A completed project `AccessInsertRuntimeObject` can be found in a folder `DBProjects\Chapter 6` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

6.7 INSERT DATA INTO THE ORACLE DATABASE USING THE RUNTIME OBJECTS

Similarly, as we did in the last section for the Microsoft Access database, we can modify the `SQLInsertRuntimeObject` project and make it work for the Oracle database.

Open the Windows Explorer to create a new folder such as `Chapter 6`, and then copy the project `SQLInsertRuntimeObject` from the folder `DBProjects\Chapter 6` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Refer to Section 6.2.1 to rename this project to `OracleInsertRuntimeObject`.

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 to the Oracle database. Basically, we need to perform the following modifications:

1. Imports the Oracle namespace—all Oracle data components are defined here.
2. Database Connection string—make it connect to the Oracle 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 and Data Provider-related classes and objects for all forms.

Modification items 2 and 3 are included in the LogIn form window with the LogIn data table, and modification item 4 is performed in the Faculty form with the Faculty data table in the Oracle database. Let's do these modifications one by one.

6.7.1 Add the Oracle Driver Reference and Modify the Imports Commands

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. Also starting from .NET Frameworks 4.0, Microsoft is no longer to support any driver for the Oracle database. Therefore, we need to use a third-party Oracle driver ([dotConnect 6.30](#)) to handle all data actions for the Oracle database in this application. Refer to Appendix F to get more detailed information about how to download and install this driver, and how to add references related to this Oracle driver.

Now let's first add these driver-related references to our new project. Perform the following operations to complete this addition:

1. Go to the Solution Explorer window, right-click on our project `OracleInsertRTOObject` and select **Add Reference** item from the pop-up menu to open the Add Reference wizard, which is shown in Figure 6.38.
2. Keep the .NET tab selected and browse down the list until you find two Oracle-related libraries, `Devart.Data` and `Devart.Data.Oracle`. Then select both of them and click on the OK button to add these references into our project.
3. 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 that both references have been added into the project.

Next, let's open the code window of the LogIn form by clicking on the View Code button from the Solution Explorer window. On the opened code window, move your cursor to the top and modify two Imports commands to:

```
Imports Devart.Data  
Imports Devart.Data.Oracle
```

In this way, we finished the modification for the Imports commands in the LogIn form window. Make the same modifications to the rest of form windows:

- Faculty Form
- Course Form
- ConnModule

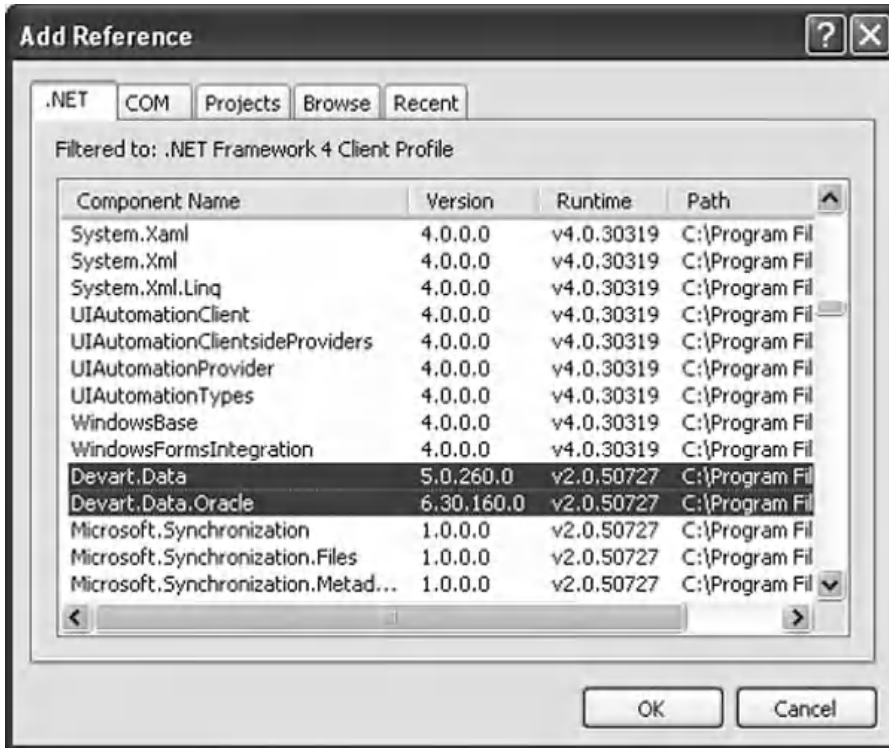


Figure 6.38. The newly added reference libraries.

Since we will not use the Student and the SP forms for our data insertion, no modification should be made to them. Now let's modify the Connection string for the LogIn form.

6.7.2 Modify the 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 code window of the ConnModule to modify the global connection object.

Change the connection object from `SqlConnection` to `oraConnection` and connection class from `SqlConnection` to `OracleConnection`, as shown in Figure 6.39. The modified codes have been highlighted in bold.

Now open the `Form_Load` event procedure of the LogIn form since the connection string is defined in there. Change the name and the content of the Connection string, as shown in Figure 6.40.

Let's have a closer look at these modifications to see how they work.

- A.** The modifications to the Imports commands are shown here.
- B.** Change the name and the content of the connection string as shown in step **B**.

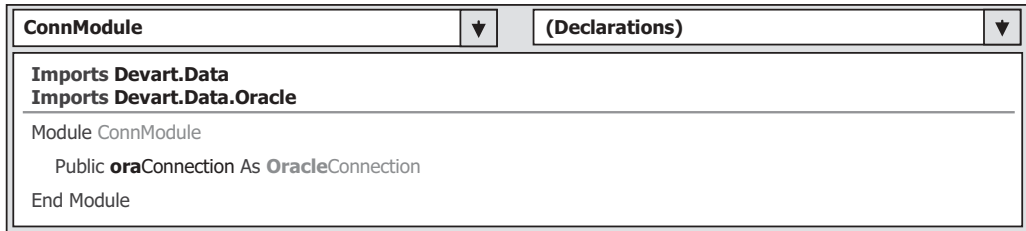


Figure 6.39. The modified codes for the ConnModule.

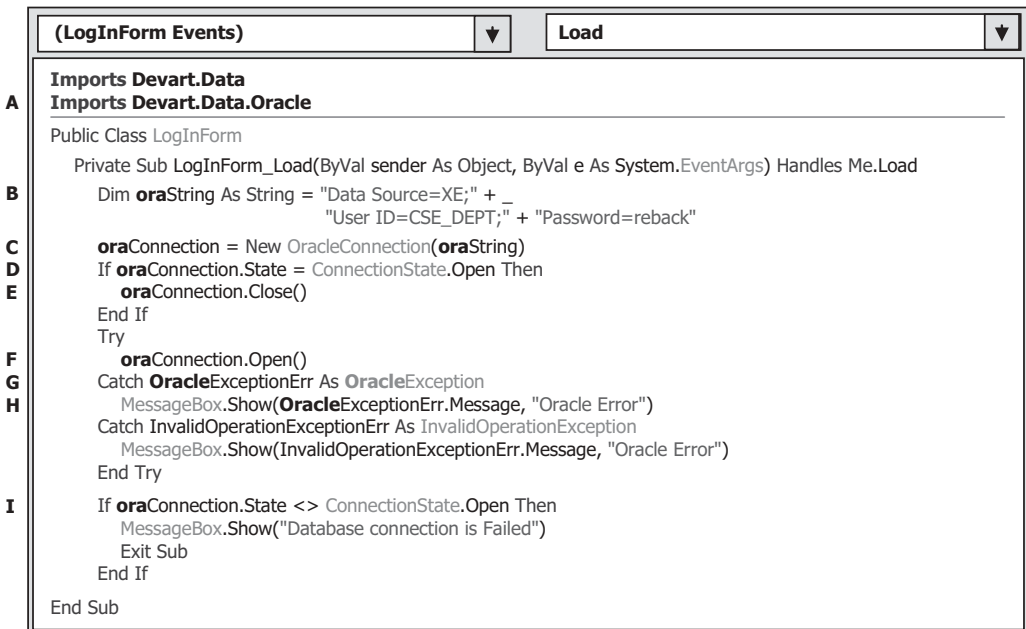


Figure 6.40. Modifications to the Connection string in LogIn form.

- C. Change the prefix of the Connection object from `sql` to `ora`, and change the prefix of the Connection class from `Sql` to `Oracle` since we need to use the Oracle database and Oracle data provider in this project. Steps involved in these modifications are steps C through I.

Go to **FileSave All** to save those modifications. Next, let's continue to modify the login query strings in the LogIn form.

6.7.3 Modify the LogIn Query Strings

In this application, two LogIn buttons are used for this form since two login methods are developed. To save time and space, we only modify one method, the TableAdapter method. Open this event procedure by double-clicking on the TabLogIn button from the



Figure 6.41. Modifications to the login query string in LogIn form.

LogIn form window, and perform the following modifications that are shown in Figure 6.41 for this event procedure.

Let's take a closer look at these modifications to see how they work.

- A.** Most parts of this query string are working with the Oracle database, and the only modification is the LIKE operator used in the WHERE clause. Change these two LIKE operators to the comparison operator = : before two parameters name and word, respectively. This is the syntax used in the Oracle database.
- B.** Starting from **B**, change the prefix for all Oracle classes used in this event procedure from Sql to Oracle. All modifications have been highlighted in bold. Steps involved in these modifications are steps **B** through **D**.
- E.** Starting from **E**, change the prefix for all Oracle objects used in this event procedure from sql to ora. All modifications have been highlighted in bold. Steps involved in these modifications are step **E** and steps **G** through **R**.
- F.** Change the prefix for both Oracle classes and objects from sql to ora, and from Sql to Oracle, respectively. Step involved in this modification is step **F** only.

You can perform the similar modifications to the codes in the `ReadLogin` and the `Cancel` button's event procedures.

Now, let's go to the `Faculty` form to modify the `Faculty` table query string.

6.7.4 Modify the Faculty Query String and Query Related Codes

First, make sure that the `Imports` commands that are located at the top of this form are modified as we did in Section 6.7.1. Then, open the `Form_Load` event procedure and change the `Connection` object, which is located at the first line, from the `SqlConnection` to the `oraConnection`, as shown below:

```
If oraConnection.State <> ConnectionState.Open Then
```

Now, open the `Select` button's `Click` event procedure and perform the modifications that are shown in Figure 6.42 to this event procedure.

Let's have a closer 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 `LIKE` operator, which is in the `cmdString2` and located before the dynamic parameter `facultyName`, to the Oracle comparison operator `=` : since this is the requirement of the Oracle database. Also, remove the `@` symbol before the parameter `facultyName`.
- B.** Change the prefix of all Oracle data classes from the `Sql` to the `Oracle`. Steps involved in these modifications are **B** and **C**. All modifications have been indicated in bold.
- D.** Change the prefix of all Oracle data classes and objects from `Sql` to `Oracle`, and from `sql` to `ora`. Steps involved in these modifications are **D** and **E**. All modifications have been highlighted in bold.
- E.** Change the prefix of all Oracle objects from the `sql` to the `ora`. Steps involved in these modifications are **F** through **X**.

The modification in step **Y** is to remove the `@` symbol before the dynamic parameter `facultyName`, and this is the syntax of the Oracle database operations.

Another modification is for the user-defined subroutine `FillFacultyReader()`. The data type of the argument `FacultyReader` should be changed from `SqlDataReader` to `OracleDataReader`.

The next modification is for the codes inside the `Insert` button `Click` event procedure in the `Faculty` Form window.

6.7.5 Modify the Faculty Insert String and Insertion Related Codes

Open the `Insert` button `Click` event procedure and perform the modifications that are shown in Figure 6.43 to this procedure.

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

- A.** The insert query string is modified based on the requirement of the Oracle database operations. The major changes are for the `VALUE` clause, which is to use the colon operator `:` to replace the `@` symbol before each input parameter.

	cmdSelect	▼	Click	▼
	Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click			
	Dim cmdString1 As String = "SELECT faculty_id, faculty_name, office, phone, college, title, email FROM Faculty "			
A	Dim cmdString2 As String = "WHERE faculty_name =: facultyName"			
	Dim cmdString As String = cmdString1 & cmdString2			
B	Dim paramFacultyName As New OracleParameter			
C	Dim FacultyTableAdapter As New OracleDataAdapter			
D	Dim oraCommand As New OracleCommand			
E	Dim oraDataReader As OracleDataReader			
F	Dim oraDataTable As New DataTable			
	Dim ds As New DataSet()			
Y	paramFacultyName.ParameterName = "facultyName"			
	paramFacultyName.Value = ComboName.Text			
G	oraCommand.Connection = oraConnection			
H	oraCommand.CommandType = CommandType.Text			
I	oraCommand.CommandText = cmdString			
J	oraCommand.Parameters.Add(paramFacultyName)			
	Call ShowFaculty(ComboName.Text)			
	If ComboMethod.Text = "TableAdapter Method" Then			
K	FacultyTableAdapter.SelectCommand = oraCommand			
L	FacultyTableAdapter.Fill(oraDataTable)			
M	If oraDataTable.Rows.Count > 0 Then			
N	Call FillFacultyTable(oraDataTable)			
	Else			
	MessageBox.Show("No matched faculty found!")			
	End If			
O	oraDataTable.Dispose()			
P	oraDataTable = Nothing			
	FacultyTableAdapter.Dispose()			
	FacultyTableAdapter = Nothing			
	ElseIf ComboMethod.Text = "DataReader Method" Then			
Q	oraDataReader = oraCommand.ExecuteReader			
R	If oraDataReader.HasRows = True Then			
S	Call FillFacultyReader(oraDataReader)			
	Else			
	MessageBox.Show("No matched faculty found!")			
	End If			
T	oraDataReader.Close()			
U	oraDataReader = Nothing			
	Else '----- LINQ To DataSet method is selected			
V	FacultyTableAdapter.SelectCommand = oraCommand			
	FacultyTableAdapter.Fill(ds, "Faculty")			
	Dim facultyinfo = From fi In ds.Tables("Faculty").AsEnumerable()			
	Where fi.Field(Of String)("faculty_name").Equals(ComboName.Text) Select fi			
	For Each fRow In facultyinfo			
	txtID.Text = fRow.Field(Of String)("faculty_id")			
	txtName.Text = fRow.Field(Of String)("faculty_name")			
	txtTitle.Text = fRow.Field(Of String)("title")			
	txtOffice.Text = fRow.Field(Of String)("office")			
	txtPhone.Text = fRow.Field(Of String)("phone")			
	txtCollege.Text = fRow.Field(Of String)("college")			
	txtEmail.Text = fRow.Field(Of String)("email")			
	Next			
	End If			
W	oraCommand.Dispose()			
X	oraCommand = Nothing			
	End Sub			

Figure 6.42. Modifications to the query codes in the Faculty form.

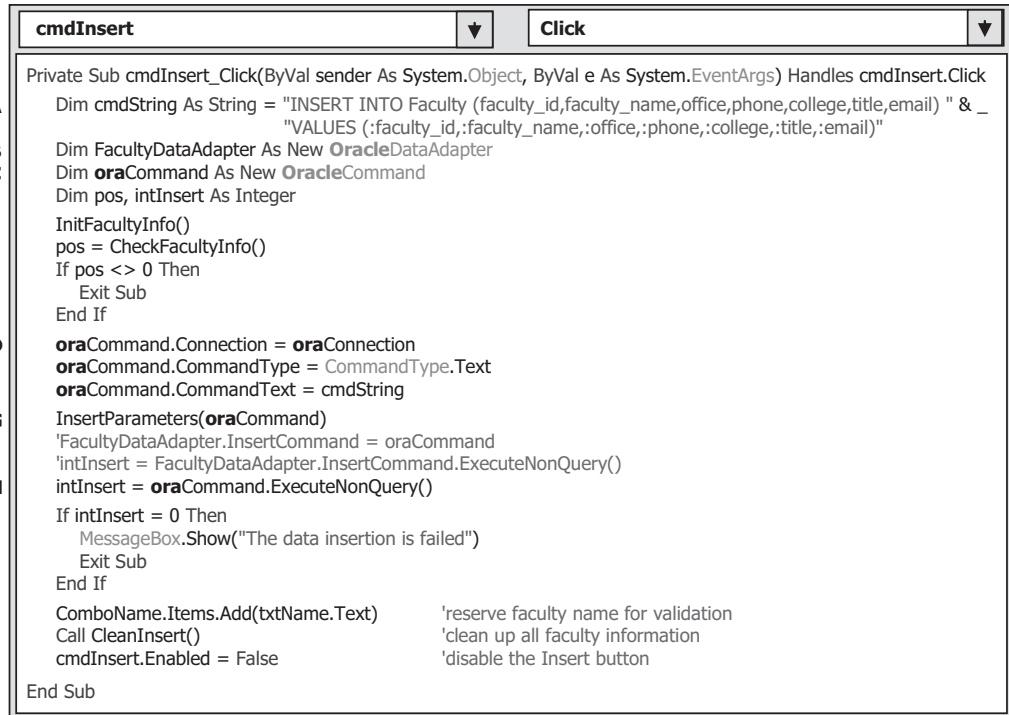


Figure 6.43. Modifications to the Insertion Codes in the Faculty form.

- B.** Change the prefix of all data provider classes from **Sql** to **Oracle**, and the prefix of all data provider objects from **sql** to **ora**, respectively. Steps involved in these modifications are **B** and **C**.
- D.** Change the prefix of all data provider objects from **sql** to **ora**. Steps involved in these modifications are **D** through **H**.

Another modification is for the user-defined subroutine procedure **InsertParameters()**. Perform the following modifications to this subroutine procedure:

- A.** Change the data type of argument **cmd** from **SqlCommand** to **OracleCommand**.
- B.** Change the data type of seven input parameters in the **Add()** method from **SqlDbType** to **OracleDbType**. Also remove the **@** symbol before each input parameter

Your modified subroutine **InsertParameters()** should match the one that is shown in Figure 6.44. All modified codes have been highlighted in bold.

Before we can run the project to insert data into the database, we need to finish the modifications to other forms. Basically, this modification is to change the connection object, data provider-related classes and objects for all other forms to match to the Oracle database operations.

6.7.6 Modifications to Other Forms

The following four forms contain the connection object, data provider-related classes, and objects: Selection, Course, Student, and SP Form. In order to make our project compliable,

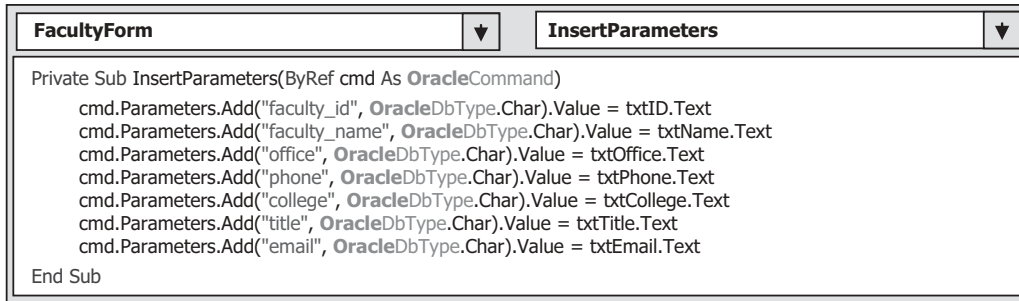


Figure 6.44. The modified codes for the subroutine InsertParameters().

we can modify the related codes in those forms one by one. The codes involved in these modifications are included in the following event procedures:

- Selection Form
 - The Exit button Click event procedure
- Course Form
 - The Form_Load() event procedure
 - The Select button Click event procedure
 - The user-defined subroutine FillCourseReader()
 - The CourseList SelectedIndexChanged() event procedure
 - The user-defined subroutine FillCourseReaderTextBox()
- Student Form
 - Imports namespaces
 - The Form_Load() event procedure
 - The user-defined subroutine BuildCommand()
- SP Form
 - The Form_Load() event procedure
 - The Select button Click event procedure

Let's do the modifications one by one starting from the Selection Form.

6.7.6.1 Modify the Codes in the Selection Form

Open the Exit button Click event procedure in the Selection Form, and change the code line `sqlConnection.Close()` to `oraConnection.Close()`.

6.7.6.2 Modify the Codes in the Course Form

Open the Form_Load event procedure of the Course Form. Change the connection object name from the `sqlConnection` to the `oraConnection`. Your finished modification for this connection object should match the one that is shown below:

```
If oraConnection.State <> ConnectionState.Open Then
```

Modification to the Select button's Click event procedure in the Course form is to change the joined table query string. 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)
```

Change the prefix of all data provider-related classes from `Sql` to `Oracle`, and the prefix of all data provider related objects from `sql` to `ora` in this event procedure. Change the data type of the dynamic parameter `@name` in the `Add()` method from `SqlDbType` to `OracleDbType`, and remove the `@` operator before the `name`.

Modification to the user-defined subroutine `FillCourseReader()` is to change the data type of argument `CourseReader` from the `SqlDataReader` to `OracleDataReader`.

Modifications to the `CouseList SelectedIndexChanged()` event procedure include the following items:

- Replace the `LIKE @courseid` in the query string `cmdString2` with `= : courseid`.
- Change the data type of the dynamic parameter `@courseid` in the `Add()` method from `SqlDbType` to `OracleDbType`, and remove the `@` operator before the `courseid`.
- Change the prefix for all data classes and objects from `Sql` to `Oracle`, and from `sql` to `ora`, respectively.
- Change data type of the argument `CourseReader` in the user-defined subroutine `FillCourseReaderTextBox()` from `SqlDataReader` to `OracleDataReader`.

6.7.6.3 Modify the Codes in the Student Form

Modifications to this form include the following items:

- Change the connection object from `sqlConnection` to `oraConnection` in the `Form_Load` event procedure.
- Comment out the code line `cmdObj.Connection=sqlConnection` in the user-defined subroutine `BuildCommand()`.

6.7.6.4 Modify the Codes in the SP Form

Modifications to this form include the following items:

- Change the connection object from `sqlConnection` to `oraConnection` in the `Form_Load` event procedure.
- Comment out the code line `sqlCmdStudentCourse.Connection=sqlConnection` in the `Select` button `Click` event procedure.

Now we have finished all modifications to our project `OracleInsertRTOBJect`, and now we can run the project to test the data insertion to the Oracle database. A completed project `OracleInsertRTOBJect` can be found in the folder `DBProjects\Chapter 6` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Next, let's discuss how to insert data into the different databases using the stored procedure method. We will use two typical databases, SQL Server and Oracle databases, to illustrate how to insert records into these two kinds of databases.

6.8 INSERT DATA INTO THE DATABASE USING STORED PROCEDURES

In this section, we discuss how to insert data into the database using the stored procedures. We provided a very detailed introduction to the stored procedures and illustrated how to use this method to perform the data query for the Student form and Student table in Section 5.19.8 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 is that we will use that Course form to illustrate inserting data into the Oracle database in the next section.

6.8.1 Insert Data into the 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 use the Course form window to perform the data insertion using the stored procedures. Recall that when we developed that project, an `Insert` button is added into the Course form window. We can use this button to trigger the data insertion function 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 6.2.1 to perform renaming and modifications to the project namespaces and related project files.

The operational procedure of this course data insertion is: as the project runs, after the user has finished the correct login process and selected the item Course Information from the Selection form, the Course form window will be displayed. The form allows users to enter one new course record represented by seven pieces of course information into the appropriate textboxes. By clicking on the `Insert` button, a new course record related to the selected faculty member is inserted into the database.

Let's first develop the codes for our SQL Server stored procedures.

6.8.1.1 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 Course form window is shown up, the user needs to insert new course data based on the faculty name, not the faculty ID. So 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 5.19.6 in Chapter 5. However, it is more flexible and convenient to use stored procedures to perform multiple queries, especially when the queries are for 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 open the Server Explorer window, and 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 5.4.1 in Chapter 5 for the detailed information about 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 wizard, and then enter the codes that are shown in Figure 6.45 into this new 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 Faculty Name combo box, and all other input parameters should be entered by the user to the associated textbox in the Course form window.
- B. A local variable `@FacultyID` is declared, and it is used to hold the returned query result, `faculty_id`, from the execution of the first query to the Faculty table in 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`.
- D. The second query is 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.

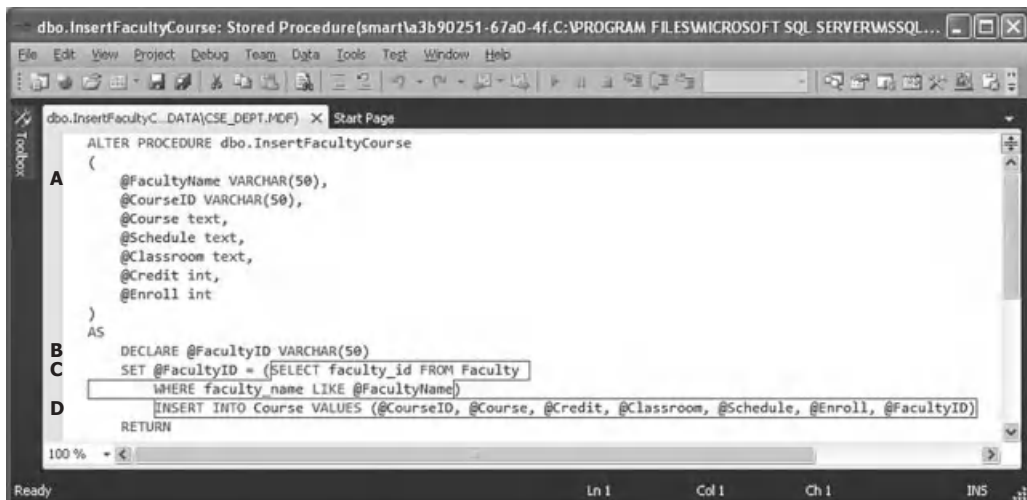


Figure 6.45. The codes for the stored procedure `dbo.InsertFacultyCourse`.

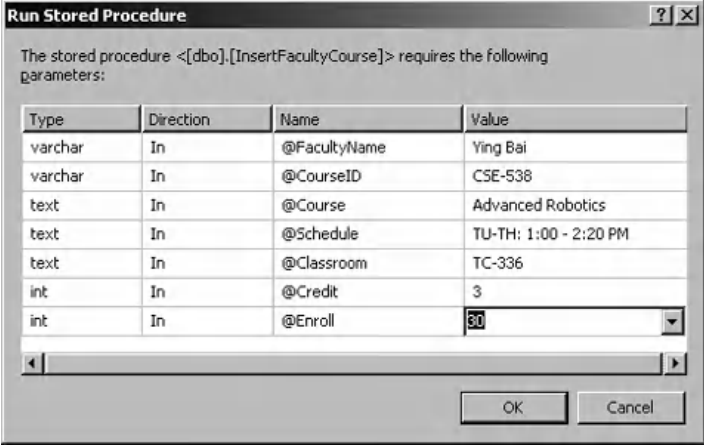
The coding process for this stored procedure is simple and easy to be understood. One point you should know is the order of parameters in the **VALUES** parameter list. This order must be identical with the column order in the **Course** table. Otherwise, an error may be encountered when this stored procedure is executed.

Go to the **File|Save 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 newly 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 wizard. Enter the input parameters into the associated box for a new course record, and your finished parameters wizard is shown in Figure 6.46.

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 6.47.

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 select the item **Show Table Data**. Browse this table to 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!



The stored procedure <[dbo].[InsertFacultyCourse]> requires the following parameters:

Type	Direction	Name	Value
varchar	In	@FacultyName	Ying Bai
varchar	In	@CourseID	CSE-538
text	In	@Course	Advanced Robotics
text	In	@Schedule	TU-TH: 1:00 - 2:20 PM
text	In	@Classroom	TC-336
int	In	@Credit	3
int	In	@Enroll	30

OK Cancel

Figure 6.46. The Run Stored Procedure wizard.

```

Output

Running [dbo].[InsertFacultyCourse] ( @FacultyName = Ying Bai, @CourseID =
CSE-538, @Course = Advanced Robotics, @Schedule = TU-TH: 1:00 - 2:20 PM,
@classroom = TC-336, @Credit = 3, @Enroll = 30 ).

(1 row(s) affected)
(0 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[InsertFacultyCourse].

```

Figure 6.47. The running result of the stored procedure.

Next, we need to develop the codes in Visual Basic.NET environment to call this stored procedure to insert a new course record into the database from our user interface.

6.8.1.2 Develop Codes to Call Stored Procedures to Insert Data into the Course Table

The coding process 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 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 can be read back and displayed in the form window. Let's begin with the coding process for the first step now.

6.8.1.2.1 Validate Data Before the Data Insertion Two user-defined procedures, `InitCourseInfo()` and `CheckCourseInfo()`, are developed in this part to perform the data validation before the data insertion action. Open the code window of the Course form and enter the codes shown in Figure 6.48 into this window to create a user-defined subroutine procedure `InitCourseInfo()` and a user-defined function `CheckCourseInfo()`.

Let's take a look at the following pieces of codes to see how they work.

- A. A For loop is used to create a new textbox array.
- B. The content of each textbox is assigned to the Text property of the associated textbox in that textbox array initialized in step A.

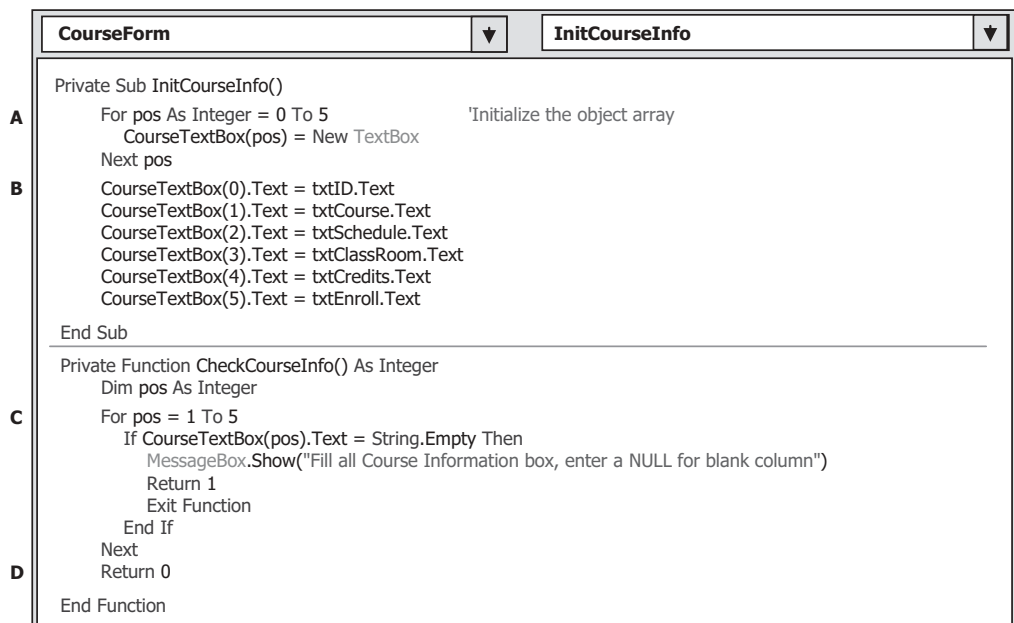


Figure 6.48. The codes for user-defined subroutine and function.

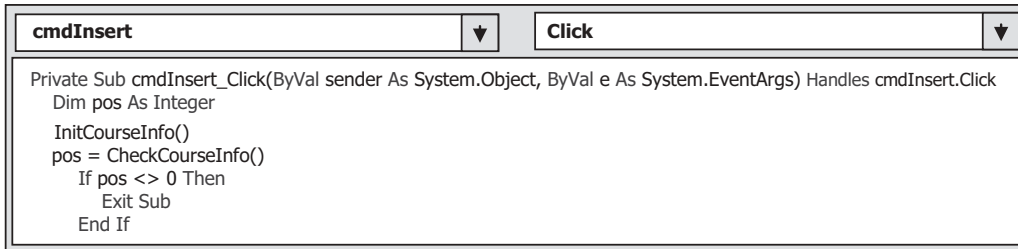


Figure 6.49. The first coding part for the Insert button's event procedure.

- C. To check each textbox, a For loop is utilized to scan the CourseTextBox array. A warning message would be displayed, and the function returns a nonzero value to the calling procedure to indicate that this checking is failed if any textbox is empty.
- D. Otherwise, a zero is returned to indicate that this checking is successful.

Now let's do our coding process for the data validation before the data insertion.

This data validation can be performed by calling one subroutine `InitCourseInfo()` and one function `CheckCourseInfo()`, which we have discussed above, in the Insert button's Click event procedure. Open the Insert button's Click event procedure and enter the codes that are shown in Figure 6.49 into this event procedure.

The function of this piece of codes is straightforward and easy to be understood. First, the subroutine `InitCourseInfo()` is called to set up one-to-one relationship between each item in the `CourseTextBox()` array and each associated textbox that stores a piece of course information. Next, the function `CheckCourseInfo()` is executed to make sure that the new course information is completed and valid; in other words, no textbox is empty.

Now let's develop and complete the codes to call the stored procedure to perform the new course data insertion.

6.8.1.2.2 Develop Codes to Call Stored Procedures Open the Insert button's Click event procedure and add the codes that are shown in Figure 6.50 into this event procedure.

The codes we developed in the last section have been highlighted with a gray background. Let's take a look at those new added codes to see how they work.

- A. The query string is assigned with the name of the stored procedure we developed in Section 6.8.1.1 in this chapter. 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. The Visual Basic.NET project could not find the stored procedures, and a timeout error would be encountered if the query string does not match the name of the stored procedure.
- B. Some other components and variables used in this procedure are declared here. The local integer variable `intInsert` is used to hold the returned value of execution of the `ExecuteNonQuery()` method. The SQL Command object `sqlCommand` is created here, too.
- C. 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

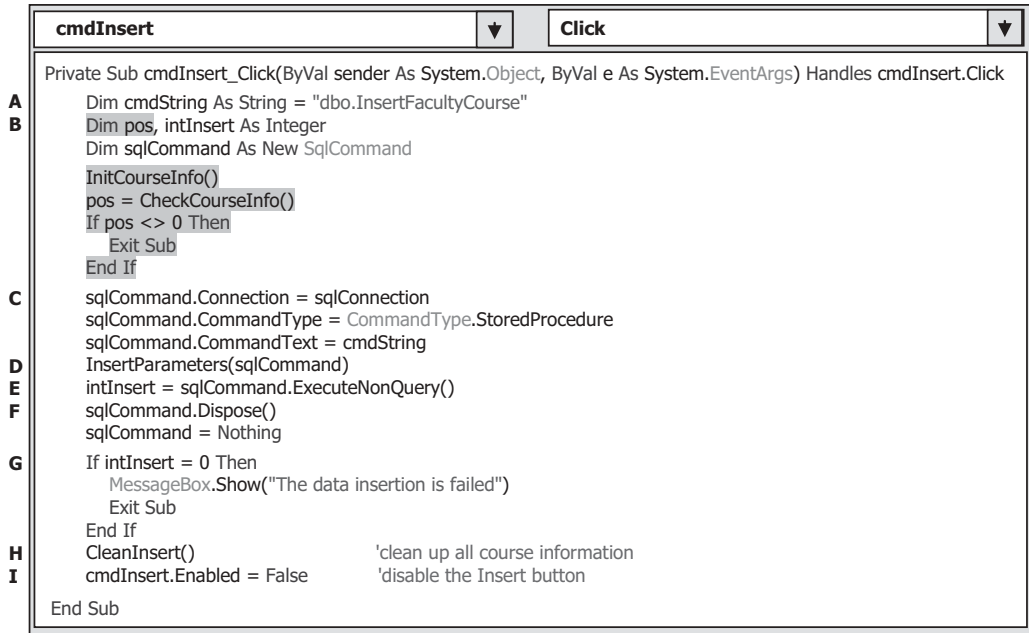


Figure 6.50. The modifications to the Insert button's Click event procedure.

property of **StoredProcedure** to indicate that the command type of this Command object is stored procedures, and a stored procedure will be called when this Command is executed. The name of the stored procedure must be assigned to the **CommandText** property of the Command object to provide the direction for the Visual Basic.NET project.

- D.** The user-defined subroutine **InsertParameters()**, whose detailed codes are shown in Figure 6.51, is executed to fill all input parameters into the **Parameters** collection of the Command object to finish the initialization of the Command object.
- E.** The **ExecuteNonQuery()** method of the Command class is executed to call the stored procedure to perform this new course data insertion.
- F.** The Command object is cleaned up and released after this data insertion.
- G.** The **ExecuteNonQuery()** method will return an integer to indicate whether this calling is successful or not. The returned value equals to the number of rows or records that have been successfully added 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, and the procedure is exited.
- H.** After a record has been successfully inserted into the Course table, all six pieces of information stored in all textboxes, except the Course ID, are cleaned up to make it ready for the next data insertion.
- I.** Also, the Insert button is disabled to avoid multiple insertions of the same data into the database.

The detailed codes for the user-defined subroutine **InsertParameters()** are shown in Figure 6.51.

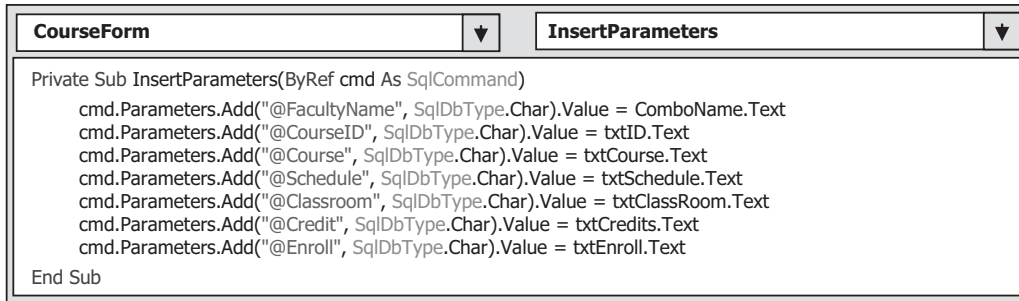


Figure 6.51. The codes for the user-defined subroutine InsertParameters().

The function of this subroutine 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 of this piece of codes 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 in here must be identical with the input parameter's name `@FacultyName` that exist in the input parameter's list we defined at the beginning of the stored procedure `dbo.InsertFacultyCourse`. A runtime 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 6.45 for a detailed list of all parameters' names defined in the stored procedure.

Now we have finished the coding process for this data insertion operation. 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.

Select the default faculty member **Ying Bai** from the Faculty Name combobox and enter the following data into the associated textbox as the information for a new course:

- 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
- 32 Enrollment textbox

Your finished information window should match the one that is shown in Figure 6.52.

Click on the **Insert** button to call the stored procedure to insert this new course record into the database. Immediately, all textboxes, except the Course ID, are cleaned up, and 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.

Figure 6.52. The running status of the Course Form window.

6.8.1.2.3 Validate Data After the Data Insertion To confirm and validate this new course record insertion, we can use the **Select** button's Click event procedure to retrieve this new course record from the database and display it in this Course form.

Select the default faculty member **Ying Bai** from the Faculty Name combo box and click on the **Select** button. All courses taught by this faculty are displayed in the CourseList box. The last item, **CSE-668**, is the course we just added into the Course table in the last section. Click on that **course_id**, and all pieces of related course information are displayed in six textboxes, which is shown in Figure 6.53. This is the evidence that our data insertion using the stored procedure is successful!

A completed project **SQLInsertRTOObjectSP** that includes the data insertion using the stored procedure can be found in the folder **DBProjects\Chapter 6** located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

6.8.2 Insert Data into the 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 structure of the stored procedure. An Oracle package must be used to contain the stored procedures in the Oracle database if the stored returned needs to return any data. A typical example of using an Oracle package is that a stored procedure contains a **SELECT** statement and needs to return the query result. A normal Oracle stored procedure is needed if the query does not need to return any data, such as the Insert, Update, and Delete queries.

Figure 6.53. The data validation process.

In this section, we try to use the Course form and Course table to discuss how to insert a new course record into the Course table using the stored procedure in Oracle database environment. Because the Insert query does not need to return any data, a normal Oracle stored procedure is good enough for this application.

To illustrate how to insert data into the Oracle database using stored procedures, we will utilize the following three steps:

1. Develop an Oracle stored procedure to perform inserting data into the Oracle database
2. Develop the codes to call the stored procedure developed in step 1 to complete the data insertion function
3. Validate the data insertion using the Course form window

To save time and space, we modify the project `OracleInsertRTOObject` we developed in Section 6.7 in this chapter and create a new project named `OracleInsertRTOObjectSP`. Refer to Section 6.2.1 to get more detailed information about how to rename a current project to create a new project.

Now let's start from step 1 to develop an Oracle stored procedure.

6.8.2.1 Develop Stored Procedures in Oracle Database

A very detailed discussion about creating and manipulating Oracle packages and stored procedures is provided in Section 5.20.7 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 to insert data into the database, so no returned data is needed for this section. Therefore, we only need to create stored procedures in Oracle database, not package, to perform the data insertion functionality.

As discussed in Section 5.20.7 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 11g XE to create our stored procedures.

Open the Oracle Database 11g XE home page by going to **start|All Programs |Oracle Database 11g Express Edition|Get Started** items. Finish the APEX login process by entering the correct username and password (in our case, it is **SYSTEM** and **reback**). Log in to the APEX Workspace using the **CSE_DEPT** as the names for both Workspace and Username, and **reback** as the Password. Click on the **SQL Workshop** and then the **Object Browser** icon to open the Object Browser page. Select the **Procedures** from the list and click on the **Create** button to open the Create Database Object wizard. Click on the **Procedure** to open the Create Procedure wizard. The opened wizard is shown in Figure 6.54.

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 wizard.

The next wizard allows us to enter all input parameters. For this stored procedure, we need to perform two queries, so 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 **Faculty Name** combo box control from the **Course** form window. 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: **Course ID**, **Course Title**, **Credit**, **Classroom**, **Schedule**, **Enrollment**, and **Faculty Name**. The last input parameter

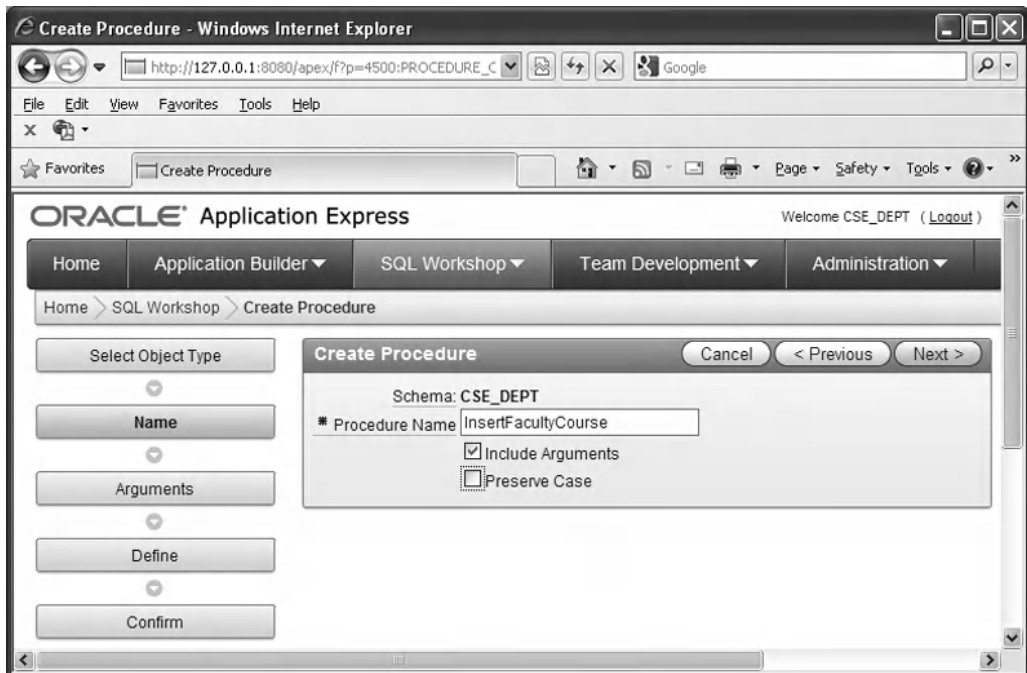


Figure 6.54. The opened Create Procedure wizard.

ORACLE® Application Express Welcome CSE_DEPT (Logout)

Home Application Builder SQL Workshop Team Development Administration

Home > SQL Workshop > Create Procedure

Select Object Type

Name

Arguments

Define

Confirm

Create Procedure Cancel < Previous Next >

Schema: CSE_DEPT
Procedure Name: INSERTFACULTYCOURSE

Argument Name	In/Out	Argument Type	Default	Move
CourseID	IN	VARCHAR2		▼
Course	IN	VARCHAR2		▼ ▲
Credit	IN	NUMBER		▼ ▲
Classroom	IN	CHAR		▼ ▲
Schedule	IN	VARCHAR2		▼ ▲
Enroll	IN	NUMBER		▼ ▲
FacultyName	IN	VARCHAR2		▼ ▲

Figure 6.55. The finished argument list.

Faculty Name is used by the first query, and the first 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 2.11.5 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 selection 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 that is shown in Figure 6.55.

Click on the Next button to go to the procedure definition page. Enter the codes, which are shown in Figure 6.56, 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 6.57.

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.

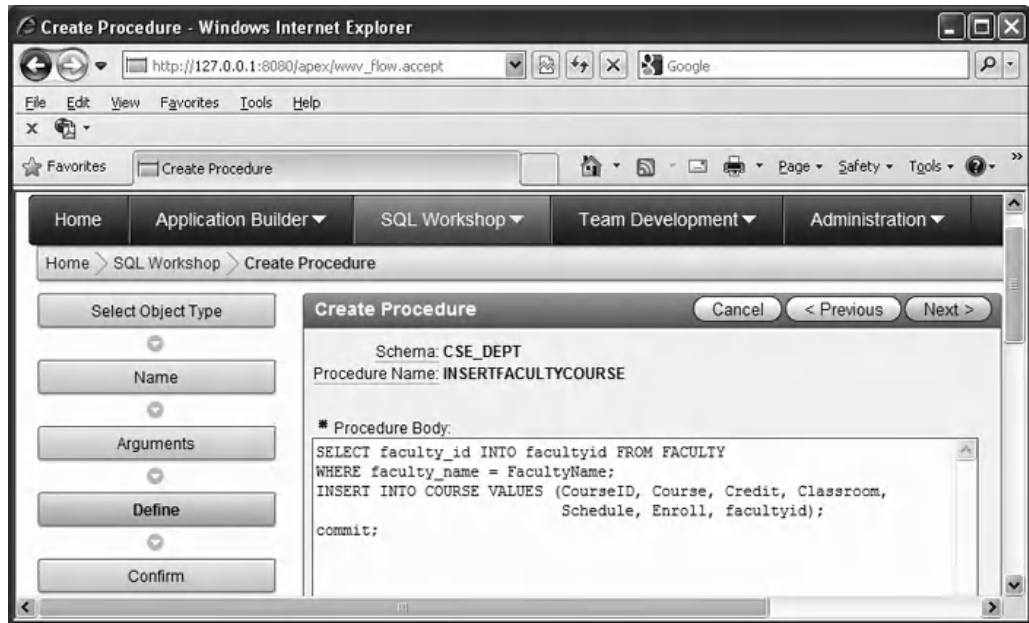


Figure 6.56. The stored procedure body.



Figure 6.57. The completed stored procedure.

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 6.57. This coding line has been highlighted with the black color. This local variable is used to hold the returned `faculty_id` from the execution of the first query.

Another important issue is for the input parameters or arguments in an `INSERT VALUES` command, which is the order of those parameters or arguments. This order 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 placed in the `INSERT VALUES` 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 **Save & 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 11g Express Edition by clicking on the **Close** button. Next, we need to develop our codes in Visual Basic.NET project to call this stored procedure to perform the data insertion function.

6.8.2.2 Develop Codes to Call Stored Procedures to Insert Data into the Course Table

Basically, the codes to be developed in this section are very similar to those we developed in Section 6.8.1.2. The function of this piece of codes is to allow users to enter seven pieces of information related to a new course into the Course table. In the following sections, we only emphasize and highlight the important and different parts of the codes for the Oracle database.

6.8.2.2.1 Validate Data Before the Data Insertion The codes to be developed in this section are identical with those we did in Section 6.8.1.2.1. You can copy the codes in Figures 6.48 and 6.49 and paste them to the code window of the Course form and the **Insert** button Click event procedure in our current project. In Section 6.8.2, we have modified the project `OracleInsertRTOObject` and created a new project `OracleInsertRTOObjectSP`. Now open this new project and the code window of the Course Form. Double-check the following two Imports commands at the top of this code window:

```
Imports Devart.Data
Imports Devart.Data.Oracle
```

Next, let's finish the codes for the **Insert** button Click event procedure to call the stored procedure to perform the course record insertion.

6.8.2.2.2 Develop Codes to Call Stored Procedures The main coding job to call the stored procedure is made inside the **Insert** button's Click event procedure in the

Course Form window. The codes for this event procedure are very similar to those we did for the same event procedure in the last project `SQLInsertRTOObjectSP`. Open that project and that event procedure, and copy the codes from that procedure and paste them into our `Insert` button's Click event procedure. Also, copy the user-defined subroutine procedures `InsertParameters()` and `CleanInsert()`, and paste them into our current Course code window.

Some code modifications are made for the `Insert` button Click event procedure and two user-defined subroutine procedures to make them work for the Oracle database. The modified codes in the `Insert` button Click event procedure have been highlighted in bold and shown in Figure 6.58. Let's give a detailed discussion for those modifications one-by-one based on the steps defined in Figure 6.58.

First, let's concentrate on the modifications for the `Insert` button's event procedure.

- A. Change the content of the query string, which is 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 11g XE in Section 6.8.2.1.
- B. Change the prefix before the Oracle classes and objects to `Oracle` and `ora`.
- C. Change the prefix before all Oracle objects from `sql` to `ora`. Steps involved in this change are from C to I.

Now, let's take care of the modifications to the first user-defined subroutine procedure `InsertParameters()`. All modifications have been highlighted in bold and shown in Figure 6.59.

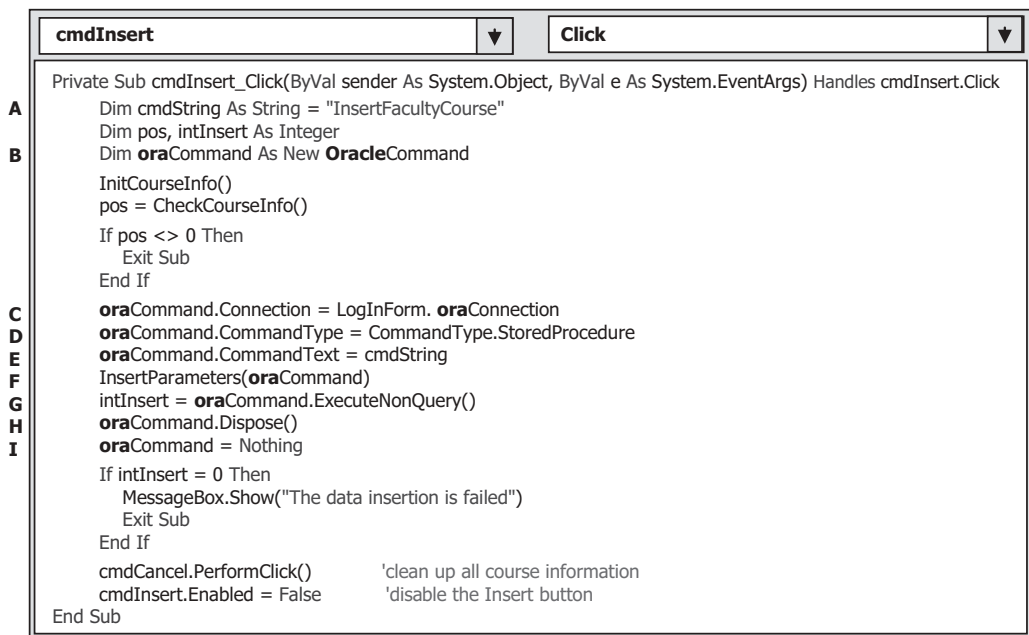


Figure 6.58. Modified codes for the `Insert` button's event procedure.

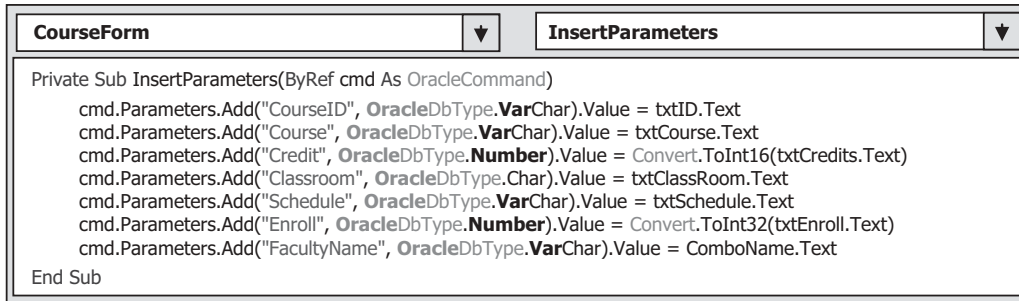


Figure 6.59. Modifications to the subroutine InsertParameters().

The function of this subroutine is to assign each piece of information stored in each textbox to the associated input parameter we defined in the Oracle's stored procedure InsertFacultyCourse. One key point is that the order of these seven input parameters, which is represented from the top to the bottom, must be identical with the order of the input parameters we defined in the stored procedure. For example, the order of the input parameters in the stored procedure is: CourseID, Course, Credit, Classroom, Schedule, Enroll, and FacultyName. The order to assign those parameters in the user-defined subroutine procedure InsertParameters() (refer to Figure 6.59) must be identical with the order listed above. A runtime error would be encountered if the order of those parameters is not matched with the order of those input parameters defined in the stored procedure as the project runs. Refer to Figure 6.55 for the order of the seven input parameters defined in the stored procedure.

The codes in the user-defined subroutine CleanInsert() does not need any modification.

Now, we have finished the coding process for this data insertion operation. 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.

Keep the default faculty member Ying Bai selected from the Faculty Name combo box and enter the following data into the associated textbox as the information for a new course:

- 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 that is shown in Figure 6.60.

Click on the Insert button to call the stored procedure to insert this new course record into the database. Immediately, all six textboxes become empty, and the Insert

Figure 6.60. The running status of the Course Form window.

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.

6.8.2.2.3 Validate Data After the Data Insertion To confirm and validate this data insertion, we can use the **Select** button Click event procedure to retrieve this newly inserted course record and display it in the Course form window.

Keep the faculty member **Ying Bai** selected from the Faculty Name combo box and click on the **Select** button to try to retrieve the newly inserted course record and display it in this Course form window. All courses taught by the selected faculty are displayed in the CourseList box. The last item is the course we just added into the Course table in the last section. Click on that `course_id`, and all six pieces of information related to that new `course_id` are displayed in this form, as shown in Figure 6.61. This is the evidence that our data insertion using the stored procedure is successful!

Another way to confirm this data insertion is to open our sample Oracle database **CSE_DEPT** and the **Course** table. A completed project **OracleInsertRTOObjectSP** that includes the data insertion using the Oracle stored procedure can be found in the folder **DBProjects\Chapter 6** located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

6.9 INSERT DATA INTO THE DATABASE USING THE LINQ TO DATASET METHOD

As we discussed in Chapter 4, Language Integrated Query (LINQ) is a powerful method provided by Visual Studio.NET and the .NET Framework that bridges the gap between the world of objects and the world of the data. In Visual Studio.NET, you can write LINQ queries in Visual Basic.NET with SQL Server databases, XML documents, ADO.NET

CSE DEPT Course Form

Faculty Name ,Query Method

Faculty Name Ying Bai

Query Method TableAdapter Method

Course List

- CSE-668
- CSE-566
- CSC-132B
- CSE-434
- CSE-438
- CSC-234A

Course Information

Course ID CSE-668

Course Modern Controls

Schedule M-W-F: 9:00-9:55 AM

Classroom TC-309

Credits 3

Enrollment 30

Select **Insert** **Update** **Delete** **Back**

Figure 6.61. The data validation process.

DataSets, and any collection of objects that supports `IEnumerable` or the generic `IEnumerable(Of 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 in Chapter 4, 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(Of T)` or `IQueryable(Of 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 Oracle model available, therefore, we will concentrate our discussion on inserting data into the SQL Server database using the LINQ to SQL model.

Generally, the popular method of inserting a new record into the database using the LINQ query follows 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 the SQL Server Database Using the LINQ to SQL Queries

As we discussed in Section 4.6 in Chapter 4, 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 in 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 in our sample database CSE_DEPT.

6.10 CHAPTER SUMMARY

Five popular data insertion methods are discussed and analyzed with three different databases—Microsoft Access, SQL Server, and Oracle in this chapter:

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 is to try 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.

Method 2 allows users to insert new data into a database with two steps. First, the new record can be added into the data table that is 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 object 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 powerful technique coming with .NET Framework, Visual Studio.NET, 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 with 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 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 object method to improve the efficiency of the data insertion and provides more flexibility in data insertion.

Seven real projects are provided in this chapter to give readers a clear and direct picture in developing professional data insertion applications in Visual Basic.NET environment.

HOMEWORK

I. True/False Selections

- ___ 1. Three popular data insertion methods are: the TableAdapter.Insert(), TableAdapter.Update(), and ExecuteNonQuery() method 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. The difference between the Visual Basic collection class and the .NET Framework collection class is that the two collections start with different indexes: the former starts from 1, but the latter starts from 0.
- ___ 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. Either of above
4. Because 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 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 6.62 shows a stored procedure developed in the SQL Server database. Please develop a piece of codes in Visual Basic.NET to call this stored procedure to insert a new data into the database.

```
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 6.62. The SQL Server stored procedure.

```
Dim cmdString As String = "InsertCourse"
Dim intInsert As Integer
Dim oraCommand As 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 6.63. The codes to call the SQL Server stored procedure.

2. Figure 6.63 shows a piece of codes developed in Visual Basic.NET, and this coding is used to call a stored procedure in Oracle database to insert a new record into the database. Please create the associated stored procedure in Oracle database using the PL-SQL language.
3. Using the tools and wizards provided by Visual Basic.NET and ADO.NET to perform the data insertion for the Student form in the `InsertWizard` project. The project file can be found in the folder `DBProjects\Chapter 6` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).
4. Using the Runtime objects to complete the insert data query for the Student form by using the project `AccessInsertRTOject`. The project file can be found in the folder `DBProjects\Chapter 6` that is located at the Wiley ftp site (refer to Figure 1.2 in 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 `OracleInsertRTOjectSP`. The project file can be found in the folder `DBProjects\Chapter 6` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Chapter 7

Data Updating and Deleting with Visual Basic.NET

In this chapter, we will discuss how to update and delete data against the databases. Basically, many different methods are provided and supported by Visual Basic.NET and .NET Framework to help users to perform the data updating and deleting against the database. Among them, three popular methods are widely implemented:

1. Using `TableAdapter.DBDirect` methods, such as `TableAdapter.Update()` and `TableAdapter.Delete()`, to update and delete data directly against the databases.
2. Using `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. Using the runtime object method to develop and execute the `Command`'s method `ExecuteNonQuery()` to update or delete data against 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 Basic.NET applications. The difference between method 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, but method 2 needs two steps to perform the data updating or deleting. First, the data updating or deleting is performed to the associated tables in the `DataSet`, and then the 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 provided by the Visual Studio.NET design tools and wizards. Part II presents the data updating and deleting using the runtime object method to develop command objects to execute the `ExecuteNonQuery()` method dynamically. Updating and deleting data using the stored procedures and the LINQ to SQL query are also discussed in that part.

When finished this chapter, you will

- Understand the working principle and structure on updating and deleting data against the database using the Visual Studio.NET Design Tools and Wizards
- Understand the procedures in how to configure the TableAdapter object by using the TableAdapter Query Configuration Wizard and build the query to update and delete data against the database
- Design and develop special procedures to validate data before and after the data updating and deleting
- Understand the working principle and structure on updating and deleting data against the database using the runtime object method
- Design and build LINQ to SQL query to update and delete data
- Design and build stored procedures to perform the data updating and deleting

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, and LINQ techniques discussed in Chapter 4. Also, a sample database, CSE_DEPT, that was developed in Chapter 2 will be used throughout this Chapter.

Two kinds of databases will be used in the example projects to illustrate how to perform the data updating and deleting in this chapter. They are: SQL Server 2008 and Oracle Database 11g XE databases.

In order to save time and avoid repeatability, we will use some sample projects, such as InsertWizard, InsertWizardOracle, SQLInsertRTOBJECT, and OracleInsertRTOBJECT, we developed in the last chapter and modify them to create new associated projects to be used 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 event procedures 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 against 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 discuss 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 the data updating or deleting. The database we try to use is the Microsoft SQL Server 2008 database, CSE_DEPT.mdf, which was developed in Chapter 2, and it can be found in the folder Database\SQLServer located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). You can try to use any other databases, such as Microsoft Access 2007 or Oracle Database 11g XE. The only

issue is that you need to select and connect to the correct database with your applications when you use the Data Source to set up your data source for your Visual Basic.NET data-driven applications.

7.1 UPDATE OR DELETE DATA AGAINST DATABASES

We have already provided a very detailed discussion about the TableAdapter DBDirect methods in Section 6.1.1 in Chapter 6. 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. There are some limitations that exist when these DBDirect methods are utilized. For example, each TableAdapter is associated with a unique data table in the DataSet; therefore, the data updating or deleting can only be executed for that 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 that is related to the created TableAdapter.

7.1.1 Updating and Deleting Data from Related Tables in a DataSet

When updating or deleting data against 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 against 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 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. First, that record should be removed or deleted from the child tables, LogIn and Course tables, respectively
2. Then that record can be updated or deleted from 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 try to update a completed record (includes updating the primary key) for an existing data in our sample database, and 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 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 have been set in a **Cascade Delete** mode when we built these data tables for our sample database CSE_DEPT.mdf in Section 2.10.4 in Chapter 2.

**7.1.2 Update or Delete Data Against Database Using
TableAdapter DBDirect Methods: TableAdapter.Update
and TableAdapter.Delete**

Three typical TableAdapter’s DBDirect methods are listed in Table 6.1 in Chapter 6. For your convenience, we redraw that table in this section again, which is shown in 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 be noted 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 against the database.

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 dataset 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.

7.1.3 Update or Delete Data Against 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 against the database, one needs to perform the following two steps:

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 `DataRow`s, 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.

Table 7.2. Variations of `TableAdapter.Update()` method

Update Method	Description
<code>TableAdapter.Update(DataTable)</code>	Attempt 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 dataset, 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>	Attempt to save changes in the indicated <code>DataRow</code> to the database.
<code>TableAdapter.Update(DataRows())</code>	Attempt 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.

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 how to use these two methods to update or delete records against the database, a real example will be very helpful. Let's first create a new Visual Basic.NET project to handle these issues.

7.2 UPDATE AND DELETE DATA FOR MICROSOFT SQL SERVER 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 those staff to develop our data-updating and deleting sample project based on the InsertWizard 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 have been performed in the previous chapter.

7.2.1 Create a New Project Based on the InsertWizard Project

Open the Windows Explorer and create a new folder, such as Chapter 7, and then browse to our project InsertWizard that was developed in the last chapter and can be found in the folder DBProjects\Chapter 6 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy this project to our new folder Chapter 7. Change the name of the solution and the project folder from InsertWizard to SQLUpdateDeleteWizard. Also, change the project's name from InsertWizard.vbproj to SQLUpdateDeleteWizard.vbproj. Then double-click on the SQLUpdateDeleteWizard.vbproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Select any form window, such as LogIn Form.vb, from the Solution Explorer window. Then go to the Project!SQLUpdateDeleteWizard Properties menu item to open the project's property window. Change the Assembly name from InsertWizard to SQLUpdateDeleteWizard and the Root namespace from InsertWizard to SQLUpdateDeleteWizard, respectively.
- Click on the Assembly Information button to open the Assembly Information wizard. Change the Title and the Product to SQLUpdateDeleteWizard. Click on the OK button to close this wizard.

Go to File!Save All to save those modifications. Now we are ready to develop our graphic user interfaces based on this new project.

7.2.2 Application User Interfaces

Recall that when we developed the project `InsertWizard`, there are 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, exactly these two buttons' event procedures, to perform the data updating and deleting actions against 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 window as the user interfaces to collect the new records to perform those updating and deleting operations. Instead, we can use the Faculty form with some codes modifications to perform these two data actions.

7.2.3 Validate Data before the Data Updating and Deleting

This data validation can be neglected because when we performed a data query by clicking on the **Select** button, the retrieved data should be a complete set of data and can be displayed in the Faculty form window. This means that all textboxes have been filled by the related faculty information and no one is empty, no matter if we do some modifications or not, all textboxes are full. So this data validation before the data updating or deleting can be avoided.

7.2.4 Build the 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.

Now let's build our data updating and deleting queries using the `TableAdapter Query Configuration Wizard` and `Query Builder`.

7.2.4.1 Configure the TableAdapter and Build the Data Updating Query

Open the Data Source window by going to the `Data\Show Data Sources` menu item. Perform the following operations to build the data updating query:

1. On the opened wizard, click on the **Edit the DataSet with Designer** button that is located at the second left on the toolbar in the Data Source window to open this Designer.
2. 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`.
3. Keep the default selection **Use SQL statements** unchanged and click on the **Next** button to go to the next wizard.
4. Select and check the **UPDATE** item from this wizard since we need to perform a data updating query, and then click on the **Next** button again to continue.
5. Click on the **Query Builder** button since we want to build our updating query. The opened `Query Builder` wizard is shown in Figure 7.1.

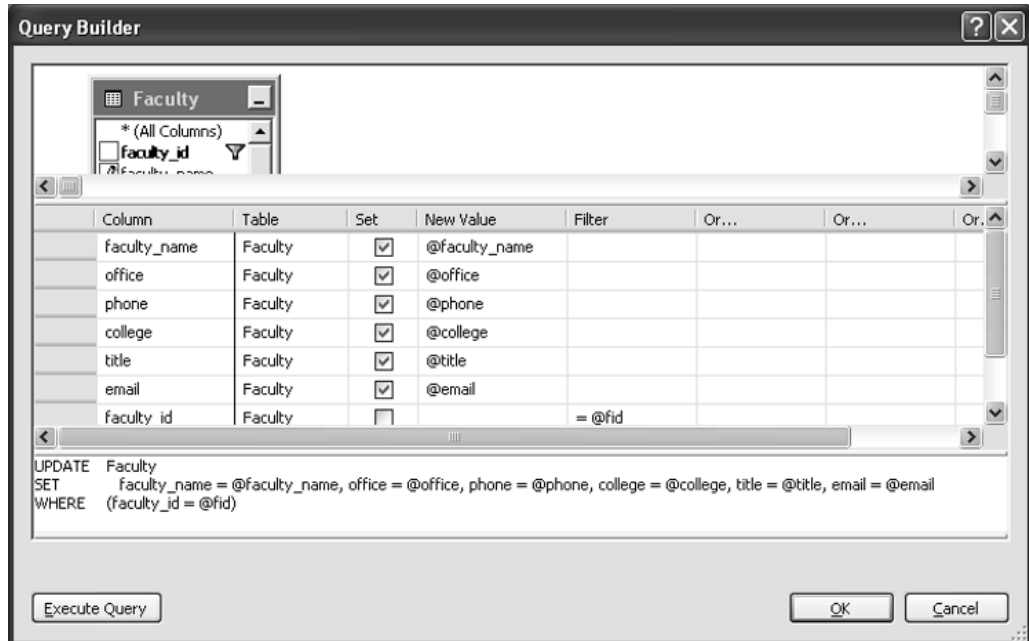


Figure 7.1. The Query Builder for the Update query.

6. Remove all contents from the `faculty_id` row under the Filter and Or columns. Uncheck the Set checkbox from the `faculty_id` row under the column Set, and enter a question mark (?) to the `faculty_id` row under the column Filter, and press the Enter key from the keyboard. Change the name of the dynamic parameter `@Param1` to `@fid` for the Filter column in the `faculty_id` row.
7. Remove all rows under the last row—email—and your finished query builder wizard should match the one that is shown in Figure 7.1.
8. Click on the OK button to go to the next wizard. Remove the SELECT statement under the WHERE clause since we do not need this function. Click on the Next to confirm this query and continue to the next step.
9. Modify the query function name from the default one to the `UpdateFaculty` and click on the Next button to go to the last wizard.
10. Click on the Finish button to complete this query building and close the wizard. Immediately, you can find that a new query function `UpdateFaculty` has been added into the Faculty TableAdapter as the last item.

Now let's continue to build our Delete query function using the Query Builder.

7.2.4.2 Build the Data Deleting Query

Reopen the Edit DataSet with Designer wizard and right-click on the last item from the Faculty table and select the Add Query item to open the TableAdapter Query

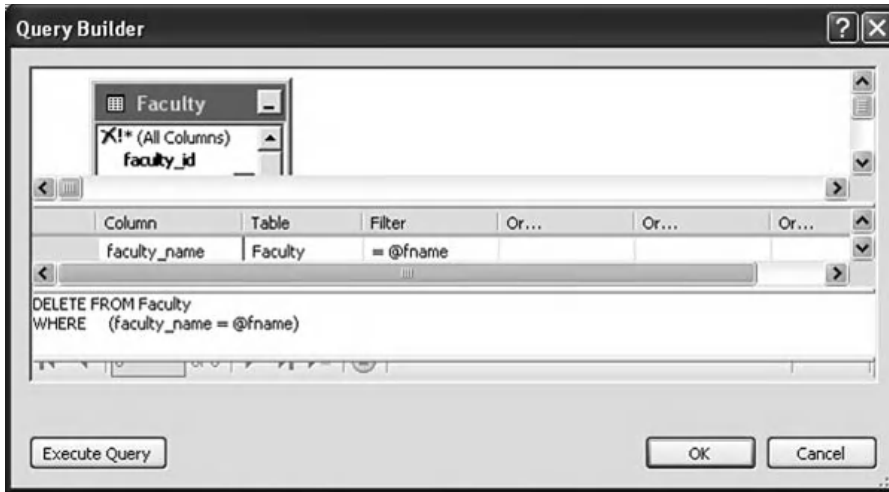


Figure 7.2. The Query Builder for the Delete query.

Configuration Wizard if it is not opened. Perform the following operations to build the data-deleting query:

1. On the opened wizard, keep the default selection **Use SQL statements** unchanged and click on the **Next** button to go to the next wizard.
2. Select and check the **DELETE** item from this wizard since we need to perform a data deleting query. Then click on the **Next** button again to continue.
3. Click on the **Query Builder** button since we want to build our deleting query. The opened Query Builder wizard is shown in Figure 7.2.
4. Delete the whole line of the row **faculty_id** and the row **@IsNull_faculty_name**. Remove the contents of the columns **Filter** and **Or** for the **faculty_name** row.
5. Enter a question mark (?) into the **Filter** column along the **faculty_name** row and press the **Enter** key on the keyboard.
6. Change the name of the dynamic parameter **@Param1** to **@fname**, and press the **Enter** key from the keyboard.
7. Your finished query builder should match the one that is shown in Figure 7.2. Click on the **OK** button to go to the next wizard.
8. Click on the **Next** button to confirm this query and continue to the next step.
9. Modify the query function name to **DeleteFaculty** and click on the **Next** button to go to the last wizard.
10. Click on the **Finish** button to complete this query building and close the wizard. Immediately, you can find that a new query function **DeleteFaculty** has been added into the **Faculty TableAdapter** as the last item.

Next, let's develop the codes to call these built query methods to perform the related data actions.

7.2.5 Develop Codes to Update Data Using the TableAdapter DBDirect Method

To perform the data updating using the built query method, some modifications to the original codes in the Faculty form are necessary. We divided these modifications into two subsections: codes modifications and codes creations.

7.2.5.1 Modifications of the Codes

The first modification is to modify the codes inside the Form_Load() event procedure in the Faculty Form class, that is, to add two new updating methods into the Query Method combo box:

1. TableAdapter DBDirect Method
2. TableAdapter.Update Method

To do that, open this method and add these two methods into the Form_Load() event procedure using the Add() method. Your modified codes for this procedure are shown in steps **A** and **B** in Figure 7.3. The newly added codes have been highlighted in bold.

7.2.5.2 Creations of the Codes

The main coding process to perform this data updating is developed inside the Update button's Click event procedure. Open this event procedure and enter the codes that are shown in Figure 7.4 into this event procedure.

Let's have a closer look at this piece of new added codes to see how it works.

- A.** All objects and variables used in this event procedure are declared here first. An instance of the FacultyTableAdapter class is created first since we need to use it to perform the data

(FacultyForm Events)▼

Load▼

```
Private Sub FacultyForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    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")
    ComboMethod.Items.Add("TableAdapter Insert")
    ComboMethod.Items.Add("TableAdapter Update")
ComboMethod.Items.Add("TableAdapter DBDirect Method")
ComboMethod.Items.Add("TableAdapter.Update Method")
    ComboMethod.SelectedIndex = 0
End Sub
```

Figure 7.3. The modified codes for the Form_Load event procedure.

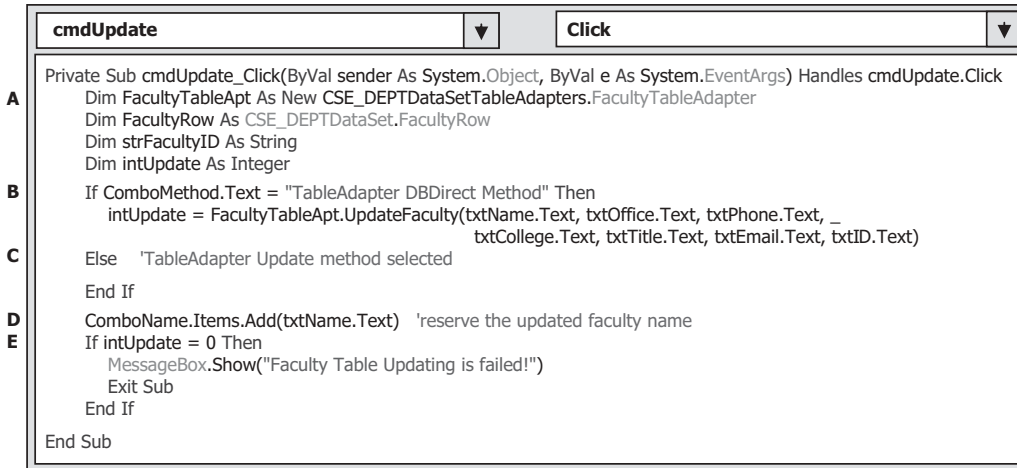


Figure 7.4. The codes for the Update command button's Click event procedure.

updating. A new row object of FacultyRow class is also created since we need this object to update the data in the DataSet to the table in the database later when we use another 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 database. The local String variable strFacultyID is used to hold the returned faculty_id value when executing the second method, TableAdapter.Update(), to update the database 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 an integer to indicate whether this function calling is successful or not. The value returned is equal to the number of rows that have been successfully update.
- C.** If the user selected the second method, TableAdapter.Update(), to update this record, the related codes that will be developed later are executed to first update that record in the DataSet, and then update it to the database.
- D.** For the validation purpose, we need to reserve this updated faculty name and save it into the Faculty Name combo box.
- E.** If the returned value of executing the updating function is equal to zero, which means that no row or record has been updated after calling that query function, a warning message is displayed and the procedure is exited.

Now let's continue to develop the codes for the second data updating method.

7.2.6 Develop Codes to Update Data Using the TableAdapter.Update Method

Open the Update button's Click event procedure if it is not opened and add the codes that are shown in Figure 7.5 into this event procedure. Let's take a closer look at this

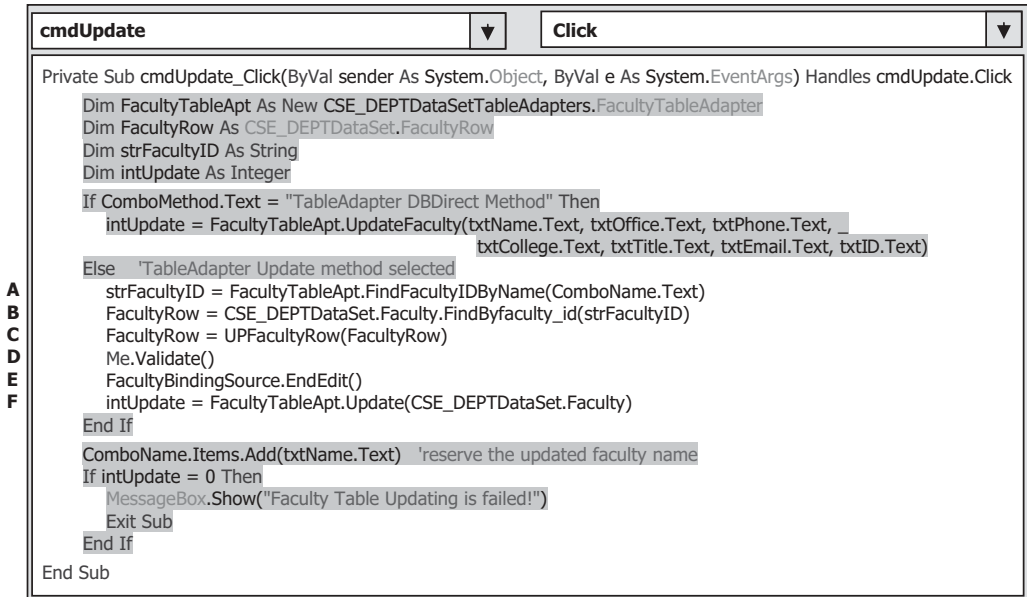


Figure 7.5. The codes for the second data updating method.

piece of newly added codes to see how it works. The codes we developed in the previous steps are highlighted with a gray background.

- 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, which is defined as `FindBy()`, to do that. However, 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 the `faculty_id`, we can use a query function `FindFacultyIDByName()` we built in Section 5.14.1 in Chapter 5 with the Faculty Name as a query criterion. One point to be noted to run this function is that the parameter Faculty Name must be an old faculty name because in order to update a faculty row, we must first find the old faculty row based on the old name. So the content in the combo box Faculty Name, `ComboName.Text`, is used as the old faculty name.
- B. After the `faculty_id` is found, the default 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 function `UPFacultyRow()` is called to assign all pieces of updated faculty information to the desired rows. 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, 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 controls 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

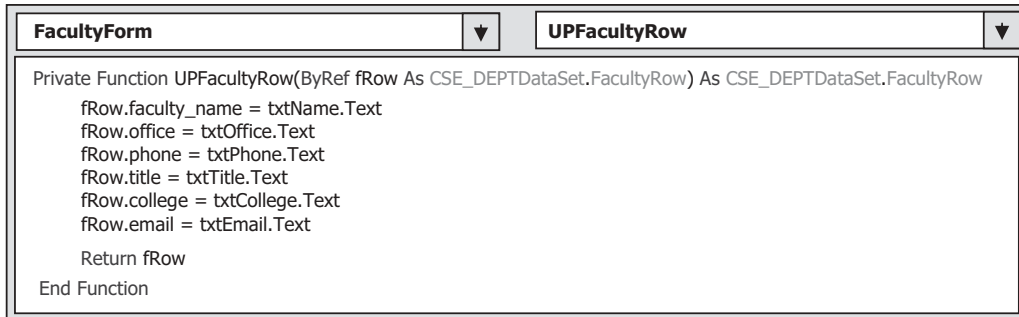


Figure 7.6. The codes for the user-defined function UPFacultyRow().

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 function UPFacultyRow() are shown in Figure 7.6. The functionality of this function is straightforward and easy understand.

The argument of this function is a DataRow object, and it is passed by a reference to the function. The advantage of passing an argument in this way is that any modifications performed to DataRow object inside the function 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.

7.2.7 Develop Codes to Delete Data Using the 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 object method, such as ExecuteNonQuery to create command object to delete data from a database.

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.

Open the Faculty form window and double-click on the **Delete** button to open its event procedure. Enter the codes that are shown in Figure 7.7 into this event procedure.

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

- A.** All data components and objects, as well as variables used in this event procedure, are declared and created here. The object of the FacultyTableAdapter class is created first since we need to use its Update() and Delete() methods to delete data later. A button object of

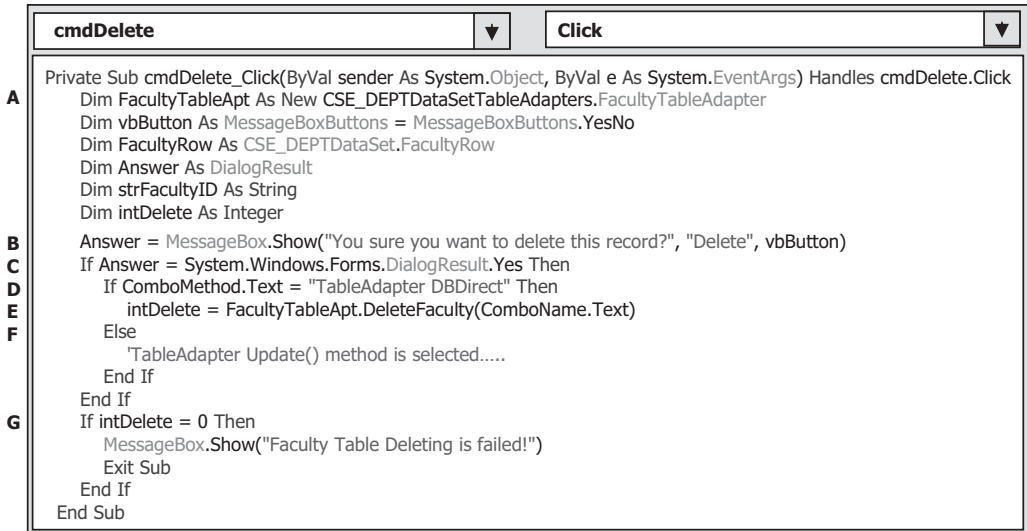


Figure 7.7. The codes for the Delete button's Click event procedure.

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 also used for the second deleting method. The local variable **Answer** is an instance of **DialogResult** class, and it is used to hold the returned value of calling the **MessageBox** function. This variable can be replaced by an integer variable if you like.

- B.** First, a **MessageBox** function 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** function 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.
- G.** The returned value of calling either the **TableAdapter.Delete()** method or the **TableAdapter.Update()** method is an integer, 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 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 develop the codes for the data deleting using the second method.

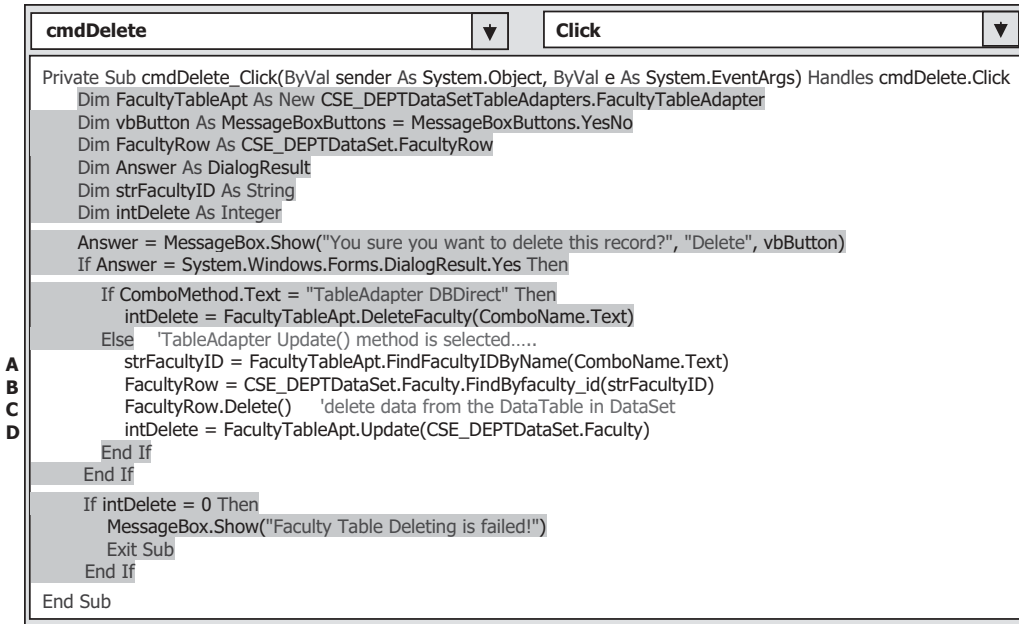


Figure 7.8. The codes for the second data deleting method.

7.2.8 Develop Codes to Delete Data Using the TableAdapter.Update Method

Add the codes that are shown in Figure 7.8 into the Delete button's Click event procedure, exactly into the Else block for the second deleting method. The codes we developed in the previous steps have been highlighted with a gray background. Let's have a close look at this piece of newly added codes to see how it works.

- A.** To identify the DataRow to be deleted from the DataTable, the user-built query method FindFacultyIDByName() is utilized. Since that method needs to use the faculty_id as the query qualification, therefore, we need first 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 that deleted DataRow to the data row in the database.

Before we can run the project to test our codes for the data updating and deleting, let's first complete the codes for the data validations for those data actions.

7.2.9 Validate the Data after the Data Updating and Deleting

As we mentioned in the previous section, we do not need to develop any code for these data validations since we can use the codes we developed for the **Select** button's Click event procedure to perform these validations.

Now let's run the project to test our codes for the data updating and data deleting. Make sure that the default faculty photo file **Default.jpg** has been stored in the default location—in our case, it is in the folder in which our Visual Basic.NET executable file is located (**C:\Chapter 7\SQLUpdateDeleteWizard\bin\Debug**). 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. Keep the default faculty member **Ying Bai** in the **Faculty Name** combo box selected and click on the **Select** button to query the detailed information for this faculty.

To update this faculty record, you can use either the **TableAdapter DBDirect** or the **TableAdapter.Update** method as you like by selecting it from the **Query Method** combo box. Keep the **Faculty ID** unchanged and then enter the following information to the associated textboxes as an updated **Faculty** record:

- **Susan Bai** Faculty Name textbox
- **Professor** Title textbox
- **MTC-255** Office textbox
- **750-378-1155** Phone textbox
- **Duke University** College textbox
- **sbai@college.edu** Email textbox
- **Default.jpg** Faculty Image textbox

Your finished new faculty information window should match the one that is shown in Figure 7.9.

Click on the **Update** button to try to update this faculty record in the **Faculty** table. To confirm this data updating action, click on the drop-down arrow of the **Faculty Name** combo box, and you will find that the updated faculty name **Susan Bai** is in there. Select this name and click on the **Select** button to retrieve this updated faculty record and display it in the associated textboxes in this form.

You can find that the original faculty record is indeed updated, which is shown in Figure 7.10.

To delete this faculty record, select either the **TableAdapter DBDirect** or the **TableAdapter.Update** method from the **Query Method** combo box. Then click on the **Delete** button. 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 that faculty record has been deleted from the database.

CSE DEPT Faculty Form

Faculty Image
Default.jpg

Faculty Name Ying Bai
Query Method TableAdapter.Update

Faculty Information

Faculty ID	B78880
Name	Susan Bai
Title	Professor
Office	MTC-255
Phone	750-378-1155
College	Duke University
Email	sbai@college.edu

Select **Insert** **Update** **Delete** **Back**

Figure 7.9. The running status of the Updated Faculty Form window.

CSE DEPT Faculty Form

Faculty Image
Default.jpg

Faculty Name Susan Bai
Query Method TableAdapter Method

Faculty Information

Faculty ID	B78880
Name	Susan Bai
Title	Professor
Office	MTC-255
Phone	750-378-1155
College	Duke University
Email	sbai@college.edu

Select **Insert** **Update** **Delete** **Back**

Figure 7.10. The updated faculty record.

One point to be noted is that after you update the faculty name by changing the content of the Faculty Name textbox, be sure that you go to the Faculty Name combo box to select the modified faculty name to perform the data validation. 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 different with the content of the Faculty Name combo box, and the former is an updated faculty name and the latter is an old faculty name if the faculty name is updated.

Our project is very successful!

It is highly recommended to recover the deleted faculty member since we want to keep our database neat and complete.

An import issue for this data recovery is the order of recovering these deleted records. Figure 7.11 shows the relationships between the Faculty table and other tables in our sample database CSE_DEPT. Based on this relationship, you should

- First recover the records in the parent table (Faculty and Course tables),
- Then recover the records in the child tables (LogIn and StudentCourse tables).

Follow the table order in Figure 7.11 and refer to Tables 7.3–7.6 to complete these records’ recovery.

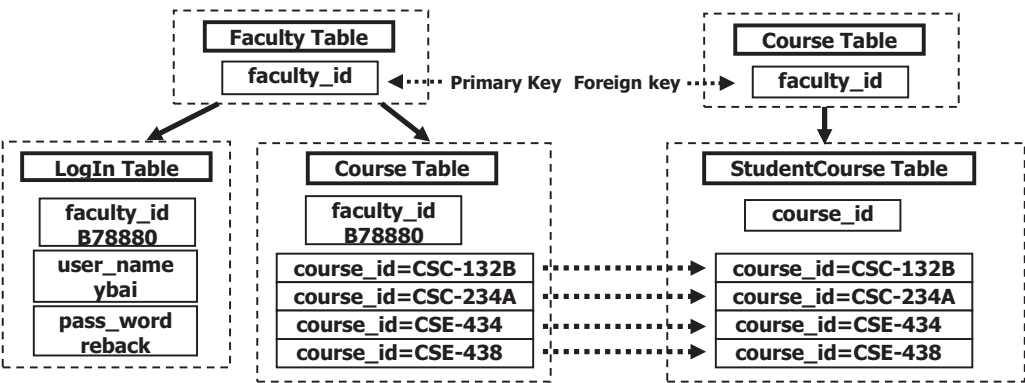


Figure 7.11. The relationships among tables.

Table 7.3. The 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. The data to be recovered in the LogIn table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	NULL

Table 7.5. The 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. The 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

You can access the sample database via the Server Explorer window to perform this recovery process.

A complete project **SQLUpdateDeleteWizard** can be found in the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Next, let's discuss how to update and delete data using the Visual Studio.NET tools and wizards for Oracle database.

7.3 UPDATE AND DELETE DATA FOR ORACLE DATABASE

It is very similar to develop a Visual Basic.NET project to modify data against the Oracle database using the Update and Delete button's Click event procedures, and the only difference is the Data Source to be connected to your applications. Refer to Section 5.16.3 in Chapter 5 and Appendix C to get detailed information in how to use and set up our Oracle sample database **CSE_DEPT** as a data source for Visual Basic.NET projects. All user interfaces and codes are identical with those codes in the last project.

Rename the project **SQLUpdateDeleteWizard** to **OracleUpdateDeleteWizard** and perform the following modifications to a new project **OracleUpdateDeleteWizard** using the DataSet Configuration Wizard:

- Remove the SQL Server database **CSE_DEPT.mdf** and **CSE_DEPTDataSet.xsd** from the project.
- Add our sample Oracle database as a new data source to this project (refer to Section 5.16.3 in Chapter 5 to add this new data source).

You also need to build the following query functions and stored procedures for the project **OracleUpdateDeleteWizard** using the DataSet Configuration Wizard:

- LogIn Form:
 1. Query function—**FillByUserNamePassWord()**.
 2. Query function—**PassWordQuery()**.
- Faculty Form:
 1. Query function—**FillByFacultyName()**.
 2. Query function—**FindFacultyIDByName()**.
 3. Query function—**InsertFaculty()**.

- 4. Query function—UpdateFaculty().
- 5. Query function—DeleteFaculty().
- Course Form:
 - 1. Query function—FillByFacultyID().
 - 2. Stored procedure—InsertCourseSP().

A completed project `OracleUpdateDeleteWizard` can be found in the folder `DBProjects\Chapter 7` located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Finally, let's discuss how to build a data driven application to update and delete data against the Microsoft Access 2007 database.

7.4 UPDATE AND DELETE DATA FOR MICROSOFT ACCESS DATABASE

It is very similar to develop a Visual Basic.NET project to modify data against the Microsoft Access database using the Update and Delete commands, and the only difference is the Data Source to be connected to your applications. Refer to Appendix C to add and connect the sample Microsoft Access 2007 database `CSE_DEPT.accdb` with the Visual Basic.NET application using the Design Tools and Wizards.

Also add the query functions and stored procedures as we did for the Oracle database. A completed project `AccessUpdateDelete` can be found in the folder `DBProjects\Chapter 7` located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

PART II DATA UPDATING AND DELETING WITH RUNTIME OBJECTS

Updating or deleting data against the database using the runtime objects method is a flexible and professional way to perform the data modification jobs in Visual Basic.NET environment. Compared with the method we 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 even if a more complicated coding job is needed. Relatively speaking, the methods we discussed in the first part are easy to learn and code, but some limitations exist for those methods. 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 times that you can modify that query function either before the project runs or after the project runs. In other words, you cannot modify that query function during the project runs.

To overcome those shortcomings, we will discuss how to update or delete data using the runtime object method in this part.

Basically, you need to use the `TableAdapter` to perform data actions against the database if you develop your applications using the Visual Studio.NET design tools and

wizards in the design time. But you should use the `DataAdapter` to make those data manipulations if you develop your project using the runtime objects method.

7.5 THE 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 methodology of this method, as well as some keynotes in using this method to perform the data updating and deleting again the databases.

As you know, ADO.NET provides different classes to help users to develop professional data-driven applications by using 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 command using the `Command` object, and then call the command's method `ExecuteNonQuery()` to update or delete records against 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`, and then you 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. 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 continue to provide any discussion for this method in this part.

The second method is to allow 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, and then build a query statement and attach some useful parameter into that query for the newly created `Command` object. You can update or

delete data against 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 `SQLUpdateDeleteRuntimeObject`, `AccUpdateDeleteRuntimeObject`, and `OracleUpdateDeleteRuntimeObject` to illustrate how to update or delete records against 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 against the SQL Server database using the sample project `SQLUpdateDeleteRuntimeObject` 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 the data updating and deleting against our sample databases using the LINQ to SQL query method. A sample project `LINQSQLUpdateDelete` will be developed in this chapter to discuss 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 THE RUNTIME OBJECTS

Now, let's first develop the sample project `SQLUpdateDeleteRuntimeObject` to update and delete data against the SQL Server database using the runtime object method. Recall that in Sections 5.18.3–5.18.5 in Chapter 5, we discussed how to select data for the Faculty, Course, and Student Form windows using the runtime object 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 against the Faculty table from the Faculty form window using the runtime object method.
2. Update and delete data against the Faculty table from the Faculty form using the runtime stored procedure method.

In order to avoid the duplication on the coding process, we will modify an existing project `SQLInsertRuntimeObject` we developed in Chapter 6 to create our new project `SQLUpdateDeleteRuntimeObject` used in this section.

Open the Windows Explorer and create a new folder such as **Chapter 7** if you have not, and then browse to the folder `DBProjects\Chapter 6` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy the project `SQLInsertRuntimeObject` to the new folder `C:\Chapter 7` we just created. Change the name of the solution and project folder from `SQLInsertRuntimeObject` to `SQLUpdateDeleteRuntimeObject`. Also change the project name from `SQLInsertRuntimeObject.vbproj` to `SQLUpdateDeleteRuntimeObject.vbproj`. Then double-click on the `SQLUpdateDeleteRuntimeObject.vbproj` to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to the **Project! SQLUpdateDeleteRuntimeObject Properties** menu item to open the project's property window. Change the **Assembly name** and the **Root namespace** from `SQLInsertRuntimeObject` to `SQLUpdateDeleteRuntimeObject`, respectively.

- Click on the **Assembly Information** button to open the Assembly Information wizard. Change the Title and the Product to **SQLUpdateDeleteRuntimeObject**. Click on the OK button to close this wizard.

Go to the **FileSave 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 Against the Faculty Table for the SQL Server Database

Let's first discuss updating data against the Faculty table for the SQL Server database. To update data against the Faculty data table, we do not need to add any new Windows form and we can use the Faculty form as the user interface. We need to perform the following two steps to complete this data updating action:

1. Develop codes to update the faculty data
2. Validate the data updating

First, let's develop the codes for our data updating action.

7.6.1.1 Develop Codes to Update the Faculty Data

Open the **Update** button's Click event procedure on the Faculty form window by double-clicking on the **Update** button from the Faculty form window and enter the codes that are shown in the top of Figure 7.12 into this event procedure.

Let's take a closer look at this piece of codes to see how it works.

- A. The Update query string is defined first at the beginning of this procedure. Six columns (except the column **faculty_id**) in the Faculty table are input parameters. The dynamic parameter **@fid** represents the **faculty_id**, which works as the query qualification and should not be updated.
- B. Some data components and local variables are declared here, such as the **Command** object and **intUpdate**. The **intUpdate** is used to hold the returned data value from calling of the **ExecuteNonQuery()** method.
- C. The **Command** object is initialized and built using the connection object and the parameter object.
- D. A user-defined subroutine **UpdateParameters()** is called to add all updated parameters into the **Command** object.
- 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.
- G. The updated faculty name is added into the Faculty Name combo box, and this name is used for the validation purpose later.
- H. 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

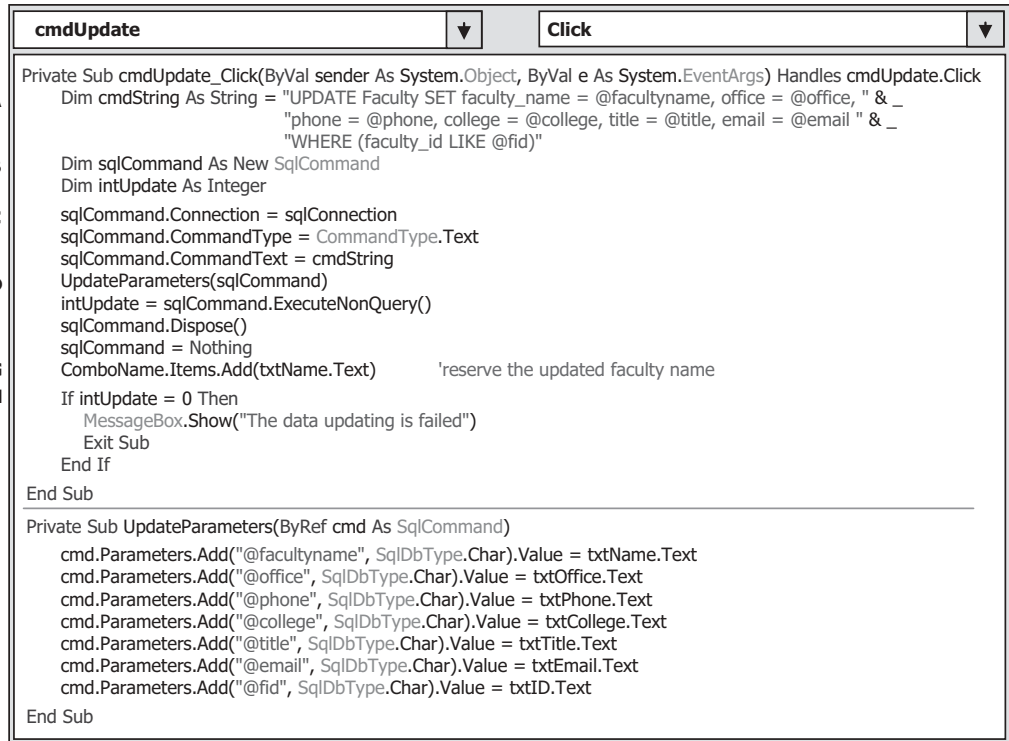


Figure 7.12. The codes for the data updating operation.

has been updated, an error message is displayed, and the procedure is exited if this situation occurred.

- I. The detailed codes for the user defined subroutine procedure `UpdateParameters()` are shown in the lower part in Figure 7.12. Six pieces of updated faculty information and the query qualification `faculty_id` are assigned to the associated columns in the Faculty table.

At this point, we finished the coding process 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.

7.6.1.2 Validate the Data Updating

We do not need to add any new form window to perform this data validation; instead, we can use the Faculty form to perform this job. 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 want to complete the coding process for the data deleting operation.

7.6.2 Delete Data from the Faculty Table for the SQL Server Database

As we mentioned in the previous section, to delete a faculty record from our database, we have to follow two steps listed below:

1. First, delete records from the child tables (LogIn and Course tables)
2. Second, delete record from the parent table (Faculty table)

The data deleting function can be performed by using the **Delete** button's Click event procedure in the Faculty Form window. Therefore, the main coding job for this function is performed inside that procedure.

7.6.2.1 Develop Codes to Delete Data

Open the **Delete** button's Click event procedure by double-clicking on the **Delete** button from the Faculty form window, and enter the codes that are shown in Figure 7.13 into this event procedure.

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

- A. The deleting query string is declared first at the beginning of this procedure. The only parameter is the faculty name. Although the primary key of the Faculty table is **faculty_id**, but 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 name as criterion in this query is that no duplicated faculty name can be used in the Faculty table for this application. In

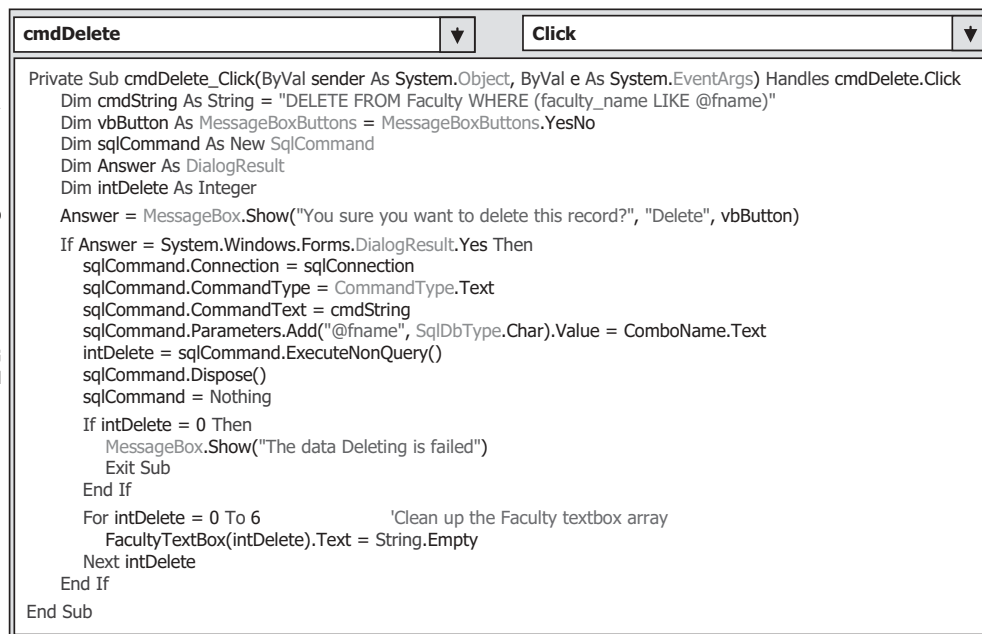


Figure 7.13. The codes for the data-deleting query.

other words, each faculty name must be unique in the Faculty table. A solution to this problem is that we can use the `faculty_id` as the query criterion 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.** Some useful components and local variables are declared here. The data type of the variable `Answer` is `DialogResult`, but one can use an integer to replace it.
- 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 data 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 `@fname` is replaced by the real parameter, the faculty name stored in the Faculty Name combo box. A key point to be noted is that you must use the faculty name stored in the combo box control, which is an old faculty name, and you cannot use the faculty name stored in the Faculty Name textbox since that is an updated 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 this data deleting action.
- 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 procedure is exited if that situation occurred.
- J.** After the data deleting is done, all pieces of faculty information stored in seven textboxes should be cleaned up. A `For` loop is used to finish this cleaning job.

Finally, let's take care of the coding process to validate the data deleting query.

7.6.2.2 *Validate the Data Deleting*

As we did for the validation for the data updating in the last section, we do not need to create any new form window to do this validation; instead, we can use the Faculty form to perform this data validation.

Now let's run the project to test both data updating and 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. In this application, this default folder is the folder in which the executable file of our Visual Basic.NET project is located, which is `C:\Chapter 7\SQLUpdateDeleteRTObject\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 perform a query to get a faculty record and display it in this form.

On the opened Faculty form window, keep the default faculty member **Ying Bai** in the Faculty Name combo box selected. Then update this faculty record by entering the following data to the associated textbox as an updated faculty record:

- Susan Bai Faculty Name textbox
- Professor Title textbox
- MTC-358 Office textbox
- 750-378-5577 Phone textbox
- Duke University College textbox
- sbai@college.edu Email textbox
- Default.jpg Faculty Image textbox

Then click on the **Update** button to update this record in the Faculty table.

To confirm this data updating, go to the Faculty Name combo box control and try to find the updated faculty name from this combo box in terms of the name. Immediately, you can find this updated faculty name. However, in order to test this data updating, first, let's select another faculty name from this box and click on the **Select** button to show all pieces of information for that faculty. Then select our updated faculty name from the box, and click on the **Select** button to retrieve that updated faculty record. Immediately, you can find that all pieces of updated information related to that faculty are displayed in this form. This means that our data updating is successful. Your updated faculty information window should match the one that is shown in Figure 7.14.

Now let's test the data deleting functionality by clicking on the **Delete** button to try to delete this updated faculty record from the Faculty table. Click on the **Yes** button to the message box, and all pieces of updated faculty information stored in seven textboxes are gone. Is our data deleting successful? To answer this question, click on the **Select** button again to try to retrieve that deleted faculty record from the Faculty table. What happened after you click on the **Select** button? A message "No matched faculty found" is shown up, and this means that the faculty record has been successfully deleted from

The screenshot shows a Windows application window titled "CSE DEPT Faculty Form". The window contains several controls:

- Faculty Image:** A label above a text box containing "Default.jpg". Below it is a placeholder image of a person's head and shoulders.
- Faculty Name_Query Method:** A section containing two dropdown menus. The first is labeled "Faculty Name" and has "Susan Bai" selected. The second is labeled "Query Method" and has "TableAdapter Method" selected.
- Faculty Information:** A section containing seven text boxes with the following values:
 - Faculty ID: 878880
 - Name: Susan Bai
 - Title: Professor
 - Office: MTC-358
 - Phone: 750-378-5577
 - College: Duke University
 - Email: sbai@college.edu
- Buttons:** At the bottom of the window are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 7.14. The updated faculty information window.

the Faculty table. Yes, our data deleting is successful. Click on the **Back** and the **Exit** button to exit the project.

It is highly recommended to recover the deleted faculty record for our sample database. To do that recovery, you need to follow the operational order listed below:

- First, recover the deleted faculty record from the parent table (Faculty and Course tables).
- Then, recover the deleted faculty record in the child tables (LogIn and the StudentCourse tables).

Refer to Figure 7.11 and Tables 7.3–7.6 in this chapter to complete this data recovery job. You can access and recover each table in our sample database via the Server Explorer window to perform this recovery process.

A completed project **SQLUpdateDeleteRTOObject** can be found in the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Next, let's discuss how to update and delete data using the runtime object method for Oracle database.

7.7 UPDATE AND DELETE DATA FOR ORACLE DATABASE USING THE RUNTIME OBJECTS

Because of the similar codes in the SQL Server and Oracle databases for the data updating and deleting, we only show the codes that are different with those for the SQL Server database. The main differences between the SQL Server and Oracle databases are the query strings for data deleting and updating. In this section, we concentrate on these query strings.

First, let's modify an existing project to create our new project. We want to modify the project **SQLUpdateDeleteRTOObject** we developed in the last section to create our new project **OracleUpdateDeleteRTOObject** used in this section.

Open the Windows Explorer and create a new folder such as **Chapter 7** if you have not, and then browse to the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy the project **SQLUpdateDeleteRTOObject** to the new folder **C:\Chapter 7**. Change the name of the solution and the project folders from **SQLUpdateDeleteRTOObject** to **OracleUpdataDeleteRTOObject**. Also, change the project file **SQLUpdateDeleteRTOObject.vbproj** to **OracleUpdataDeleteRTOObject.vbproj**. Then, double-click on the **OracleUpdataDeleteRTOObject.vbproj** to open this project.

On the opened project, perform the following modifications to get our desired project:

- Select one form window, such as the **LogIn Form.vb**, from the Solution Explorer window. Then go to **Project\OracleUpdataDeleteRTOObject Properties** menu item to open the project's property window. Change the **Assembly name** and the **Root namespace** from **SQLUpdateDeleteRTOObject** to **OracleUpdataDeleteRTOObject**.
- Click on the **Assembly Information** button to open the **Assembly Information** wizard, change the **Title** and the **Product** to **OracleUpdataDeleteRTOObject**. Click on the **OK** button to close this wizard.

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 all graphical user interfaces from 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 to the related codes:

1. Add the Oracle namespace reference to the project and modify the Imports commands.
2. Modify the connection string in the ConnModule and the LogIn form.
3. Modify the SELECT query string for the LogIn button's Click event procedure in the LogIn form.
4. Modify the SELECT query string for the Select button's Click event procedure in the Faculty form.
5. Modify the INSERT query string for the Insert button's Click event procedure in the Faculty Form window.
6. Modify the parameters' names for the INSERT command object in the Faculty Form window.
7. Modify the UPDATE query string for the Update button's Click event procedure in the Faculty form.
8. Modify the DELETE query string for the Delete button's Click event procedure in the Faculty form.
9. Modify the parameters' names for the UPDATE and the DELETE command objects in the Faculty form.
10. Modify two SELECT query strings for the Select button's Click event procedure, and the SelectedIndexChanged event procedure of the Course listbox in the Course form.
11. Modify the user-defined subroutine BuildCommand() and the SELECT query string for the Select button's Click event procedure in the Student form.
12. Delete SP Form.vb since we will not use this form in this project.

Well, it looks like that there are too many modifications we need to do for this project. But exactly it is easy to handle those modifications. Let's begin our first modification.

7.7.1 Add the Oracle Namespace Reference and Modify the Imports Command

Perform the following operations to complete this reference addition operation:

1. Right-click on our project **OracleUpdataDeleteRTOject** from the Solution Explorer window and select the **Add Reference** item from the pop-up menu to open the Add reference wizard.
2. With the .NET tab selected, scroll down the list until you find the items **Devart.Data** and **Devart.Data.Oracle**. Click on both to select them and click on the OK button to add these two references to our project.

Open the code windows of the following forms from the current project:

- ConnModule.vb
- LogIn Form.vb

- Faculty Form.vb
- Course Form.vb
- Student Form.vb

Perform the following operations to modify these Imports commands:

- Change Imports System.Data to Imports Devart.Data.
- Change Imports System.Data.SqlClient to Imports Devart.Data.Oracle.

On the ConnModule code window, change the connection class and object from:

```
Public sqlConnection As SqlConnection
To: Public oraConnection As OracleConnection
```

7.7.2 Modify the Connection String and Query String for the LogIn Form

The modifications to the LogIn form can be divided into three parts: modifications to the connection string in the Form_Load event procedure, modifications to the SELECT query string in the TableAdapter LogIn button's Click event procedure, and modifications to the SELECT query string in the DataReader LogIn button's Click event procedure.

Let's start from the first part.

7.7.2.1 Modify the Connection String in the Form Load Event Procedure

Open the Form_Load event procedure of the LogIn form and change the connection string to:

```
Dim oraString As String = "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.

7.7.2.2 Modify the SELECT Query String in the TabLogIn Button Event Procedure

Open the TabLogIn button's Click event procedure and change the SELECT query string to:

```
Dim cmdString1 As String = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "
Dim cmdString2 As String = "WHERE user_name = :Param1 AND pass_word = :Param2"
```

Also, change the prefixes of all data classes from Sql to Oracle, the prefixes of all data objects from sql to ora. Change two dynamic parameters' names from @Param1 to Param1, and from @Param2 to Param2, respectively.

7.7.2.3 Modify the SELECT Query String in the ReadLogIn Button Event Procedure

Open the ReadLogIn button's Click event procedure and change the SELECT query string to:

```
Dim cmdString1 As String = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "
Dim cmdString2 As String = "WHERE user_name = :name AND pass_word = :word"
```

Also, change the prefixes of all data classes from Sql to Oracle, the prefixes of all data objects from sql to ora. Change two dynamic parameters' names from @name to name, and from @word to word, respectively.

7.7.3 Modify the Query Strings for the Faculty Form

This modification can also be divided into three parts: Modifications to the query string for the Select button's Click event procedure, modifications to the query string for the Update button's Click event procedure, and modifications to the query string for the Delete button's Click event procedure in the Faculty form.

7.7.3.1 Modify the SELECT Query String for the Select Button Event Procedure

Open the Select button's Click event procedure and change the query string to:

```
Dim cmdString1 As String = "SELECT faculty_id, faculty_name, office, phone, college, title, " & _
    "email FROM Faculty "
Dim cmdString2 As String = "WHERE faculty_name = :facultyName"
```

Also, change the prefixes of all data classes from Sql to Oracle, the prefixes of all data objects from sql to ora for all event procedures in this form. Change the dynamic parameter's name from @facultyName to facultyName.

7.7.3.2 Modify the INSERT Query String for the Insert Button Event Procedure

Open the Insert button's Click event procedure and change the query string to:

```
Dim cmdString As String = "INSERT INTO Faculty (faculty_id, faculty_name, office, phone, college, " & _
    "title,email)VALUES(:faculty_id,:faculty_name,:office,:phone,:college,:title,:email)"
```

Change the prefixes of all data classes from Sql to Oracle, the prefixes of all data objects from sql to ora. Also, modify the data types and the names of the dynamic parameters inside the user-defined subroutine InsertParameters() as below:

- Change the data type of the passed argument command object from `SqlCommand` to `OracleCommand`.
- Change the data type for all parameters from `SqlDbType` to `OracleDbType`.
- Remove the `@` symbol before all parameters' names.

Next, let's handle the modifications to the `Update` button Click event procedure.

7.7.3.3 Modify the UPDATE Query String for the Update Button Event Procedure

Open the `Update` button's Click event procedure and change the query string to:

```
Dim cmdString As String = "UPDATE Faculty SET faculty_name = :facultyname, office = :office, " & _
    "phone = :phone, college = :college, title = :title, email = :email " & _
    "WHERE (faculty_id = : fid)"
```

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 names of the dynamic parameters inside the `UpdateParameters()` subroutine as below:

- Change the data type of the argument `cmd` to `OracleCommand`.
- Change the data type for all parameters from `SqlDbType` to `OracleDbType`
- Remove the `@` symbol before all parameters' names

Next, let's handle the modifications to the `Delete` button Click event procedure.

7.7.3.4 Modify the DELETE Query String for the Delete Button Event Procedure

Open the `Delete` button's Click event procedure and change the query string to:

```
Dim cmdString As String = "DELETE FROM Faculty WHERE (faculty_name = : fname)"
```

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 `@fname` to `fname`, then its data type from `SqlDbType` to `OracleDbType`.

At this point, we have finished all code modifications to the `LogIn` and the `Faculty` forms. Next, let's handle the codes modifications to the `Course` form.

7.7.4 Modify the Query Strings for the Course Form

The modification to this form can be divided into two parts: modifications to the query string for the `Select` button's Click event procedure and modifications to the query string for the `Course Listbox`'s `SelectedIndexChanged` event procedure.

7.7.4.1 *Modify the SELECT Query String for the Select Button Event Procedure*

Open the Select button's Click event procedure and change the query string to:

```
Dim cString1 As String = "SELECT Course.course_id, Course.course FROM Course JOIN Faculty "
Dim cString2 As String="ON (Course.faculty_id = Faculty.faculty_id) AND (Faculty.faculty_name = :name)"
```

Change the prefixes of all data classes from **Sql** to **Oracle**, the prefixes of all data objects from **sql** to **ora**. Also, change the dynamic parameter's name from **@name** to **name**, its data type from **SqlDbType** to **OracleDbType**.

Another modification is to change the data type of the argument **CourseReader** to the **OracleDataReader** in the user-defined subroutine **FillCourseReader()**.

7.7.4.2 *Modify the SELECT Query String for the CourseList Event Procedure*

Open the Course Listbox's **SelectedIndexChanged** event procedure and change the query string to:

```
Dim cmdString1 As String = "SELECT course_id, course, credit, classroom, schedule, " & _
                           "enrollment FROM Course "
Dim cmdString2 As String = "WHERE course_id = : courseid"
```

Change the prefixes of all data classes from **Sql** to **Oracle**, the prefixes of all data objects from **sql** to **ora**. Also, change the dynamic parameter's name from **@courseid** to **courseid**, and its data type from **SqlDbType** to **OracleDbType**.

Also, change the data type of the argument **CourseReader** in the user-defined subroutine **FillCourseReaderTextBox()** from **SqlDataReader** to **OracleDataReader**.

7.7.5 **Modify the Query Strings for the Student Form**

Two modifications are involved in this form: modifications to the **Select** button click event procedure and modifications to the user-defined subroutine **BuildCommand()**.

Open the **Select** button click event procedure and change the prefixes of all data classes from **Sql** to **Oracle**, the prefixes of all data objects from **sql** to **ora**. Change the data type of the argument **cmdObj** in the user defined subroutine **BuildCommand()** from **SqlCommand** to **OracleCommand**.

7.7.6 **Other Modifications**

Change the prefixes of all data classes from **Sql** to **Oracle**, the prefixes of all data objects from **sql** to **ora**. These modifications include the following procedures:

- Cancel button's Click event procedure in the **LogIn** form.
- **Form_Load** event procedure in the **Faculty** form

- Form_Load event procedure in the Course form
- Form_Load event procedure in the Student Form
- Exit button's Click event procedure in the Selection form

Remove the coding lines that contains the SP Form class **SPForm** and object **spform** from the OK button Click event procedure in the Selection Form code window.

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. Enter the following eight pieces of information into the associated textbox in this form as a new faculty record:

- P33431 Faculty ID textbox
- Peter Steff Faculty Name textbox
- Associate Professor Title textbox
- MTC-235 Office textbox
- 750-378-1130 Phone textbox
- University of Hawaii College textbox
- psteff@college.edu Email textbox
- Default.jpg Faculty Image textbox

Then click on the **Insert** button to try to insert this new faculty record into the Faculty table in the database. To confirm this data insertion, click on the drop-down arrow from the Faculty Name combo box, and you can find that the newly inserted faculty's name is in there. Click it to select it, and click on the **Select** button to retrieve this new record and display it in this form, which is shown in Figure 7.15.

CSE DEPT Faculty Form

Faculty Image
Default.jpg

Faculty Name _Query Method
Faculty Name Peter Steff
Query Method TableAdapter Method

Faculty Information
Faculty ID P33431
Name Peter Steff
Title Associate Professor
Office MTC-235
Phone 750-378-1130
College University of Hawaii
Email psteff@college.edu

Select **Insert** **Update** **Delete** **Back**

Figure 7.15. The running status of the Faculty form.

To update this faculty record, change the faculty information as follows:

- Peter Jones Faculty Name textbox
- Professor Title textbox
- MTC-555 Office textbox
- 750-330-3355 Phone textbox
- University of Florida College textbox
- pjones@college.edu Email textbox

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 from the Faculty Name combo box. First, we may 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 Jones** from the Faculty Name combo box and click on the **Select** button to try to retrieve this updated faculty record and display it in this form. Immediately, you can find that the faculty record has been updated and displayed, which is shown in Figure 7.16. Our data updating is successful.

Now, let's test our data deleting function. Keep the updated faculty name **Peter Jones** selected in the Faculty Name combo box and click on the **Delete** button to try to delete it from the Faculty table in the database. Click **Yes** on the confirmation message box, and you can find that all pieces of information related to that faculty are removed from all textboxes. To confirm that data deleting, click on the **Select** button to try to retrieve that deleted record from the Faculty table, a message "No matched faculty found!" is displayed to indicate that the querying faculty record has been deleted from the database. Yes, our data deleting is also successful.

The screenshot shows a Java Swing window titled "CSE DEPT Faculty Form". It contains several input fields and buttons. On the left, there is a "Faculty Image" section with a text field labeled "Default.jpg" and a placeholder image of a person. On the right, there is a "Faculty Name_Query Method" section with a "Faculty Name" dropdown menu showing "Peter Jones" and a "Query Method" dropdown menu showing "TableAdapter Method". Below these is a "Faculty Information" section with text fields for "Faculty ID" (P33431), "Name" (Peter Jones), "Title" (Professor), "Office" (MTC-555), "Phone" (750-330-3355), "College" (University of Florida), and "Email" (pjones@college.edu). At the bottom of the form are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 7.16. The confirmation of the data updating operation.

A complete project `OracleUpdateDeleteRTObject` can be found in the folder `DBProjects\Chapter 7` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

7.8 UPDATE AND DELETE DATA AGAINST 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 in most time, it is unnecessary 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, because the primary key `faculty_id` is good for the lifetime of the database in actual applications. Therefore, based on this fact, 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 two parts based on two types of databases we used in this book: using stored procedures to update and delete data against (1) the SQL Server 2008 database and (2) the Oracle 11g XE 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 Against SQL Server Database Using Stored Procedures

Updating and deleting data using stored procedures developed in the SQL Server database are very similar to the data updating and deleting we performed in the last section. With a small modification to the existing project `SQLUpdateDeleteRTObject`, 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 `SQLUpdateDeleteRTObject` to create our new project `SQLUpdateDeleteSP`.
2. Develop the data updating and deleting stored procedures in the SQL Server database.
3. Call the 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.

7.8.1.1 *Modify the Existing Project to Create Our New Project*

Open the Windows Explorer and create a new folder **Chapter 7** if you have not, and then browse to the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy the project **SQLUpdateDeleteRTOObject** to the new folder **C:\Chapter 7**. Change the names of the solution and the project folders from **SQLUpdateDeleteRTOObject** to **SQLUpdateDeleteSP**. Also, change the name of the project file **SQLUpdateDeleteRTOObject.vbproj** to **SQLUpdateDeleteSP.vbproj**. Then double-click on the project **SQLUpdateDeleteSP.vbproj** to open this new project.

On the opened project, perform the following modifications to get our desired project:

- Select a form window, such as the **LogIn Form.vb**, from the Solution Explorer window. Then go to the **Project\SQLUpdataRTOObjectSP Properties** menu item to open the project's property wizard. Change the **Assembly name** and the **Root namespace** from **SQLUpdateDeleteRTOObject** to **SQLUpdateDeleteSP**.
- Click on the **Assembly Information** button to open the **Assembly Information** wizard. Change the **Title** and the **Product** to **SQLUpdateDeleteSP**. Click on the **OK** button to close this dialog box.

Go to the **File\Save All** to save those modifications. Now we are ready to modify the codes based on our new project **SQLUpdateDeleteSP**.

7.8.1.2 *Modify the Codes to Update and Delete Data from the Faculty Table*

The code modifications include the following parts:

1. Replace the query string in the **Update** button's Click event procedure in the **Faculty Form** with the name of the data updating stored procedure that will be developed in the next section to allow the procedure to call the related stored procedure to perform the data updating action.
2. Replace the query string in the **Delete** button's Click event procedure in the **Faculty Form** with the name of the data deleting stored procedure that will be developed in the next section to allow the procedure to call the related stored procedure to perform the data deleting action.

Regularly, 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 that we have known those pieces of information and we can put those pieces of information into these two event procedures in advance. The assumed information includes:

- A. For the **Faculty Data Updating**:
 1. The name of the data updating stored procedure—**dbo.UpdateFacultySP**.
 2. The names of the input updating parameters—identical with the columns' names in the **Faculty** table in our sample database.
 3. The name of the input dynamic parameter—**@FacultyName**.

B. For the Faculty Data Deleting:

1. The name of the data deleting stored procedure—**dbo.DeleteFacultySP**.
2. The names of the deleting input parameters—identical with the columns' names in the Faculty table in our sample database.
3. The name of the input dynamic parameter—**@FacultyName**.

Based on these assumptions, we can first modify our codes in the **Update** button's Click event procedure. 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 event procedure and perform the appropriate modifications. Your finished modifications to this event procedure should match those codes that are shown in Figure 7.17. The modified codes have been highlighted in bold.

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

- A.** The content of the query string now should be equal to the name of the stored procedure, **dbo.UpdateFacultySP**, which will be built 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.

There is no modifications to the user-defined subroutine **UpdateParameters()**.

Next, let's modify the codes in the **Delete** button's Click event procedure. Open this event procedure and perform the modifications shown in Figure 7.18 to this procedure. The modified codes have been highlighted in bold.

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

- A.** The content of the query string now should be equal to the name of the stored procedure, **dbo.DeleteFacultySP**, which will be built later.

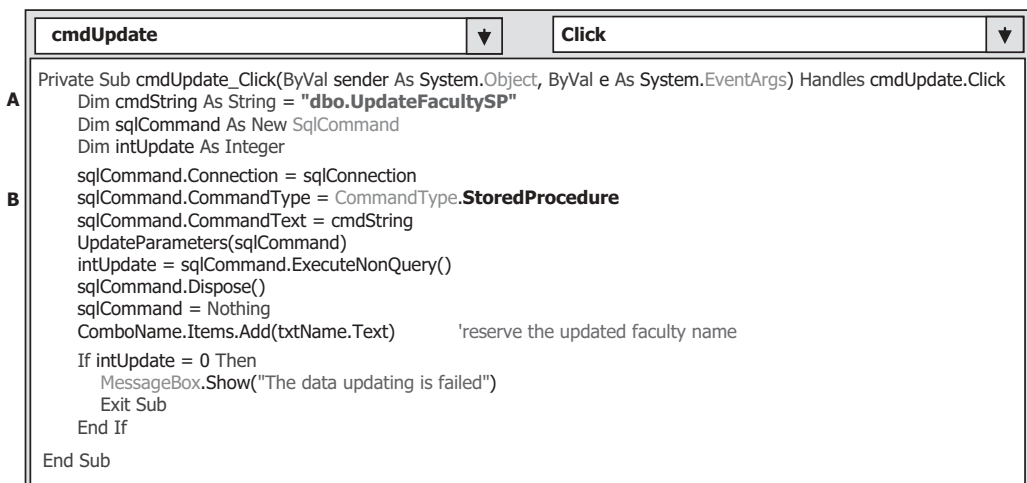


Figure 7.17. The modified codes for the Update button's event procedure.

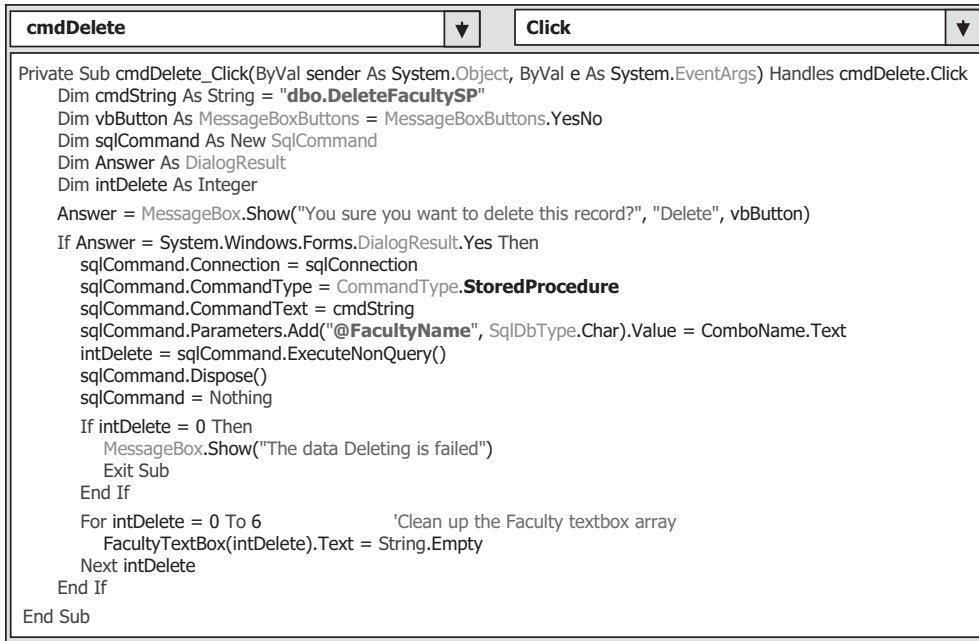


Figure 7.18. The modified codes to call the stored procedure.

- 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 input dynamic parameter to the stored procedure is @FacultyName, and this will work as a query qualification for this action.

Now we have finished all codes modifications in Visual Basic.NET environment. Let's start to create our stored procedures in the SQL Server database. There are two ways you can create the stored procedure: (1) create it in the SQL Server Management Studio Express, and (2) create it in the Server Explorer in the Visual Studio.NET environment. Since we are working for the Visual Basic.NET project, we prefer to use the second way to create our stored procedures.

7.8.1.3 Develop Two Stored Procedures in the SQL Server Database

Open the Server Explorer in the Visual Studio.NET 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 to dbo.UpdateFacultySP, which should be identical with the name of the stored procedure we used in our codes in the last section. Then add the codes that are shown in Figure 7.19 into this stored procedure as the body of our new stored procedure.

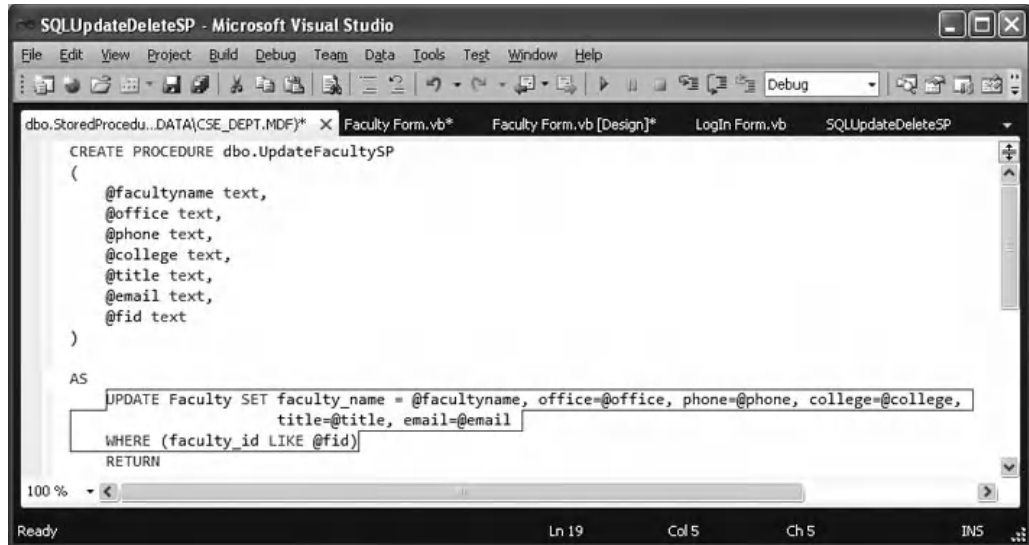


Figure 7.19. The created stored procedure `dbo.UpdateFacultySP`.

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. Here, we used the `text` to replace the `nvarchar()` since they are similar in this procedure.

Go to the menu item **File|Save StoredProcedure1** to save our stored procedure.

To test our stored procedure, right-click on our newly created stored procedure `dbo.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** wizard. Enter the following updated information into each field on the **Value** column of this wizard:

- Susan Bai Name Value
- MTC-228 Office Value
- 750-378-1220 Phone Value
- Duke University College Value
- Associate Professor Title Value
- sbai@college.edu Email Value
- B78880 faculty_id Value

Your finished information wizard should match the one that is shown in Figure 7.20.

Click on the **OK** button to run this stored procedure. The running result of execution of this stored procedure is shown in the **Output** wizard, as shown in Figure 7.21.

To confirm this data updating action, you can open the Faculty table to check it. Go to the **Server Explorer** window and right-click on the Faculty table, select **Show Table Data** to open the Faculty table. You can find that our updated record is in there, which is

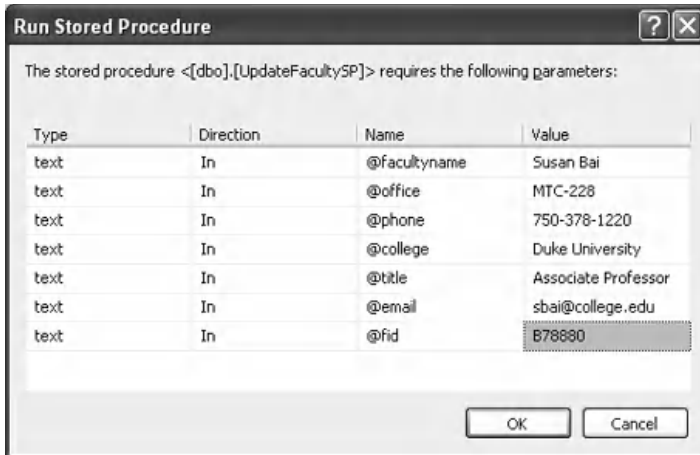


Figure 7.20. The finished information wizard.

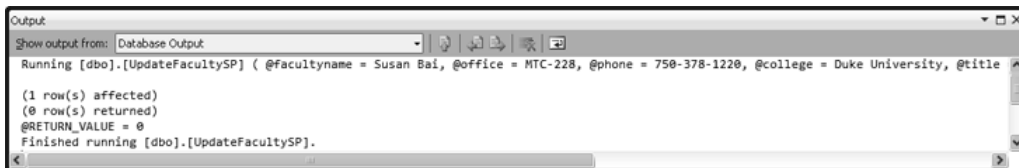


Figure 7.21. The running result of the stored procedure dbo.UpdateFacultySP.

shown as a highlighted row in Figure 7.22. Sometimes, you may need to close and restart the Visual Studio.NET and your project to see this result.

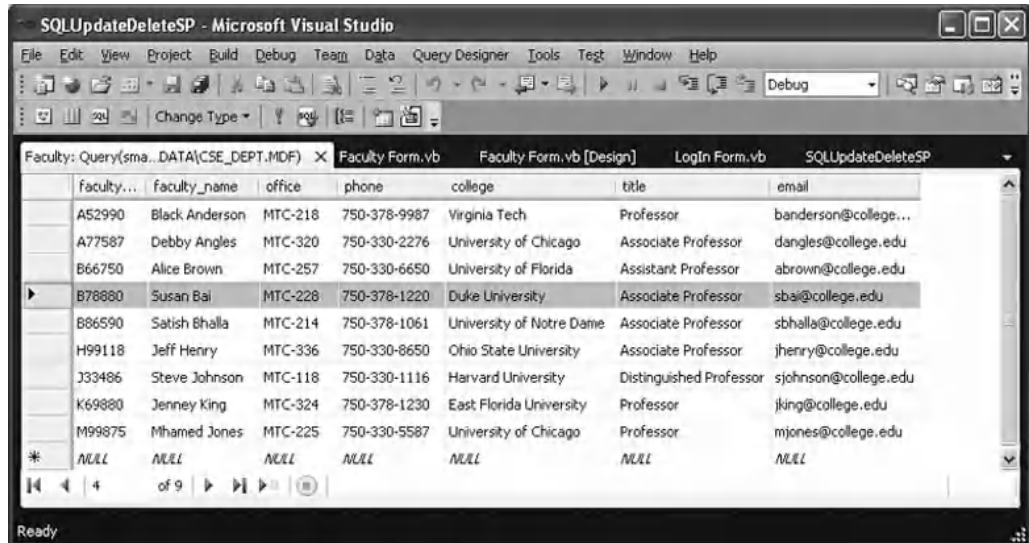
Our stored procedure is successful.

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 to recover it:

- 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

Save and close the database. Next, let's create our data deleting stored procedure `dbo.DeleteFacultySP` in the SQL Server database.

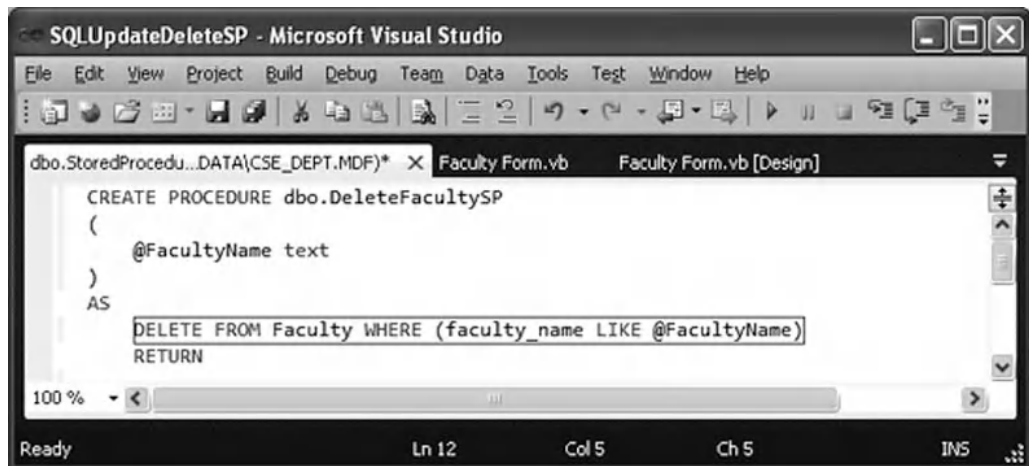
In the opened Server Explorer window, right-click the **Stored Procedures** folder and select the item **Add New Stored Procedure** to open the default procedure wizard.



The screenshot shows the 'SQLUpdateDeleteSP - Microsoft Visual Studio' window. The 'Faculty Form.vb [Design]' tab is active, displaying a table 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...
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	Susan Bai	MTC-228	750-378-1220	Duke University	Associate Professor	sba@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
M99875	Mhamed Jones	MTC-225	750-330-5587	University of Chicago	Professor	mjones@college.edu
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 7.22. The updated Faculty table.



The screenshot shows the 'SQLUpdateDeleteSP - Microsoft Visual Studio' window. The 'Faculty Form.vb [Design]' tab is active, displaying the following SQL code:

```

CREATE PROCEDURE dbo.DeleteFacultySP
(
    @FacultyName text
)
AS
DELETE FROM Faculty WHERE (faculty_name LIKE @FacultyName)
RETURN
  
```

Figure 7.23. The created data deleting stored procedure.

Change the name of the default stored procedure to `dbo.DeleteFacultySP`, which is identical with the name of the stored procedure we used in our codes in the last section. Then add the codes that are shown in Figure 7.23 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 this stored procedure, right-click on our newly created stored procedure `DeleteFacultySP` that is located under the `Stored Procedure` folder in the `Server`

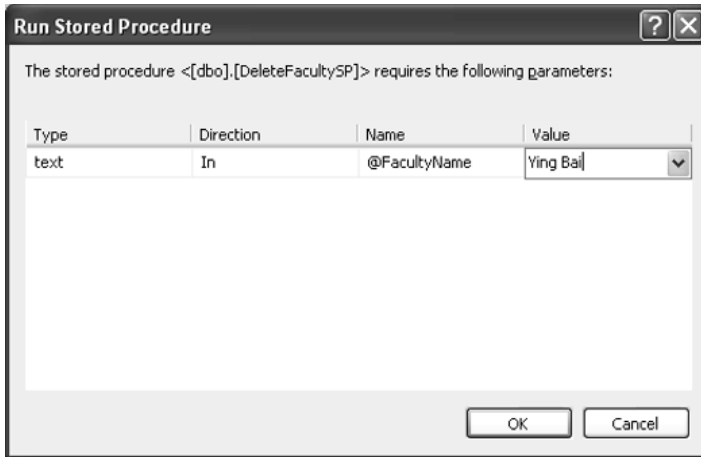


Figure 7.24. The finished Run Stored Procedure wizard.

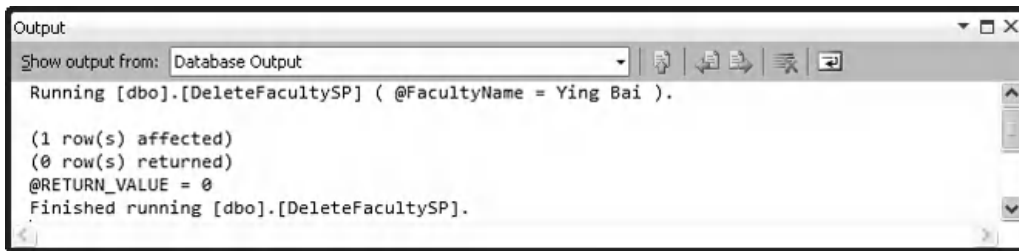


Figure 7.25. The running result of the stored procedure `dbo.DeleteFacultySP`.

Explorer window, and select the item **Execute** to open the **Run Stored Procedure** wizard. Enter the faculty name **Ying Bai** into the **Value** column of this wizard, which is shown in Figure 7.24, to try to delete this faculty record from the **Faculty** table in our sample database.

Click on the **OK** button to run this stored procedure.

The running result is displayed in the **Output** window, which is shown in Figure 7.25.

To confirm the execution of this stored procedure, you can open the **Faculty** table from the **Server Explorer** window to check it. On the opened **Faculty** table, you can find that the faculty member **Ying Bai** with a **faculty_id** of **B78880** has been deleted from this table in our sample SQL Server database.

When checking this data deleting action against the **Faculty** table in our sample database, one point is 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 confirm that data deleting action. 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 this deleted faculty record with the original data. Recall in Figure 7.11 and related discussions in Section 7.2.9, 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 that course in the StudentCourse table (child table) will be deleted, too. Therefore, in total, 11 records in our sample database are 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. In Section 7.2.9, we listed all those deleted data for those four tables in our sample database. For your convenience, we list these data again in Tables 7.7–7.10. You can use the copy/paste functions to first copy all rows from each table, and then paste them at the end of each table in our sample database.

Table 7.7. The 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 7.8. The data to be recovered in the LogIn table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	NULL

Table 7.9. The 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 7.10. The 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

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 as 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 the **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 Basic.NET project to test the data updating and deleting functions.

7.8.1.4 Call the Stored Procedures to Perform the 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. Keep the default faculty member **Ying Bai** selected from the Faculty Name combo box, click on the **Select** button to query, and display the information for the selected faculty.

To update this faculty information, enter the following data into the associated text-boxes as an updated faculty record:

- Peter Bai Faculty Name textbox
- Distinguished Professor Title textbox
- MTC-228 Office textbox
- 750-378-1220 Phone textbox
- University of Main College textbox
- pbai@college.edu Email textbox
- Default.jpg Faculty Image textbox

Click on the **Update** button to call the stored procedure to update this faculty record in the Faculty table in the database.

To confirm this updating, first, let's select any other faculty member from the Faculty Name combo box, click on the **Select** button to query, and display the information related to that selected faculty. Then select our newly updated faculty name **Peter Bai** from the Faculty Name combobox. Click on the **Select** button to retrieve that updated faculty record from the database and display it in this form. Immediately, you can find that the updated faculty record is returned and display in this form, as shown in Figure 7.26.

Now let's test the data deleting action by clicking on the **Delete** button to try to delete this updated faculty record. Click on **Yes** button on the MessageBox to perform this data deleting. Immediately, all pieces of information stored in textboxes are removed. To confirm this data deleting, click on the **Select** button to try to retrieve the deleted faculty record. 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 SQL Server database is very successful.

Figure 7.26. The confirmation of the Faculty data updating.

In order to keep our database neat and complete, refer to the last section and Tables 7.7–7.10 to recover those deleted records at four tables in our sample database via the Server Explorer window.

A complete project `SQLUpdateDeleteSP` can be found in the folder `DBProjects\Chapter 7` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

7.8.2 Update and Delete Data Against Oracle Database Using Stored Procedures

Updating and deleting data using stored procedures developed in the Oracle database are very similar to the data updating and deleting actions we developed in the SQL Server database in the last section. With a small modification to an existing project `OracleUpdateDeleteRTOBJect`, 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 `OracleUpdateDeleteRTOBJect` to create our new project `OracleUpdateDeleteSP`.
2. Develop two 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.

7.8.2.1 Modify the Existing Project to Create Our New Project

Open the Windows Explorer and create a new folder `Chapter 7` if you have not, and then browse to the folder `DBProjects\Chapter 7` that is located at the Wiley ftp site (refer to

Figure 1.2 in Chapter 1). Copy the project `OracleUpdateDeleteRTObject` to the new folder `C:\Chapter 7`. Change the names of the solution and the project folders from `OracleUpdateDeleteRTObject` to `OracleUpdateDeleteSP`. Also, change the name of the project file `OracleUpdateDeleteRTObject.vbproj` to `OracleUpdateDeleteSP.vbproj`. Then double-click on the project `OracleUpdateDeleteSP.vbproj` to open this project.

On the opened project, perform the following modifications to get our desired project:

- Select one form window, such as the `LogIn Form.vb`, from the Solution Explorer window. Then go to the `Project\OracleUpdateDeleteSP Properties` menu item to open the project's property window. Change the `Assembly name` and the `Root namespace` from `OracleUpdateDeleteRTObject` to `OracleUpdateDeleteSP`.
- Click on the `Assembly Information` button to open the `Assembly Information` wizard. Change the `Title` and the `Product` to `OracleUpdateDeleteSP`. Click on the `OK` button to close this wizard.
- Go to the `File\Save All` to save those modifications. Now we are ready to modify our codes based on our new project `OracleUpdateDeleteSP`.

7.8.2.2 *Modify the Codes to Update and Delete Data from the Faculty Table*

The code modifications include the following parts:

1. Replace the query string in the `Update` button's `Click` event procedure in the `Faculty Form` with the name of the data updating stored procedure that will be developed in the next section to allow the event procedure to call the related stored procedure to perform the data updating action.
2. Replace the query string in the `Delete` button's `Click` event procedure in the `Faculty Form` with the name of the data deleting stored procedure that will be developed in the next section to allow the event procedure to call the related stored procedure to perform the data deleting action.

Regularly, these modifications should be performed after stored procedures have been created in the Oracle database since we need some pieces of information from those 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 have known those pieces of information, and we can put them into our event procedures in advance when we perform the codes modifications.

The assumed information includes:

- A. For the `Faculty Data Updating`:
 1. The name of the data updating stored procedure—`UpdateFacultySP`.
 2. The names of the input updating parameters—prefix an `in` before each parameter's name that is equal to the name of each column in the `Faculty` table in our sample database.
 3. The name of the input dynamic parameter—`FacultyName`.
- B. For the `Faculty Data Deleting`:
 1. The name of the data deleting stored procedure—`DeleteFacultySP`.
 2. The name of the input dynamic parameter—`FacultyName`.

Based on these assumptions, we can first modify our codes in the **Update** button's Click event procedure. The key point is that we need to remember the names of these parameters and the names of the stored procedures and put them into our stored procedures when we developed them in the next section.

Open the **Update** button's Click event procedure and modify its codes. Your finished modifications to this event procedure should match those codes that are shown in Figure 7.27. The modified codes have been highlighted in bold.

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

- A.** The content of the query string now should be equal to the name of the stored procedure **UpdateFacultySP**.
- B.** The **CommandType** property of the **Command** object is 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 code modifications to the user defined subroutine procedure **UpdateParameters()** are shown in Figure 7.28.

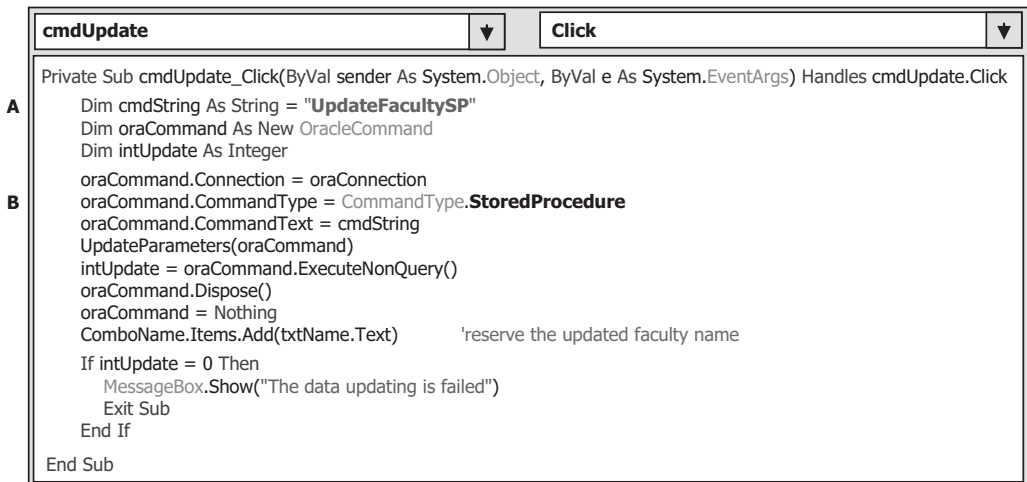


Figure 7.27. The modified codes for the Update button's event procedure.

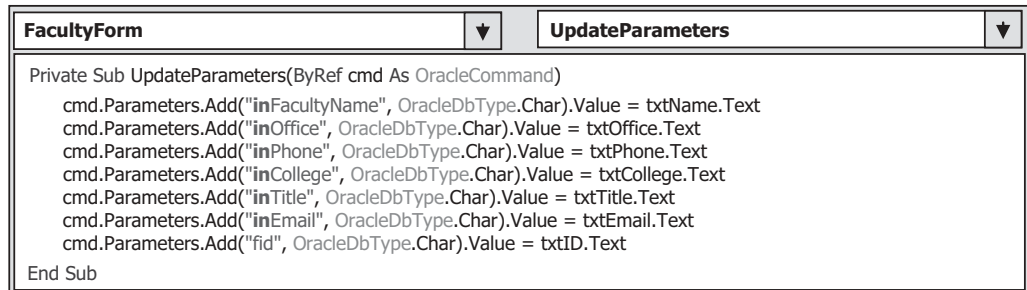


Figure 7.28. The modified codes for the subroutine UpdateParameters().

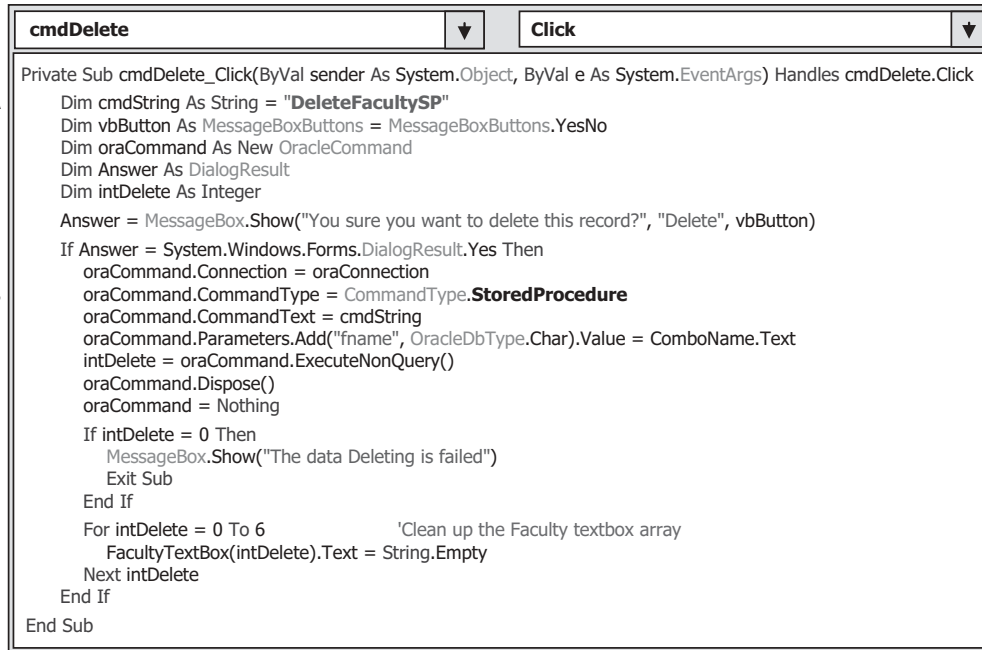


Figure 7.29. The modified codes for the Delete Button's Click event procedure.

Only one modification is performed for this subroutine, which is to add a prefix in before each input updating parameter. The reason for that is: our Oracle stored procedure is written in the PL-SQL language, and it is a case-insensitive language. In order to distinguish between the columns' names of the Faculty table and the input updating parameters' names, we need to add these prefixes.

Next, let's perform the code modification to the Delete button's Click event procedure. Open that event procedure and make the code modifications shown in Figure 7.29 to this event procedure.

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

- A.** The content of the query string now should be equal to the name of the stored procedure DeleteFacultySP.
- B.** The CommandType property of the Command object is set to **StoredProcedure** to tell the project that a stored procedure should be called as the project runs to perform the data deleting action.

Now we have finished all code modifications in Visual Basic.NET environment. Let's start to create our stored procedures in the Oracle database. Refer to Sections 5.20.7.1 and 5.20.7.2 in Chapter 5 to get a detailed discussion about the stored procedure and the package in the Oracle database.

In this section, since we want to perform the data updating and deleting functions, therefore, we do not need any query to return any data from the database. The stored procedure is good enough for our applications. There are many ways to create the stored procedure in the Oracle database; one way is to create it using the Object Browser wizard

in Oracle Database 11g XE, and the second way is to create it using the SQL Commands wizard. Since we used the Object Browser to create our sample database in Chapter 2, therefore, we prefer to use the second way to create our stored procedures in this section.

7.8.2.3 Develop Stored Procedures in the Oracle Database

Open the Oracle Database 11g XE home page by going to **start|All Programs|Oracle Database 11g Express Edition|Get Started** items. This time, we want to use the SQL Command wizard to create our stored procedures. An advantage of using this tool is that you can run and test your stored procedure directly in the Oracle Database 11g XE environment as soon as the stored procedure is done, and that 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 Basic.NET project.

Perform the following operations to open the SQL Commands wizard:

1. Click on the **APEX** button to open the login wizard.
2. Enter the Username and Password to complete this APEX login process. In our case, just enter **SYSTEM** and reback into these two boxes.
3. Click on the **Already have an account? Login Here** button since we have created our sample database **CSE_DEPT** in Chapter 2.
4. Keep the Workspace and the Username's content **CSE_DEPT** unchanged, and enter reback into the Password box to complete the login process for the workspace.
5. Click on the **SQL Workshop** icon and then **SQL Commands** icon to open the SQL Commands wizard.

Enter the codes that are shown in Figure 7.30 into this page as the body of our stored procedure **UpdateFacultySP**.

Your finished stored procedure should match the one that is shown in Figure 7.31.

Now highlight all codes of this stored procedure and click on the **Run** button to create our stored procedure. Immediately, you can find a message is displayed in the bottom

```
Create or replace PROCEDURE UpdateFacultySP
(inFacultyName IN VARCHAR2,
inOffice IN VARCHAR2,
inPhone IN VARCHAR2,
inCollege IN VARCHAR2,
inTitle IN VARCHAR2,
inEmail IN VARCHAR2,
fid IN VARCHAR2) AS
begin
UPDATE Faculty
SET faculty_name = inFacultyName, office = inOffice, phone = inPhone, college = inCollege,
title = inTitle, email = inEmail
WHERE faculty_id = fid;
end;
```

Figure 7.30. The code body of the stored procedure **UpdateFacultySP**.

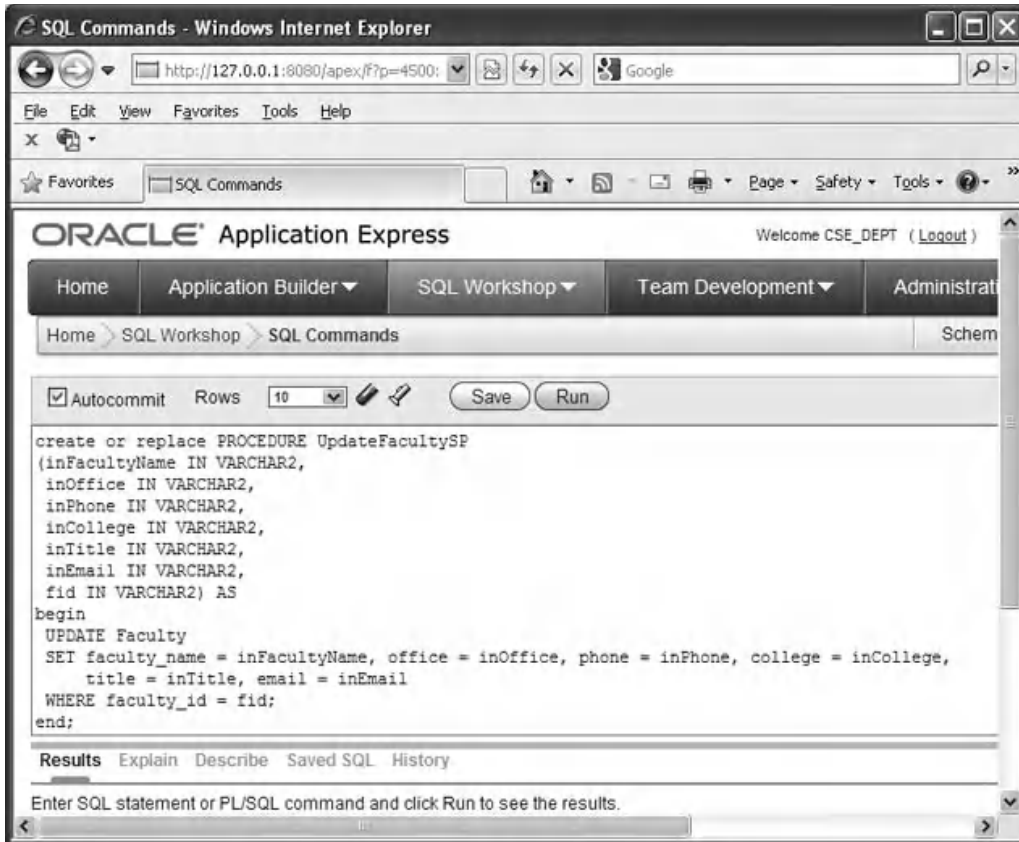


Figure 7.31. The finished code body of the stored procedure UpdateFacultySP.

```
begin
UpdateFacultySP('Susan Bai', 'MTC-355', '750-378-2355', 'Duke University', 'Professor',
                'sbai@college.edu', 'B78880');
end;
```

Figure 7.32. The codes to run the stored procedure UpdateFacultySP.

pane in the Results tab to indicate that the stored procedure is created, which is shown below:

```
Procedure created.
0.79 seconds
```

To run and test this stored procedure, type the codes that are shown in Figure 7.32 under the codes of stored procedure. 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 in Figure 7.33, is displayed in the bottom pane under the Results tab.

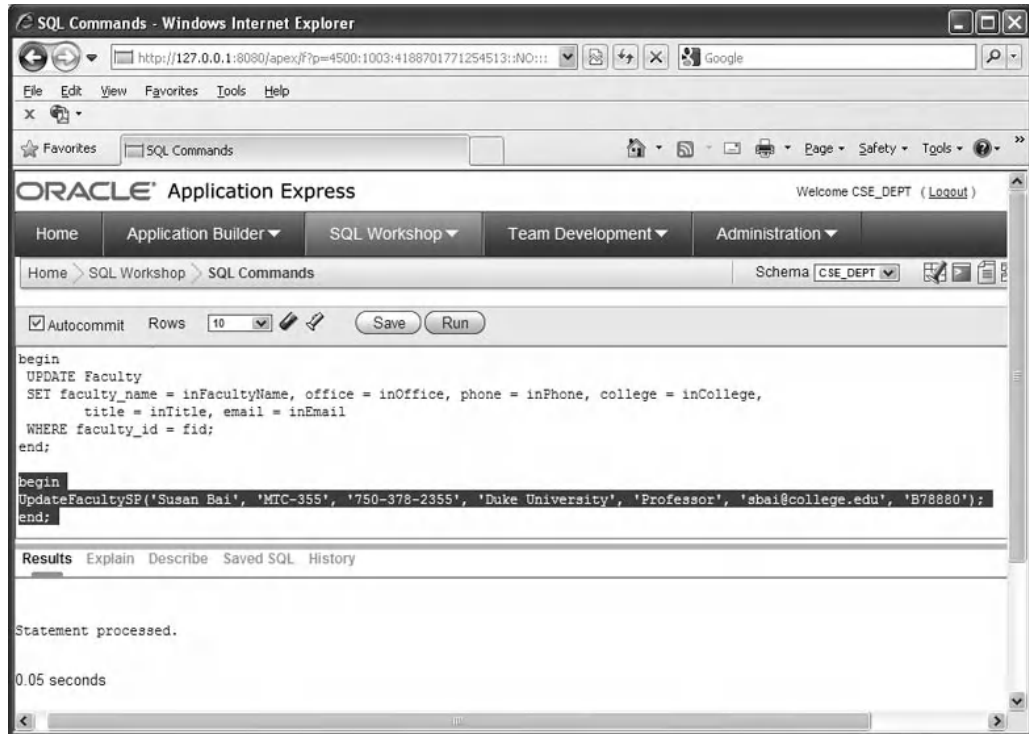


Figure 7.33. The running result of the stored procedure UpdateFacultySP.

**Statement processed.
0.05 seconds**

Click on the **Save** button to save this stored procedure. Enter **UpdateFacultySP** into the Name box and click on the **Save** button to complete this saving action.

Now let's open our Faculty table to confirm that the selected row has been updated after the stored procedure **UpdateFacultySP** is executed. Click on the **SQL Workshop** button that is located at the third button in the top row. Then click on the Object Browser icon to open the Object Browser wizard. Keep the **Tables** item selected in the top and double-click on the Faculty item, and then click on the **Data** tab to open this table.

The opened Faculty table is shown in Figure 7.34.

It can be found that the first row, which is pointed by the arrow, 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 Commands wizard. Click on the **SQL Workshop** button that is located at the third button in the top row. Then click on the SQL Commands icon to open the SQL Commands wizard. Enter the codes that are shown in Figure 7.35 into this page as the body of our 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 that a message is displayed in the bottom pane in the **Results** tab to indicate that the stored procedure is created, which is shown below:

The screenshot shows a web browser window titled 'Object Browser - Windows Internet Explorer'. The address bar shows a URL starting with 'http://127.0.0.1:8080/...'. The browser displays the 'FACULTY' table with the following data:

EDIT	FACULTY_ID	FACULTY_NAME	OFFICE	PHONE	COLLEGE	TITLE	EMAIL
	B78880	Susan Bai	MTC-355	750-378-2355	Duke University	Professor	sbai@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
	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
	B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu

At the bottom right of the table, it says 'row(s) 1 - 8 of 8'.

Figure 7.34. The updated Faculty table.

```

create or replace PROCEDURE DeleteFacultySP
(fame IN VARCHAR2) AS
begin
  DELETE FROM Faculty WHERE faculty_name = fame;
end;

```

Figure 7.35. The code body of the data deleting stored procedure.

Procedure created.
0.74 seconds

Click on the **Save** button to save this stored procedure. Enter **DeleteFacultySP** into the **Name** box and click on the **Save** button to complete this saving action. Because of the complexity in recovering this faculty record, we would not test this stored procedure in this SQL Commands wizard.

Before we can close the Oracle Database 11g XE wizard, it is highly recommended to recover the Faculty table to its original value. To do that, click on the **SQL Workshop** button that is located at the third button in the top row. Then click on the **Object Browser** icon to open the Object Browser wizard. Keep the **Tables** item selected in the top and double-click on the **Faculty** item, and then click on the **Data** tab to open this table.

Click on the **Edit** icon before the first row to open the Edit Row wizard. Then enter the original faculty data into the associated box to recover this row:

- Ying Bai Faculty Name box
- MTC-211 Office box
- 750-378-1148 Phone box

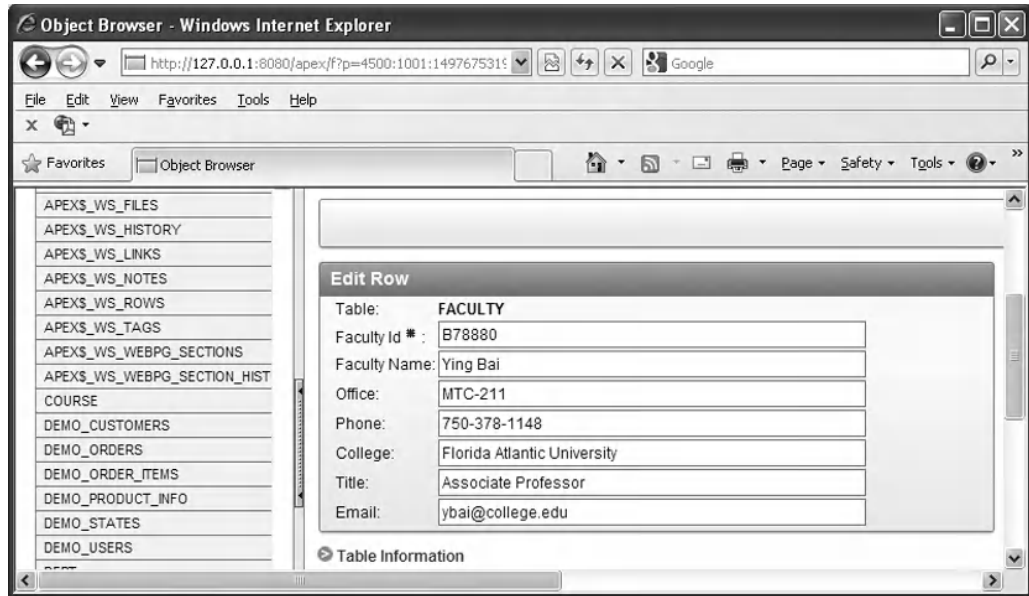


Figure 7.36. The recovered faculty record.

- Florida Atlantic University College box
- Associate Professor Title box
- ybai@college.edu Email box

Your recovered faculty record for Ying Bai is shown in Figure 7.36. Click on the Apply Changes button to save this recovered record into the Faculty table.

Now Close the Oracle Database 11g XE after these stored procedures are created. Next, we need to call these stored procedures from the Visual Basic.NET project to perform the data updating and deleting functions.

7.8.2.4 Call the Stored Procedure to Perform the Data Updating and Deleting

Since we have finished the modifications to our new project in Section 7.8.2.2, 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 OracleUpdateDeleteSP, 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. Keep the default faculty name Ying Bai selected from the Faculty Name combo box. Then click on the Select button to query and display all pieces of information for the selected faculty.

To update this faculty record, enter each piece of updating information into the associated textbox to perform this data updating:

- Tailor 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
- Default.jpg Faculty image textbox

Click on the **Update** button to call the stored procedure **UpdateFacultySP** to update this faculty record in the Faculty table in our sample database.

To confirm this data updating, first, let's select any other faculty from the Faculty Name combo box and click on the **Select** button to query and display each piece of information for the selected faculty. Then select our newly updated faculty **Tailor Bai** from the Faculty Name combo box. Click on the **Select** button to retrieve that updated faculty record from the database, and display it in this form. Immediately, you can find that the updated faculty record is returned and displayed, which is shown in Figure 7.37.

Next, let's test the data deleting action by clicking on the **Delete** button to try to delete this updated record. Click on **Yes** button on the MessageBox to confirm this faculty record deleting. Immediately, all pieces of information stored in seven textboxes are removed. To confirm this data deleting, click on the **Select** button to try to retrieve the deleted faculty record. 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 Oracle database are very successful.

The screenshot shows a Windows-style application window titled "CSE DEPT Faculty Form". It contains several input fields and buttons. On the left, there is a "Faculty Image" section with a text box containing "Default.jpg" and a placeholder image of a person. To the right, there is a "Faculty Name_Query Method" section with a "Faculty Name" dropdown menu set to "Tailor Bai" and a "Query Method" dropdown menu set to "TableAdapter Method". Below these is a "Faculty Information" section with text boxes for "Faculty ID" (878880), "Name" (Tailor Bai), "Title" (Distinguished Professor), "Office" (MTC-228), "Phone" (750-378-1222), "College" (University of Miami), and "Email" (tbai@college.edu). At the bottom of the form are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 7.37. The confirmation of the updated faculty record.

In order to keep our database neat and complete, refer to Tables 7.7–7.10 in Section 7.8.1.3 to recover those deleted records one by one using the **Insert Row** button in the Object Browser wizard. The point is the order to recover these deleted records and the value for NULL columns. The correct order and NULL values are:

1. Recover the deleted faculty record from the Faculty table since it is a parent table.
2. Recover all other deleted records for all other related child tables, such as the Course, the LogIn, and the StudentCourse tables.
3. Leave a blank for any NULL column, such as the `student_id` in the LogIn table.

A complete project `OracleUpdateDeleteSP` can be found in the folder `DBProjects\Chapter 7` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

7.8.3 Update and Delete Data Against Databases Using the 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, updating, and deleting actions against our sample database `CSE_DEPT.mdf` using the LINQ to SQL query. We leave the data insertion coding process as a homework for the readers. Refer to Section 4.6.2.2 in Chapter 4 to complete the codes for this **Insert** button's Click event procedure. Now let's perform the following steps to create our new project `LINQSQLQuery`:

1. Create a new Visual Basic.NET Windows-based project and name it as `LINQSQLQuery`.
2. Change the File Name to `Faculty Form.vb`.
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 in the project `SQLUpdateDeleteSP` we developed in this Chapter and paste them into this new form.
5. Add the Imports `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 scrolling down the .NET list. Select the item `System.Data.Linq` from the list and click on the OK button.

6. Add the following directives at the top of the Faculty Form window:
 - Imports System.Data.Linq
 - Imports System.Data.Linq.Mapping
7. Follow 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. 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 Visual Basic.NET code file for our sample database, CSE_DEPT.vb.

Now let's begin the coding process for this project. Since we need to use the **Select** button's Click event procedure 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 codes.
2. Develop the codes for the **Select** button's Click event procedure to retrieve the selected faculty record using the LINQ to SQL query.
3. Develop the codes for the **Update** button's Click event procedure to update the selected faculty member using the LINQ to SQL query.
4. Develop the codes for the **Delete** button's Click event procedure to delete the selected faculty member using the LINQ to SQL query.

Now let's start from Part I.

7.8.3.1 Create a New Object of the 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 codes includes retrieving all updated faculty members from the Faculty table in our sample database using the LINQ to SQL query and displaying them in the Faculty Name combo box control.

Open the Form_Load() event procedure of the Faculty Form window, and enter the codes that are shown in Figure 7.38 into this procedure.

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

- A. Two namespaces related to LINQ To SQL are imported since we need to use some components stored in that namespace to perform the data queries.
- B. A new form 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.

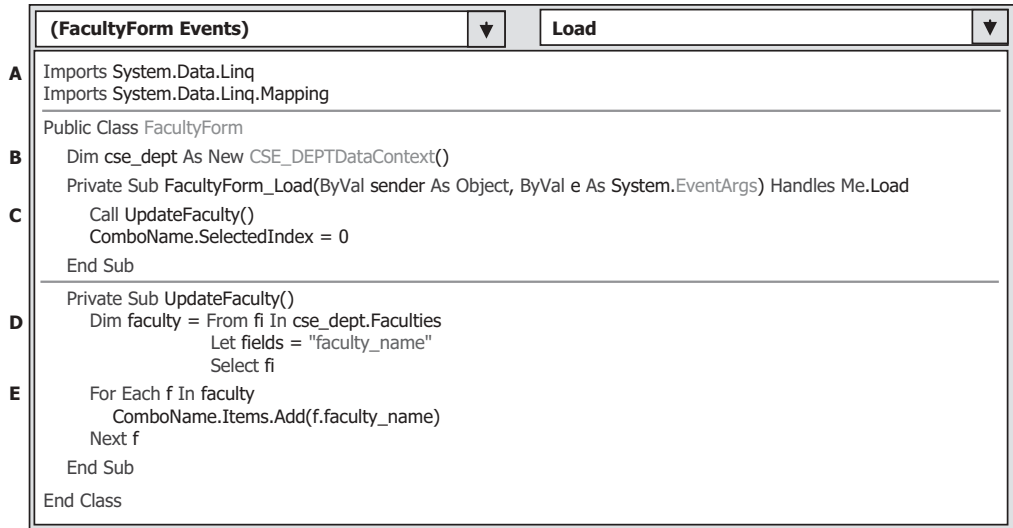


Figure 7.38. Initialization codes for the Form_Load event procedure of the Faculty Form.

- C.** A user -defined subroutine procedure **UpdateFaculty()** is executed to retrieve all updated faculty members from our sample database and display them in the Faculty Name combo box control to allow the user to select a desired faculty.
- 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.
- E.** The LINQ query is executed to pick up all queried faculty members and add them into the Faculty Name combo box control in our Faculty Form.

Let's continue to develop the codes for the second part.

7.8.3.2 Develop the Codes for the Select Button Click Event Procedure

Double-click on the **Select** button to open its Click event procedure and enter the codes that are shown in Figure 7.39 into this procedure. The function of this piece of codes is to retrieve the detailed information for the selected faculty member from the Faculty table in our sample database and display them in the associated textbox control in the Faculty Form window as this **Select** button is clicked by the user.

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

- A.** The user defined subroutine **ShowFaculty()** is executed to identify and display a matched faculty image for the selected faculty member. You can copy the codes for this subroutine from the Faculty Form in the project **SQLUpdateDeleteSP** we developed in this chapter and paste them into this code window.
- 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**).

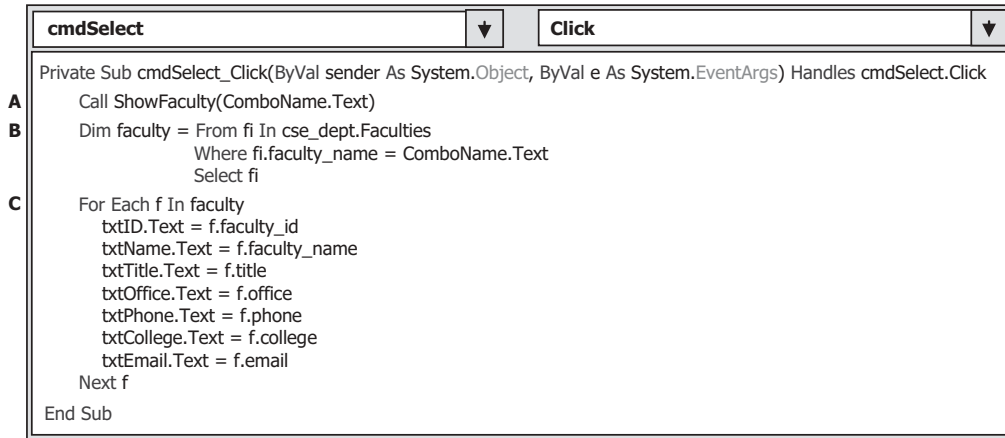


Figure 7.39. The codes for the Select button Click event procedure.

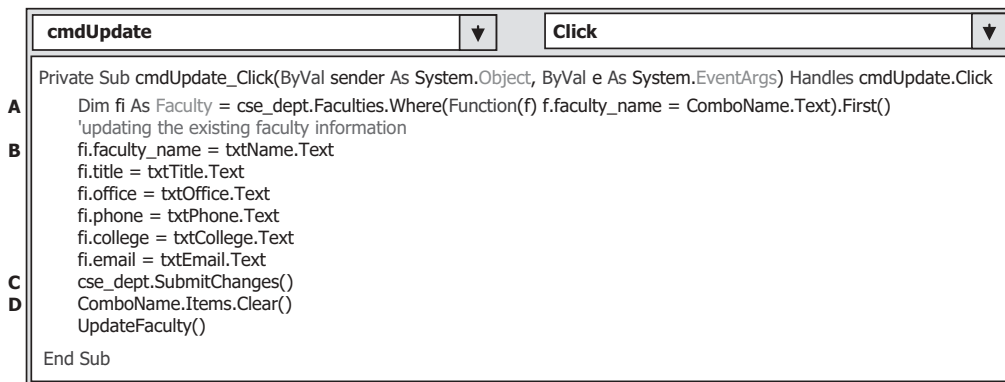


Figure 7.40. The codes for the Update button Click event procedure.

- C.** The LINQ query is executed to pick up all columns for the selected faculty member and display them on the associated textbox in this Faculty Form.

Now let's concentrate on the coding process for our data updating and deleting actions.

7.8.3.3 Develop the Codes for the Update Button Click Event Procedure

Double-click on the Update button from our Faculty Form window to open its Click event procedure and enter the codes that are shown in Figure 7.40 into this procedure.

Let's have a closer look at this piece of codes 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 matched record. It does not matter for our application since we have only one record that is associated with this specified `faculty_name`.

- B. All six columns, except the `faculty_id`, for the selected faculty record, 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 cannot really occur until the `SubmitChanges()` method is executed.
- D. The Faculty Name combo box is cleaned up, and the user-defined subroutine procedure `UpdateFaculty()` is executed to refresh the updated faculty members stored in that control.

Next, let's perform the coding development for the data deleting action. This coding process is similar to that of data updating we did in this part.

7.8.3.4 Develop the Codes for the Delete Button Click Event Procedure

Double-click on the Delete button from the Faculty Form window to open its Click event procedure and enter the codes that are shown in Figure 7.41 into this procedure.

Let's have a closer look at this piece of codes 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 member (`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 to our database.
- D. All TextBoxes stored information related to the deleted faculty are cleaned up by assigning an empty string to each of them.

The codes for the Back button Click event procedure is simple. Just enter `Me.Close()` into that event procedure.

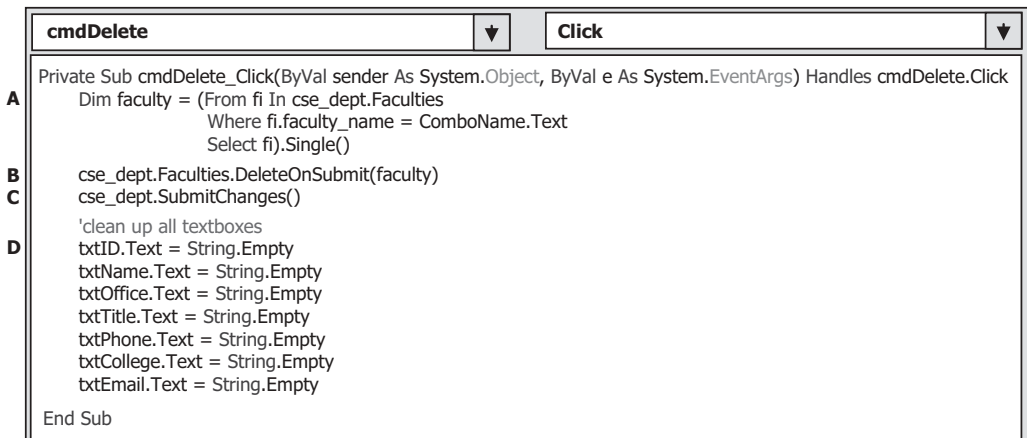


Figure 7.41. The codes for the Delete button Click event procedure.

Now we can build and run our project to test the data updating and deleting action 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 executable file `LINQSQLQuery.exe` is located. In this application, it should be: `C:\Chapter 7\LINQSQLQuery\bin\Debug`.

Copy all faculty image files from the folder `Images` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1) and paste them into the folder shown above.

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
- 750-378-1220 Phone textbox
- University of Main College textbox
- pbai@college.edu Email textbox
- Default.jpg Faculty Image textbox

To confirm this data updating, you need to find the updated faculty name from the Faculty Name combo box and click on the **Select** button to try to retrieve the updated faculty record and display it in this form. Remember, you need to recover those updated pieces of 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
- (Keep empty) Faculty Image textbox

Now let's test the deleting function by clicking on the **Delete** button. Click on the **Yes** button to confirm and perform this data deletion. To confirm this deleting action, click on the **Back** button to terminate our project. Then run the project again and you can find that the faculty member Ying Bai cannot be found in the Faculty Name combo box, which means that the selected faculty member has been deleted from our database.

Our data updating and deleting actions using the LINQ to SQL in the SQL Server database are very successful.

In order to keep our database neat and complete, refer to Tables 7.7–7.10 in Section 7.8.1.3 to recover those deleted records one by one using the SQL Server Management Studio. The point is the order to recover these deleted records and the value for NULL columns. The correct order and NULL values are:

1. Recover the deleted faculty record from the Faculty table since it is a parent table.
2. Recover all other deleted records for all other related child tables, such as the Course, the LogIn, and the StudentCourse tables.
3. Leave a blank for any NULL column, such as the `student_id` in the LogIn table.

A complete project `LINQSQLQuery` can be found in the folder `DBProjects\Chapter 7` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

7.9 CHAPTER SUMMARY

Data updating and deleting queries are discussed in this chapter with two popular databases: SQL Server and Oracle.

Five popular data updating and deleting methods are discussed and analyzed with eight real project examples:

1. Using `TableAdapter.DBDirect` methods, such as `TableAdapter.Update()` and `TableAdapter.Delete()`, to update and delete data directly against the databases.
2. Using `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. Using the runtime object method to develop and execute the `Command`'s method `ExecuteNonQuery()` to update or delete data against the database directly.
4. Using the stored procedures to update or delete data against the database directly.
5. Using 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 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 Basic.NET applications. The difference between method 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, but 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 Visual Studio.NET design tools and wizards. Part II presents the data updating and deleting using the runtime object method to develop command objects to execute the `ExecuteNonQuery()` method dynamically. Updating and deleting data against our sample database using the stored procedures and 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 database: Microsoft Access, SQL Server, and Oracle databases. The stored procedures and LINQ to SQL methods are discussed in the last section 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, TableAdapter.Update(), and ExecuteNonQuery() method of the Command class.
- ___2. Unlike the Fill() method, a valid database connection must be set before a 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 new stored procedure in the Server Explorer window.
- ___10. To call stored procedures to update data against 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 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. Either of above
4. Because 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 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 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

```

CREATE OR REPLACE PROCEDURE dbo.UpdateStudent
( @Name IN VARCHAR(20),
  @Major IN text,
  @SchoolYear IN int,
  @Credits IN float,
  @Email IN text
  @StudentID IN VARCHAR(20))
AS
UPDATE Student SET student_name=@Name, major=@Major, schoolYear=@SchoolYear,
credits=@Credits, email=@Email
WHERE (student_id=@StudentID)
RETURN

```

Figure 7.42. A SQL Server stored procedure.

```

Dim cmdString As String = "UpdateCourse"
Dim intInsert As Integer
Dim oraCommand As 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 7.43. A piece of VB codes.

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

- Figure 7.42 shows a stored procedure developed in the SQL Server database. Please develop a piece of codes in Visual Basic.NET to call this stored procedure to update a record in the database.
- Figure 7.43 shows a piece of codes developed in Visual Basic.NET, and this piece of codes 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.
- 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 can

be found the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

4. Using the Runtime objects to complete the update data query for the Student form by using the project **SQLUpdateDeleteRTOObject**. The project file can be found in the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in 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 **OracleUpdateRTOObjectSP**. The project file can be found the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in 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 **OracleUpdateRTOObjectSP**. The project file can be found in the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). It is highly recommended to recover those deleted records after they are deleted).

Hints: You need to delete the related records from the **LogIn**, **Course**, and the **StudentCourse** tables, and then delete record from the **Student** table.

Chapter 8

Accessing Data in ASP.NET

We have provided a very detailed discussion on database programming with Visual Basic.NET using the Windows-based applications in the previous chapters. Starting from this chapter, we will concentrate on the database programming with Visual Basic.NET using Web-based applications. To develop a 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 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, 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 understand 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 a foundation of the ASP.NET. So we need first to give a detailed discussion about the .NET Framework.

8.1 WHAT IS THE .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 Basic.NET or Visual C#.NET.

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 in developing 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. MSIL can be recognized by CLR, and CLR can compile and execute the MSIL codes by using the Just-In-Time compiler.

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.

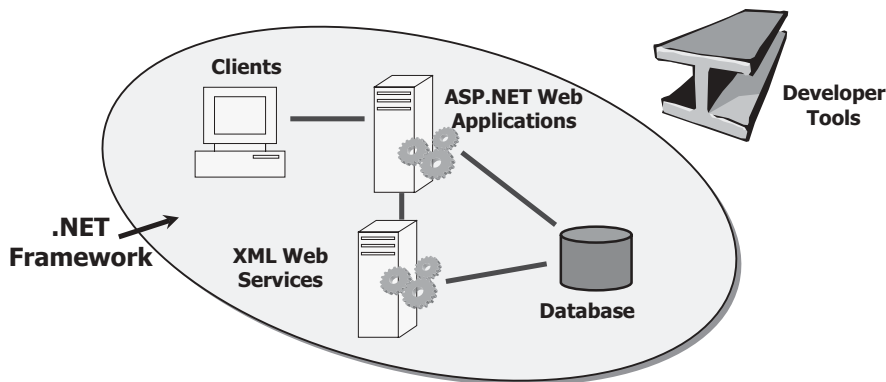


Figure 8.1. The .NET Framework model.

The .NET Framework supports three types of user interfaces:

- 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.

Summarily, the advantages of using the .NET Framework to develop Windows-based and Web-based applications include, but no limited to:

- The .NET Framework is based on Web standards and practices, and it fully supports Internet technologies, including the 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 functionality 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 have a closer look at the ASP.NET.

8.2 WHAT IS ASP.NET?

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 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.

A completed structure of an ASP.NET Web application is shown in Figure 8.2.

Unlike the 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.

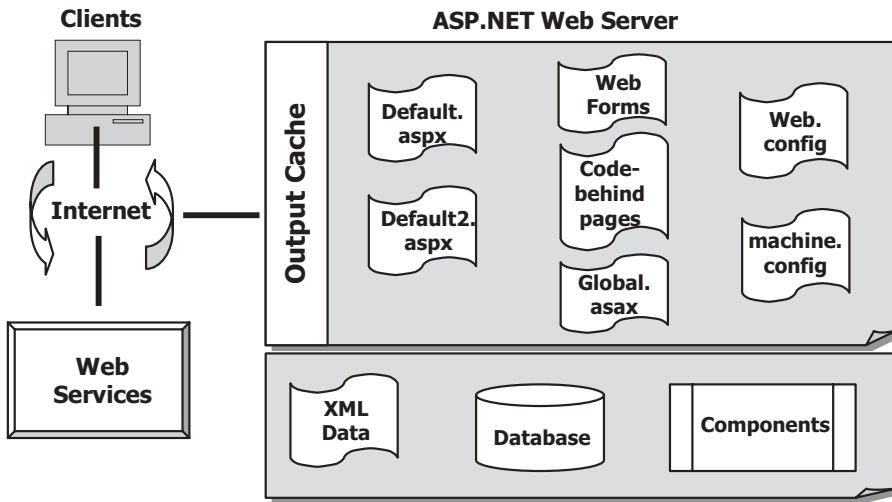


Figure 8.2. The structure of an ASP.NET Web application.

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 procedures 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.vb.
- 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 that is designed by the extension of .asmx.vb.
- 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 .asax file. The Global.asax 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.asax 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 the security issues; instead, in most industrial

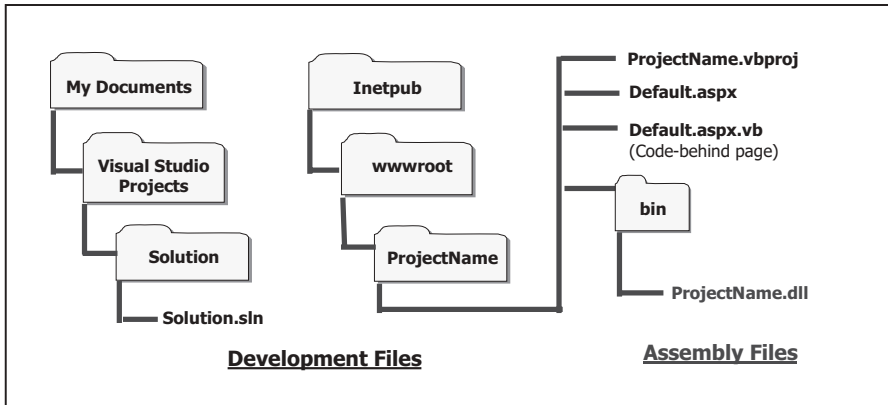


Figure 8.3. ASP.NET Web application file structure.

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 are related 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 (dll) file.

Figure 8.3 shows a typical file structure of an ASP.NET Web application.

The folders listed 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 created on the right side are new to us, and the functionalities of those folders are:

- 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.vbproj** 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.vb) 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 website and will be executed by the Web server as a request is received from the client at the running time.
2. **AssemblyInfo.vb file.** This file is used to write the general information, especially assembly version and assembly attributes, about the assembly.

As the 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 source 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 MSIL code and send 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 that the execution or running of a Web application is easy and straightforward, but in practice, a lot of data round trips occurred between the client and the server. To make it clear, let's make a little more analysis to see what happened 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 sends 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. When the user sends a request to the server to ask to start a logon process for the first time, 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 received the LogIn Web page and entered 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. If, after the server received and processed the logon information, the server found that the logon information is invalid, the server needs to rebuild the LogIn form and resend it back to the client with some warning message. So 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 those round trips is to make sure that all information entered from the client side should be as correct as possible. In other words, try to make as much validation as possible in 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 The Requirements to Test and Run the 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 order to make our Web project simple and easy, we always use our local computer as a local server. 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 start, then click on Control Panel, and click on Add or Remove Programs.
- Click 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 FrontPage 2000 Server Extensions, click on the Details button. Check the item FrontPage 2000 Server Extensions to select it. Although Microsoft has stopped supporting to this version of the server, and the current version is FrontPage 2002 Server Extensions, you can still use it.

Click on the OK and the Next button to begin to install the IIS and the FrontPage 2000 Server Extensions to your computer. You may need the system OS CD/DVD to finish this installation.

As we 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 both the SQL Server and the Oracle database belong to the 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 from the database through the Web pages.

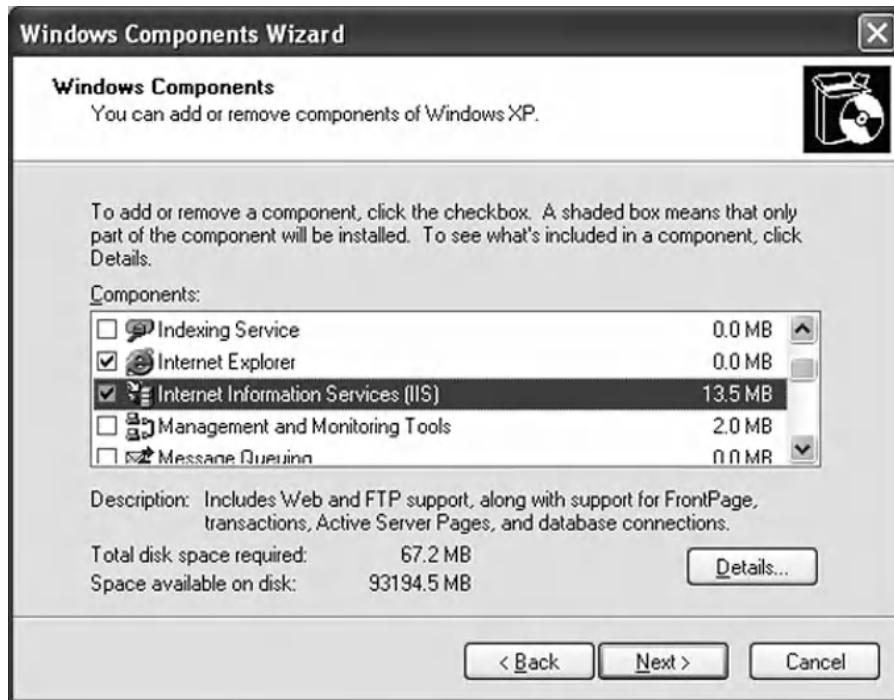


Figure 8.4. The opened Windows Components wizard.

This chapter is organized in the following ways:

1. Develop ASP.NET Web application to select and display data from the Microsoft SQL Server database.
2. Develop an ASP.NET Web application to insert data into the Microsoft SQL Server database.
3. Develop an ASP.NET Web application to update and delete data against the Microsoft SQL Server database.
4. Develop an 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 an ASP.NET Web application to insert data into the Oracle database.
7. Develop an ASP.NET Web application to update and delete data against the Oracle database.

Let's start from the first one 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.

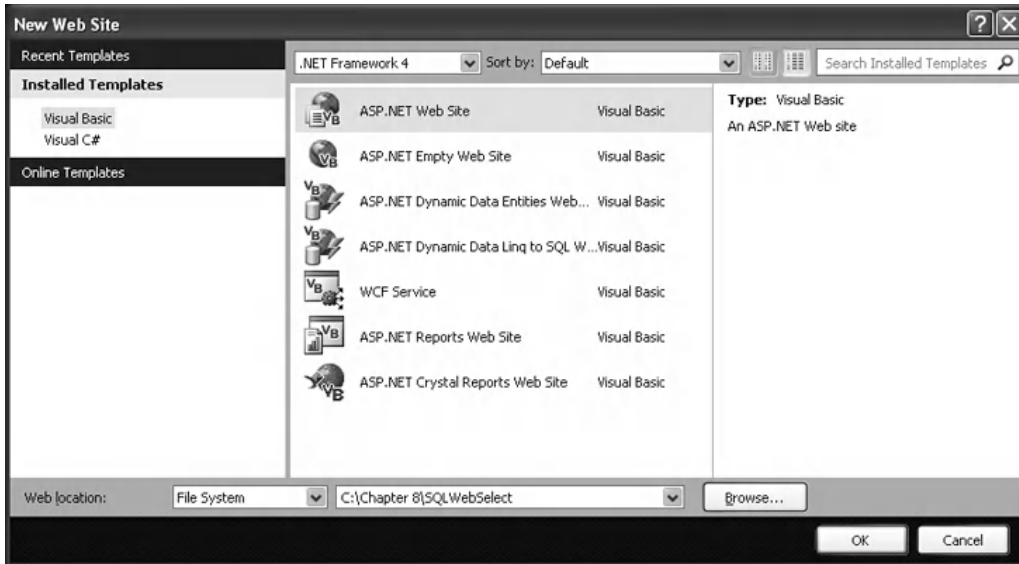


Figure 8.5. The opened Template wizard.

NET and click on the File|New Web Site to create a new ASP.NET Web application project.

On the opened New Web Site wizard, which is shown in Figure 8.5, keep the default template **ASP.NET Web Site** selected. Also, keep the default **Language** Visual Basic unchanged. Click on the **Browse** button to browse to any location or folder where you want to save this project to. In fact, you can place your new project in any folder as you like in your computer. In our case, we place it in our folder: **C:\Chapter 8**. Browse to that folder and click on the **Open** button. Attach our Web project **SQLWebSelect** after the folder **C:\Chapter 8** in the **Web Location** box. Your finished New Web Site wizard is shown in Figure 8.5. Click on the **OK** button to continue.

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 in the server-side. Now let's perform some modifications to this form to make it as our **LogIn** form.

8.3.1 Create the User Interface: LogIn Form

Right-click on the **Default.aspx** item and select the **Rename** item from the pop-up menu to change the name of this Web Form to **LogIn.aspx**. Then we need to perform the following modifications to the Source file to make it as our **LogIn** Form page:

1. Open the Source file by clicking on the **Source** tab on the bottom of the window. Move the cursor to the end of the second line and change the value of the **Inherits** item from **Inherits = "_Default"** to **Inherits = "_LogIn"** (do not worry if a blue underscore appears; we can fix this in the next step).

2. Double-click on the `LogIn.aspx.vb` from the Solution Explorer window to open the code-behind page. Change the class name from `_Default` to `_LogIn`.
3. Go to **Build/Rebuild Web Site** to build our project.

The Source file basically is an HTML file that contains the related codes for all controls you added into this Web form in the HTML format. 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 Basic.NET code format.

Now let's click on the View Designer button from the Solution Explorer window to open and design our LogIn Web page.

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 make this style such as the header and footer of the form apply 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 remove all default contents from this page and add the controls shown in Table 8.1 into our LogIn user interface or Web page. One point to be noticed is 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 as you want 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 that is shown in Figure 8.6.

Before we can add the codes into the code-behind page to respond to the controls to perform the logon process, first, we must run the project to allow the `Web.config` file

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. The finished LogIn Web form.

to recognize those controls we have added into the Web form. Click the Start Debugging button on the toolbar to run our project. Click on the OK button to a prompted MessageBox to add a Web.config file with the debugging enabled as the project runs. Your running Web page should match the one that is shown in Figure 8.6. Click on the Close button that is 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 the Codes to Access and Select Data from the Database

Open the code-behind page by clicking on the View Code button from the Solution Explorer window. First, we need to add two imports commands as we did for those projects in the previous chapters. Add the following two commands to the top of this code window to import the namespace of the SQL Server Data Provider:

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

Next, we need to create a global variable, `sqlConnection`, for our connection object. Enter the following code under the class header:

```
Public sqlConnection As SqlConnection
```

This connection object will be used by all Web forms in this project.

Now we need to develop the codes for the `Page_Load()` event procedure, which is similar to the `Form_Load()` event procedure in the Windows-based application. Go to the Class Name combo box and select the (Page Events) item, and select the Load item

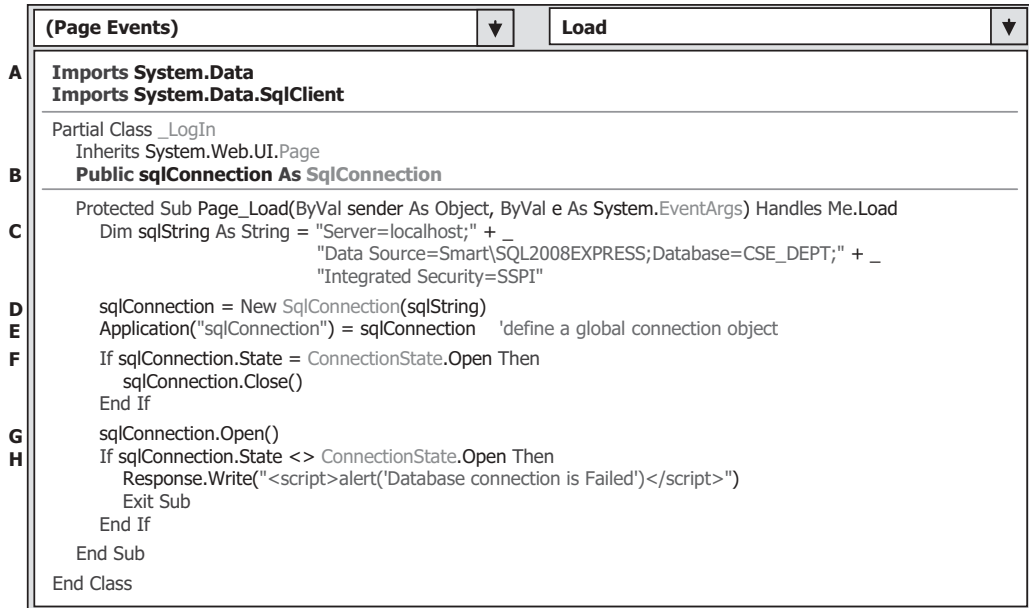


Figure 8.7. The codes for the Page_Load event procedure.

from the Method Name combo box to open this event procedure. Enter the codes that are shown in Figure 8.7 into this event procedure.

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

- A.** First, the namespaces for the SQL Server Data Provider are imported since we need to use some components defined in those namespaces to perform data actions in this page.
- B.** A global connection object `sqlConnection` is declared first, and this object will be used by all Web forms in this project later to connect to the database.
- C.** As we did for the `Form_Load()` event procedure in the Windows-based applications, we need to perform the database connection in this event procedure. 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 global 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 defined in the Windows-based applications, one cannot access a global variable by prefixing the form's name before the global 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 that 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 is, 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 whether the connection is successful or not. 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 debug 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 develop the codes for the **LogIn** button's Click event procedure. The function of this piece of codes 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 successful. Click on the View Design button from the Solution Explorer window and then double-click on the **LogIn** button to open its event procedure. Enter the codes that are shown in Figure 8.8 into this event procedure.

Let's have a closer look at this piece of codes 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. Since this query statement is relatively long, we split it into two substrings. Of course, you can use the concatenating operator "&" to make these two strings as one if you like.
- B. Some data objects are created here such as the **Command** object, **DataReader** object, and **Parameter** objects.
- C. Then two parameter objects are initialized with the parameter's name and value properties. The **Command** object is built by assigning it with the **Connection** object, **commandType**, and **Parameters** collection properties of the **Command** class.
- D. The `Add()` method is utilized to add two actual parameters to the **Parameters** collection of the **Command** class.
- E. 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.
- F. If the **HasRows** property of the **DataReader** is **True**, this means that 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 in the client browser.
- G. Otherwise, no matched username or password found in the database, and a warning message is created and sent back to the client and displayed in the client browser.
- H. The used objects such as the **Command** and the **DataReader** are released.
- I. Since we have not created any other pages, at this moment, we temporarily close the connection between our page and the database. In our formal application, this connection

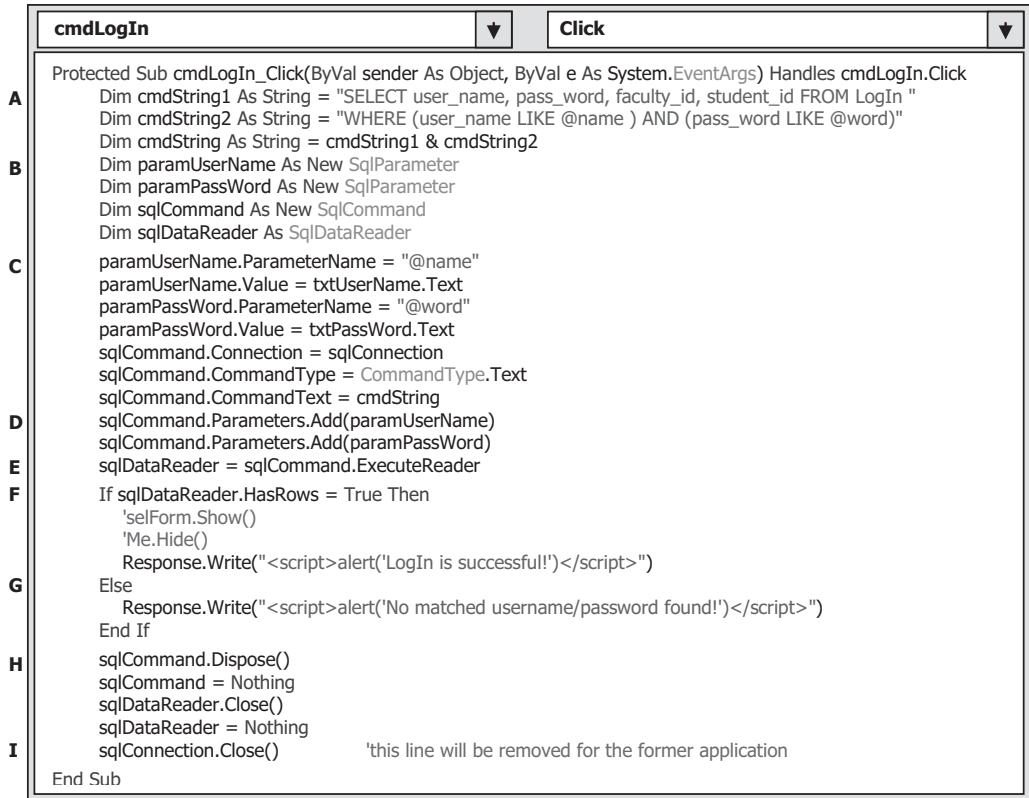


Figure 8.8. The codes for the LogIn button's Click event procedure.

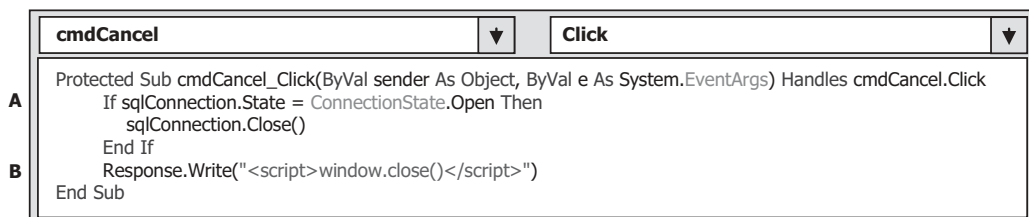


Figure 8.9. The codes for the Cancel button's event procedure.

should be open until the Exit button in the Selection Web form is clicked by the user as the project runs.

Next, let's develop the codes for the Cancel button's Click event procedure.

The function of this event procedure is to close the current Web page if this Cancel button is clicked, 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 event procedure and enter the codes shown in Figure 8.9 into this procedure.

The function of this piece of codes 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.

8.3.3 Validate the Data in the Client Side

In order to reduce the burden on the server, we should make every effort to perform the data validation in the client side. In other words, before we can send requests to the server, we need to make sure that our 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.

We want to use the first control, `RequiredFieldValidator`, to validate our two text boxes, `txtUserName` and `txtPassWord`, in the LogIn page to make sure that both of them are not empty when the LogIn button is clicked by the user as the project runs.

Open the Design View of the LogIn Web form, go to the Toolbox window, and expand the Validation tab. 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`

Table 8.2. Validation Controls

Validation Control	Functionality
<code>RequiredFieldValidator</code>	Validate whether the required field has valid data (not blank).
<code>RangeValidator</code>	Validate whether a value is within a given numeric range. The range is defined by the <code>MaximumValue</code> and <code>MinimumValue</code> properties provided by users.
<code>CompareValidator</code>	Validate whether a value fits a given expression by using different Operator properties, such as "equal", "greater than", "less than", and the type of the value, which is set by the Type property.
<code>CustomValidator</code>	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 send it to the server to get the feedback from it.
<code>RegularExpressionValidator</code>	Validate whether a value fits a given regular expression by using the <code>ValidationExpression</code> property, which should be provided by the user.



Figure 8.10. Adding the data validation: RequiredFieldValidator.

Perform the similar dragging and placing operations to locate 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 that is 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, directly 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 username and password, click on the `LogIn` button again, and a successful login message is displayed. So you can see how the `RequiredFieldValidator` works to reduce the processing load for the server.

One good thing always brings some bad things, 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 data. 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 the 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 the previous chapters, after the logon process, 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 against the database via the different tables by selecting the different items. So this Selection page needs to perform the following jobs:

- Provide and display all available selections to allow users to select them.
- 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 item 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, and then click on the **Add** button to add this page into our project.

On the opened Web form, add the controls shown 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 from the keyboard to locate the positions on the page for those controls. Your finished Selection page should match the one that is shown in Figure 8.11.

Table 8.3. Controls on 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. The finished Selection page.

Next, let's create the codes for this Selection page to allow users to select a different page to perform the associated data actions.

8.3.5 Develop the Codes to Open the 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 that is located on 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 two Imports commands to the top of this page to provide the namespace for the SQL Data Provider:

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

Then select the (Page Events) from the Class Name combo box and select the Load item from the Method Name combo box to open the Page_Load event procedure. Enter the codes that are shown in Figure 8.12 into this event procedure to add all selection items into the combo box control **ComboSelection**.

The function of this piece of codes is straightforward. Three pieces of information are added into the combo box **ComboSelection** by using the **Add()** method, and these pieces of information will be selected by the user as the project runs.

Next, we need to create the codes for two buttons' Click event procedures. First, let's develop the codes 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 event procedure. Enter the codes that are shown in Figure 8.13 into this event procedure.

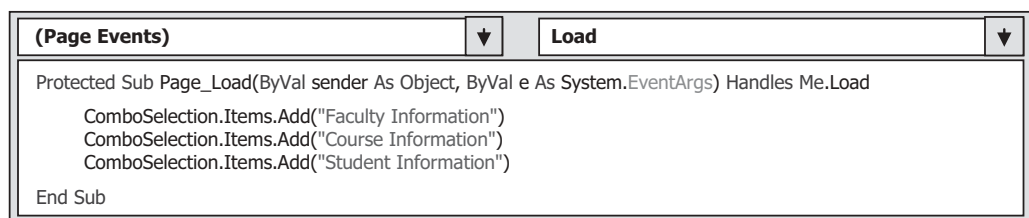


Figure 8.12. The codes for the Page_Load event procedure of the Selection page.

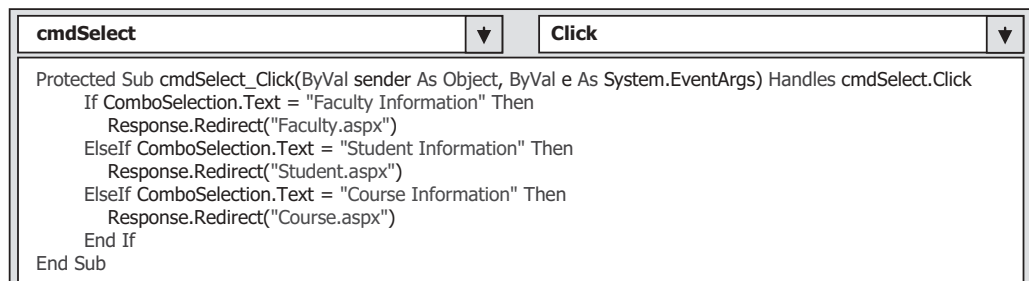


Figure 8.13. The codes for the Select button's event procedure.

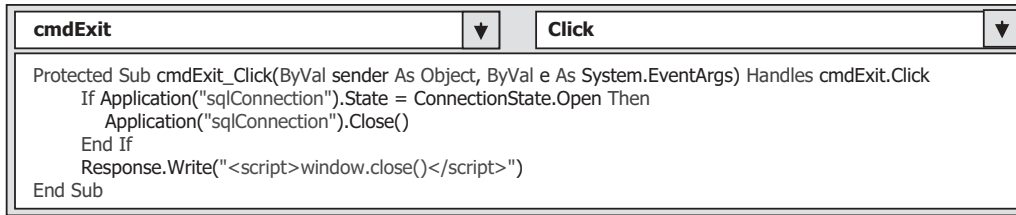


Figure 8.14. The codes for the Exit button's Click event procedure.

The function of this piece of codes is easy. Based on the information selected by the user, the related Web page is opened by using the server's Response object by using the `Redirect()` method of the server's Response object. All of these three pages will be created and discussed in the following sections.

Finally, let's take care of the coding process for the Exit button's Click event procedure. The function of this piece of codes 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 event procedure. Enter the codes that are shown in Figure 8.14 into this event procedure.

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.

8.3.6 Modify the Codes in the LogIn Page to Transfer to the Selection Page

Now we have finished the coding process for the Selection page. Before we can run the project to test this page, we need to do some modifications to the codes in the LogIn button's Click event procedure 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 event procedure, and replace the code line that is located inside the If block:

```
Response.Write("<script>alert('LogIn is successful!')</script>")
```

with the following code line:

```
Response.Redirect("Selection.aspx")
```

Also, remove the last code line `sqlConnection.Close()` from this event procedure since we need this connection opened during our project runs.

In this way, as long as the login 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 of the Selection page. Since the Selection page is located at the same application as the LogIn page does, a direct page name is used.

Now let's run the application to test these 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 on the Start Debugging button to run our project.



Figure 8.15. The running status of the second page: Selection page.

Enter a suitable username and password, such as `jhenry` and `test`, into the username and password boxes, and click on the **LogIn** button. The Selection page is displayed if this login process is successful, as shown in Figure 8.15.

Click on the **Exit** button to close the application. Now let's begin to develop our next page, Faculty page.

8.3.7 Create the 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 wizard, keep the default template **Web Form** selected and enter `Faculty.aspx` into the **Name** box as the name for this new page, and then click on the **Add** button to add this new page into our project.

On the opened Web form, add the controls that are shown in Table 8.4 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 from the keyboard to locate the positions on the page for those controls. You cannot 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.

Now you can enlarge this **Image** and place it in the left on this page by dragging it to that position.

Two important points to be noted when building this page are:

1. After drag and place the **Image** control into this page, go to the Properties window and set the **ImageAlign** property to **Left**.
2. Click on any place in this page and go to the Properties window. Select the **Style** property and click on the expansion button to open the **Modify Style** wizard. Click on the **Position** item from the left pane and select the **absolute** item from the **position** combo box in the right pane.
3. Click on the **OK** button to complete these setups.

Table 8.4. Controls on the Faculty form

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	CSE_DEPT Faculty Page	0	#E0E0E0	Bold/Large
Label	Label2	Faculty Image	1		Bold/Small
TextBox	txtImage		2		
Label	Label3	Faculty Name	3		Bold/Small
DropDownList	ComboName		4		
Image	PhotoBox		24		
Label	Label4	Faculty ID	5		Bold/Small
TextBox	txtID		6		
Label	Label5	Name	7		Bold/Small
TextBox	txtName		8		
Label	Label6	Title	9		Bold/Small
TextBox	txtTitle		10		
Label	Label7	Office	11		Bold/Small
TextBox	txtOffice		12		
Label	Label8	Phone	13		Bold/Small
TextBox	txtPhone		14		
Label	Label9	College	15		Bold/Small
TextBox	txtCollege		16		
Label	Label10	Email	17		Bold/Small
TextBox	txtEmail		18		
Button	cmdSelect	Select	19		Bold/Medium
Button	cmdInsert	Insert	20		Bold/Medium
Button	cmdUpdate	Update	21		Bold/Medium
Button	cmdDelete	Delete	22		Bold/Medium
Button	cmdBack	Back	23		Bold/Medium

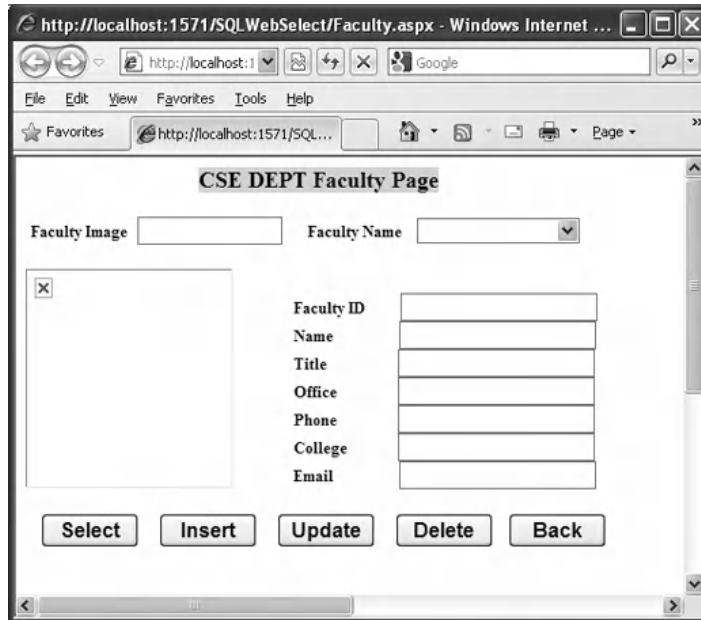


Figure 8.16. The finished Faculty page.

Your finished Faculty page should match the one that is shown in Figure 8.16.

An easy way to build this Faculty page is to add an existing Faculty page `Faculty.aspx` that can be found in the folder `VB Forms\Web` in the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Perform the following operations to add this existing Faculty page into our project:

1. Right-click on our current project from the Solution Explorer window, and select the **Add Existing Item**.
2. Browse to the folder `VB Forms\Web` in the Wiley ftp site and select the `Faculty.aspx` item. Then click on the **Add** button to add this page into our project. You can also save this page into a temporary folder in your computer and then perform this adding action.

Although we have added five buttons into this Faculty page, in this section, we only take care of the **Select** and the **Back** button since we want to discuss how to retrieve data based on the query command entered by the user from the database and display the retrieved result in this Faculty page. The other buttons will be used in the following sections later.

Now let's begin to develop the codes for the Faculty page to perform a data query from the Faculty table in our sample database.

8.3.8 Develop the Codes to Select the 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 that is 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, and add two Imports commands to the top of this code page:

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

The codes for this page can be divided into three parts: coding for the Page_Load() event procedure, coding for the Select button's Click event procedure, and coding for other procedures. First, let's take care of the coding for the Page_Load() event procedure.

8.3.8.1 Develop the Codes for the Page_Load Event Procedure

In the opened code page, open the Page_Load() event procedure by selecting the item (Page Events) from the Class Name combo box and the item Load from the Method Name combo box. Enter the codes that are shown in Figure 8.17 into this event procedure.

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

- A.** A form-level textbox array is created first since we need this array to hold seven pieces of faculty information and display them in seven textboxes later.
- B.** 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,

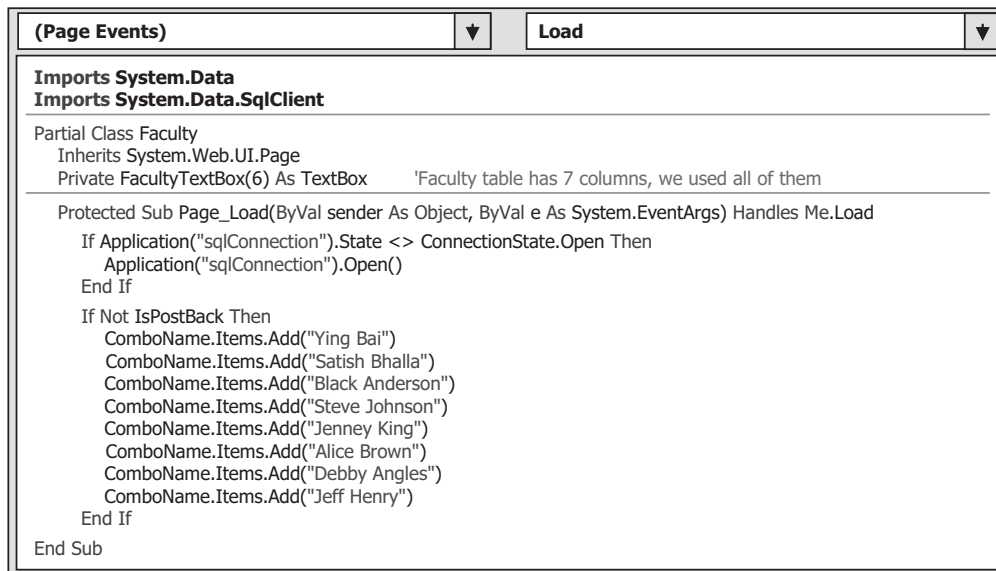


Figure 8.17. The codes for the Page_Load event procedure.

is utilized to perform this checking and connecting to our database if it has not been connected.

- C. As the project runs, each time the user clicks the **Select** button to perform a data query, a request is sent to the database server and the Web server (they can be the same server). Then, the Web server will post back a refreshed Faculty page to the client when it received this request (**IsPostBack = True**). When this happened, the **Page_Load** event procedure will be activated, and the duplicated eight faculty members are attached to the end of the Faculty Name combo box control again. To avoid this duplication, we need to check the **IsPostBack** property of the page and add eight faculty members into the Faculty Name combo box control only one time when the project starts (**IsPostBack = False**). Refer to Section 8.3.9.1 for more detailed discussion about the **AutoPostBack** property.

Next, we need to develop the codes for the **Select** button's Click event procedure to perform the data query against the database.

8.3.8.2 Develop the Codes for the Select Button Event Procedure

The function of this piece of codes is to make a query to the database to retrieve the faculty information based on the selected faculty member by the user from the Faculty Name combo box control, and display those pieces of retrieved information in seven textbox controls on the Faculty page.

Open this **Select** button's Click event procedure by double-clicking on this button from the Design View of the Faculty form, and enter the codes that are shown in Figure 8.18 into this event procedure.

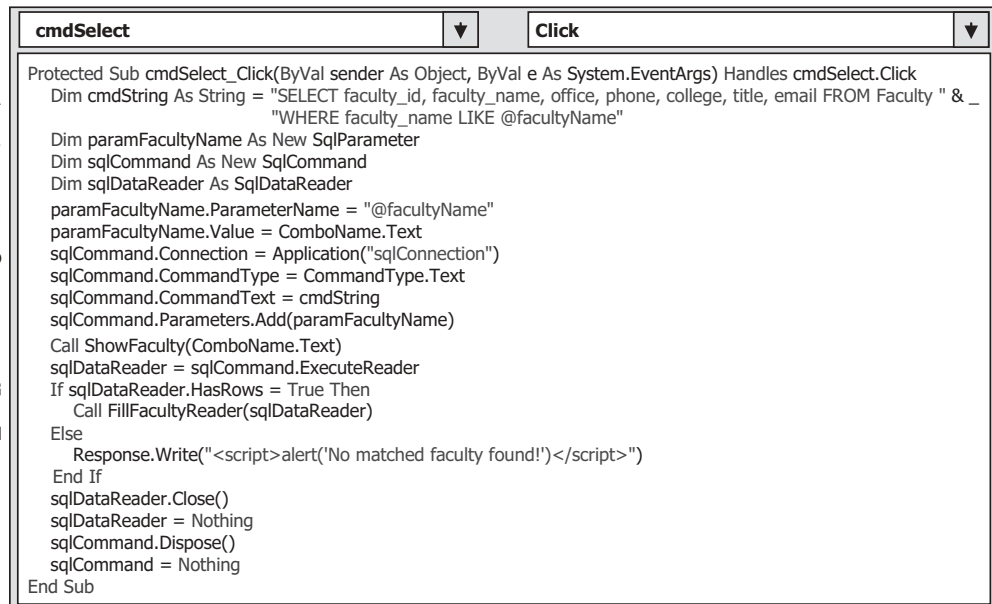


Figure 8.18. The codes for the Select button's Click event procedure.

Let's take a closer look at this piece of codes to see how it works.

- A. The query string that contains a **SELECT** statement is declared here since we need to use this as our command text. The dynamic parameter of this query is **facultyName**, defined in the **WHERE** clause.
- B. Some data components, such as the **Command**, **Parameter**, and **DataReader** objects, are declared here since we need to use them to perform the data query later.
- C. The **Parameter** object is initialized by assigning the dynamic parameter's name and value to it.
- D. 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, the **Parameters** collection object, and the **CommandType**, as well as the **CommandText** properties.
- E. The user-defined subroutine **ShowFaculty()** that will be developed later is called to display the selected faculty photo in the **Image** control on the **Faculty** page.
- 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 subroutine **FillFacultyReader()** is called to fill seven 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 and the query has failed. A warning message is displayed in 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 finished the codes for the **Select** button's **Click** event procedure.

8.3.8.3 Develop the Codes for Other Procedures

Next, let's take care of the coding process for other procedures in this **Faculty** page; this includes the coding process for the following procedures:

1. User-defined subroutine procedure **ShowFaculty()**.
2. User-defined subroutine procedure **FillFacultyReader()**.
3. User-defined subroutine procedure **MapFacultyTable()**.
4. **Back** button's **Click** event procedure.

The third subroutine **MapFacultyTable()** is used and called by the second subroutine **FillFacultyReader()** in our project. Now let's discuss the coding process for these subroutines one by one.

First, let's see the coding process for the subroutine **ShowFaculty()**. The function of this subroutine 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:\Chapter 8\SQLWebSelect**. You must store all faculty photo

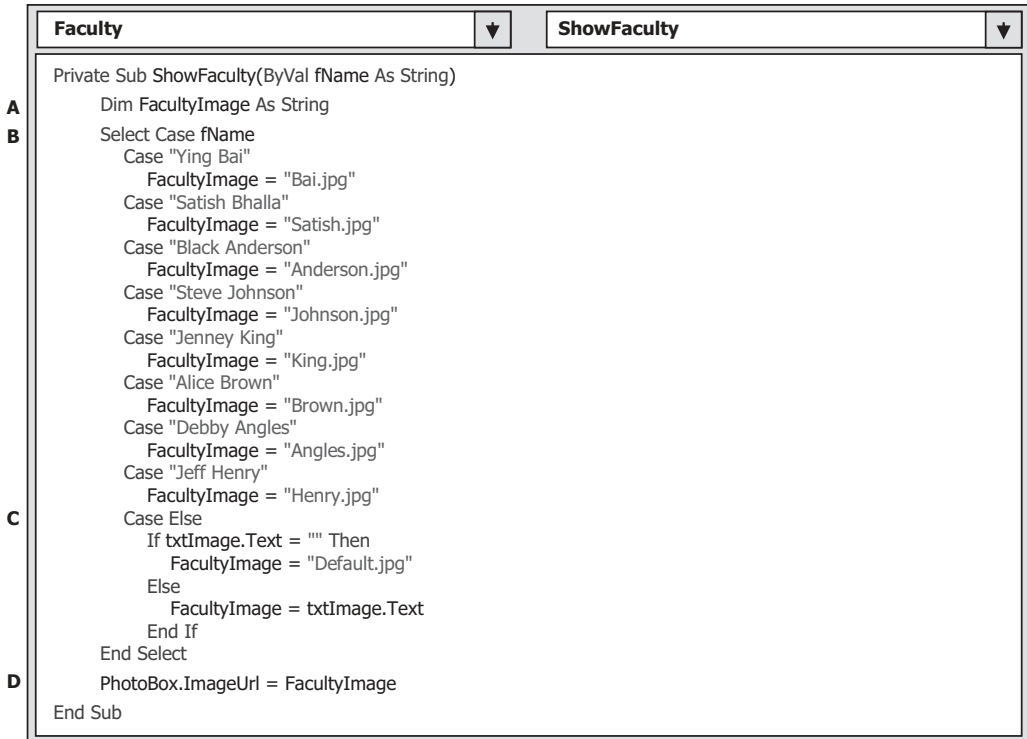


Figure 8.19. The codes for the subroutine ShowFaculty.

files in this location before you can run the 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, which includes the drive, path, and the name of the photo file.

To make it simple, in this project, we used the default folder to store our faculty photo files, which is `C:\Chapter 8\SQLWebSelect`.

Open the code page of the Faculty page, and type and create this subroutine as shown in Figure 8.19.

Let's have a closer look at the codes of this subroutine to see how they work.

- A.** A local string variable `FacultyImage` is created, and it is used to hold the name of the matched faculty photo file.
- B.** The `Select...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.** If not, this means that no matched faculty photo file is existed. If the Faculty Image textbox is empty, a default faculty image file is assigned to the local variable `FacultyImage`. Otherwise, the faculty image file stored in that box is used as a new faculty image.
- D.** The name of the matched faculty photo file is assigned to the `ImageUrl` property of the `Image` control to display that faculty photo.

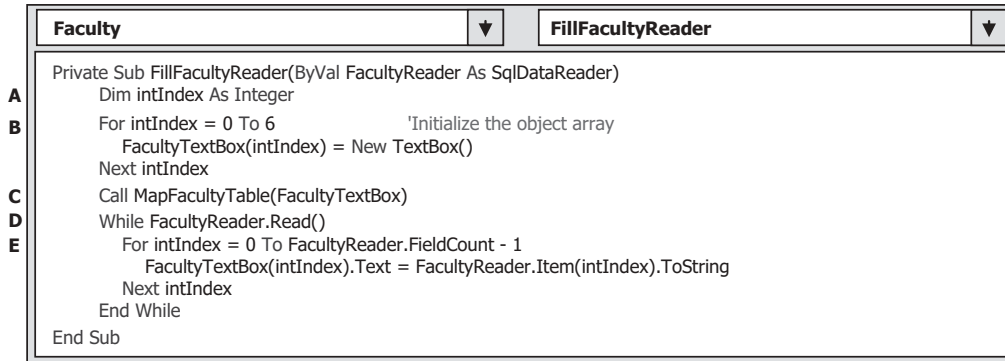


Figure 8.20. The codes for the subroutine FillFacultyReader.

One significant difference in displaying an image between the Windows-based and the Web-based application is that the `Image.Url` 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 the matched image file is needed to display the associated image in the 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 subroutine is `FillFacultyReader()`. Open the code page of the Faculty Web form, and type the codes that are shown in Figure 8.20 to create this subroutine inside the Faculty class.

The function of this subroutine 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 have a closer look at this piece of codes to see how it works.

- A.** A loop counter `intIndex` is declared first.
- B.** Seven instances of the textbox array are created and initialized. These seven objects are mapped to seven columns in the Faculty table in the database.
- C.** Another user-defined subroutine `MapFacultyTable()` is called to set up the correct mapping between the seven textbox controls on the Faculty page window and the seven columns in the query string `cmdString`.
- D.** A `While` loop is executed as long as the loop condition, `Read()` method, is `True`; this means that a valid data is read out from the `DataReader`. This method will return a `False` if no any 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...Next` loop is utilized to pick up each data read out from the `DataReader` object, and assign each of them to the associated textbox control on the Faculty page window. The `Item` property with the index is used here to identify each data from the `DataReader`.

Now let's develop the codes for the subroutine `MapFacultyTable()`. The function of this subroutine, 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. The reason for that is because the order of the textboxes distributed in the Faculty page may not be identical with the order of the data columns in the query

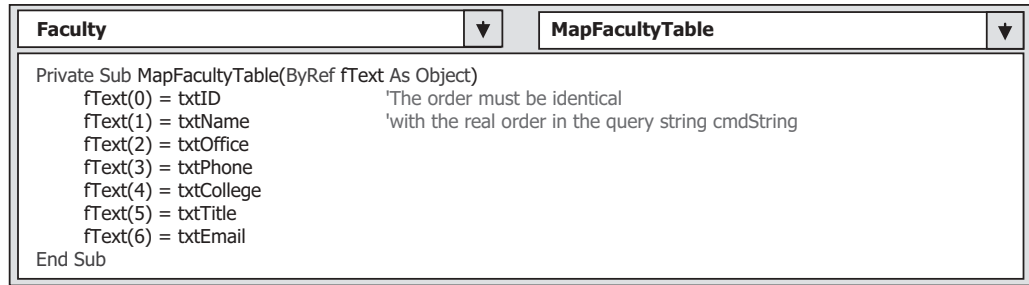


Figure 8.21. The codes for the subroutine MapFacultyTable.



Figure 8.22. The codes for the Back button's Click event procedure.

string cmdString we created at the beginning of the **Select** button's Click event procedure.

Open the code page of the Faculty Web form, and type the codes that are shown in Figure 8.21 to create this subroutine inside the Faculty class.

The order of seven textboxes on the right-hand side of the equal 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 has a correct one-to-one relation with the queried columns in the query string cmdString.

Finally, let's take care of the coding process for the **Back** button's Click event procedure. The function of this piece of codes 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 event procedure and enter the code line shown in Figure 8.22 into this procedure.

This piece of codes 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 by the user. The server resends the Selection page to the client when this button is clicked, and a request is sent to the server.

We have finished all coding development for the Faculty page. It is the time for us to run the project to test our pages. Before you can run the project, make sure that you have stored all faculty photo files in the default location C:\Chapter 8\SQLWebSelect. You can find all faculty and student image files in the folder Images that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy all image files from that folder and paste them into the default folder C:\Chapter 8\SQLWebSelect.

Now click on the Start Debugging button to run our project. Enter the suitable user-name 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

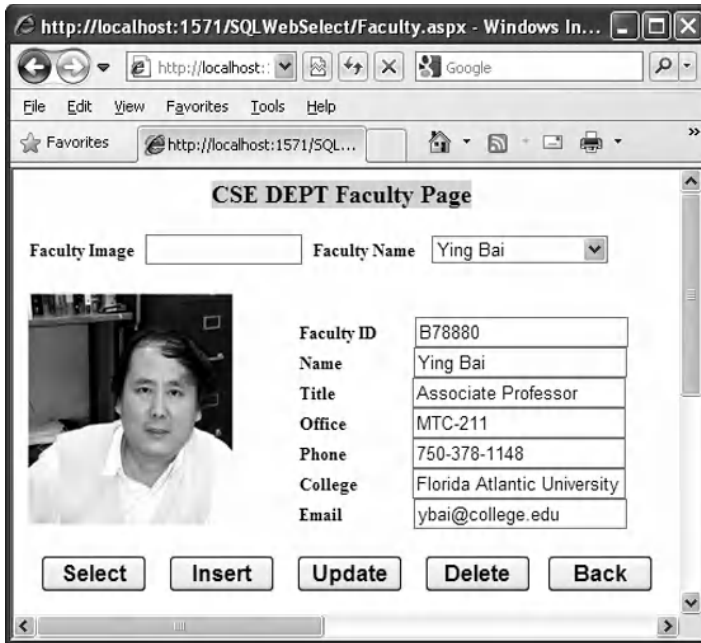


Figure 8.23. The running status of the Faculty page.

from the Faculty Name combo box, such as Ying Bai, and then click on the **Select** button to retrieve the selected faculty information from the database. All pieces of information related to that selected faculty are retrieved and displayed in 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.

8.3.9 Create the Fourth User Interface: Course Page

To create a new Web page and add it into 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 wizard. On the opened wizard, keep the default template **Web Form** selected. Then enter **Course.aspx** into the Name box as the name for our new page and click on the **Add** button to add it into our project.

On the opened Web form, add the controls that are 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. You must first use the Space or the Enter keys from the keyboard to locate the positions on the page for those controls. You cannot 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.

Table 8.5. Controls on the Course form

Type	ID	Text	TabIndex	BackColor	Font	AutoPostBack
Panel	Panel1		16	#C0C0FF		
Label	Label1	Faculty Name	0		Bold/Smaller	
DropDownList	ComboName		1			
ListBox	CourseList		17		Bold/Medium	True
Panel	Panel2		18	#C0C0FF		
Label	Label2	Course ID	2		Bold/Smaller	
TextBox	txtID		3			
Label	Label3	Course	4		Bold/Smaller	
TextBox	txtCourse		5			
Label	Label4	Schedule	6		Bold/Smaller	
TextBox	txtSchedule		7			
Label	Label5	Classroom	8		Bold/Smaller	
TextBox	txtClassroom		9			
Label	Label6	Credit	10		Bold/Smaller	
TextBox	txtCredit		11			
Label	Label7	Enrollment	12		Bold/Smaller	
TextBox	txtEnroll		13			
Button	cmdSelect	Select	14		Bold/Medium	
Button	cmdInsert	Insert	15		Bold/Medium	
Button	cmdUpdate	Update	16		Bold/Medium	
Button	cmdDelete	Delete	17		Bold/Medium	
Button	cmdBack	Back	18		Bold/Medium	

An easy way to build this Course page is to add an existing Course page **Course.aspx** that can be found in the folder **VB Forms\Web** in the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Perform the following operations to add this existing Course page into our project:

1. Right-click on our current project from the Solution Explorer window, and select the **Add Existing Item**.
2. Browse to the folder **VB Forms\Web** in the Wiley ftp site and select the **Course.aspx** item. Then click on the **Add** button to add this page into our project. You can also save this page into a temporary folder in your computer and then perform this adding action.

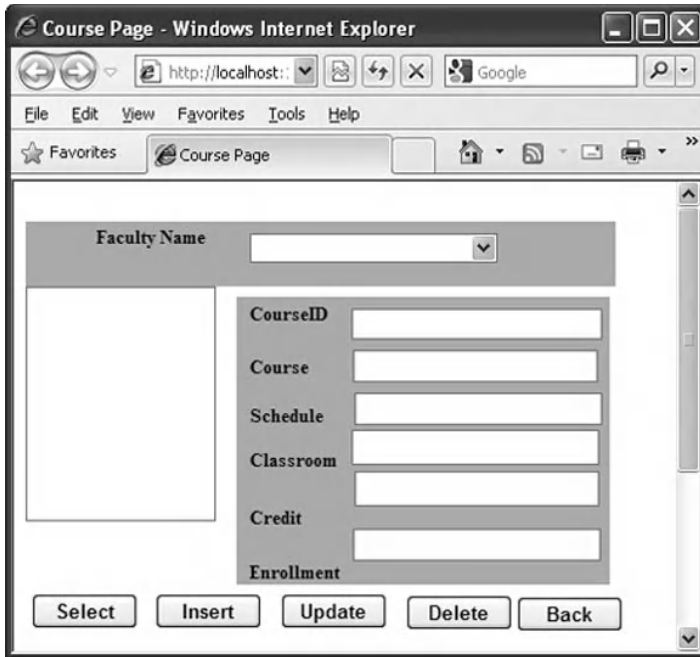


Figure 8.24. The finished Course Web page.

Your finished Course page should match the one that is shown in Figure 8.24.

Before we can continue to develop the following codes, we must emphasize one key point for the list box control used in the Web-based applications. There is a significantly different process for the list box control used in the Windows-based and Web-based applications.

8.3.9.1 The AutoPostBack Property of the List Box Control

One important property is the **AutoPostBack** property for the list box control **CourseList** in this page. Unlike the list box control used in the Windows-based application, a **SelectedIndexChanged** event will not be created in the server side if the user clicked and selected an item from the list box. The reason for that is because the default value for the **AutoPostBack** property of a list box control is set to **False** when you add a new list box to your Web form. This means that even if the user clicked and changed the item from the list box, a **SelectedIndexChanged** event can only be created in the client side, and it cannot be sent to the server. As you know, all controls, including the list box, are running at the server side when your project runs. So no matter how many times you clicked and changed the items from the list box, no event can be sent to the server side. Therefore, it looks like that your clicking on the list box cannot be responded by your project.

However, in this project, we need to use this **SelectedIndexChanged** event to trigger our event procedure to perform the course information query. In order to solve this

problem, the `AutoPostBack` property should be set to `True`. In this way, each time when you click on an item to select it from the list box, this `AutoPostBack` property will set a value to post back to the server to indicate that the user has triggered this control.

In this section, we only discuss the coding development for the `Select` and the `Back` buttons' Click event procedures to perform the course data query. The coding process for other buttons, such as `Insert`, `Update`, and `Delete`, will be discussed later in the following sections when we perform the data inserting, updating, or deleting actions against the database using the Web pages.

Now let's develop the codes for the `Select` and the `Back` buttons' Click event procedures to pick up the course data from the database using the Course Web page.

8.3.10 Develop the Codes to Select the Desired Course Information

The functions of the Course page are:

1. When the user selected the desired faculty member from the Faculty Name combo box control and clicks on the `Select` button, all IDs of the courses taught by the selected faculty should be retrieved from the database and displayed in the list box control `CourseList` on the Course page.
2. When the user clicks on any `course_id` from the list box control `CourseList`, the detailed course information related to the selected `course_id` in the list box will be retrieved from the database and displayed in six textboxes on the Course page.

Based on the function analysis above, we need to concentrate our coding process on two event procedures: the `Select` button's Click event procedure and the `CourseList` box's `SelectedIndexChanged` event procedure. The first piece of codes is used to retrieve and display all `course_id` related to courses taught by the selected faculty in the list box control `CourseList`, and the second coding is to retrieve and display the detailed course information, such as the course title, schedule, classroom, credit, and enrollment, related to the selected `course_id` from the `CourseList` control.

The above coding jobs can be divided into four parts:

1. Coding for the Course page loading and ending event procedures. These procedures include the `Page_Load()` and the `Back` button's Click event procedure.
2. Coding for the `Select` button's click event procedure.
3. Coding for the `SelectedIndexChanged` event procedure of the list box control `CourseList`.
4. Coding for other user-defined subroutine procedures.

Before we can take care of the first coding job, we need to add two `Imports` commands to the top of the Course page. Open the code window of the Course page and enter two `Imports` commands to the top of that page:

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

Now let's start our coding process from the first part.

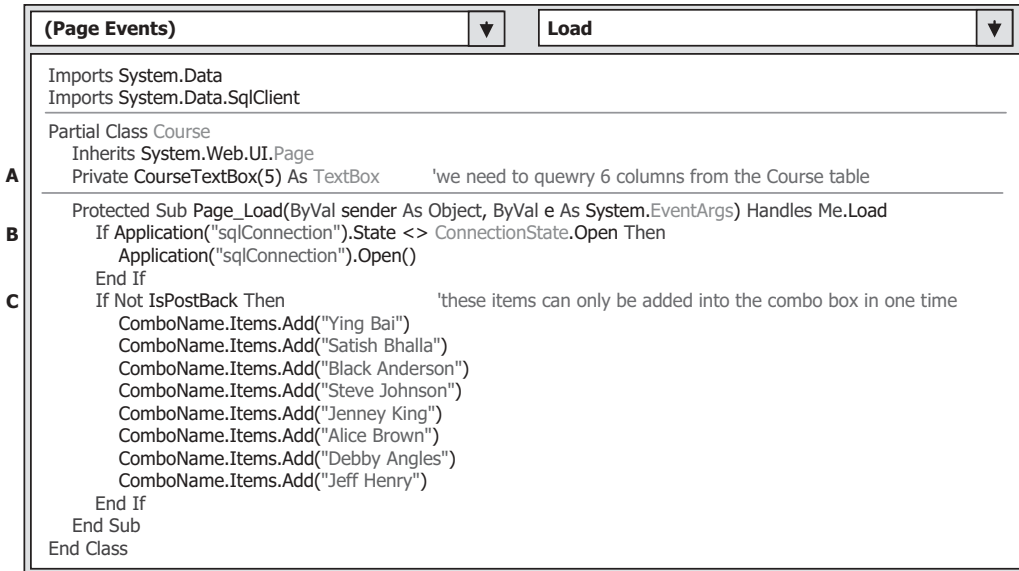


Figure 8.25. The codes for the Page_Load event procedure.

8.3.10.1 Coding for the Course Page Loading and Ending Event Procedures

Open the Page_Load event procedure by selecting (Page Events) from the Class Name combo box and Load from the Method Name combo box from the code window. Enter the codes that are shown in Figure 8.25 into this event procedure.

Let's take a closer look at this piece of codes to see how it works.

- A.** This coding fragment is similar to the one we did for the Faculty form. Six textbox controls are used to display the detailed course information that is related to the selected faculty from the Faculty Name combo box. The Course table has seven columns, but we only need six of them, so the size of this TextBox array is 5, and each element or each TextBox control in this array is indexed from 0 to 5.
- B.** The function of this code segment is: Before we can perform any data query, we need to check whether a valid connection is available. Since we created a global 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.
- C.** The following codes are used to initialize the Faculty Name combo box control, and the Add() method is utilized to add all faculty members into this combo box 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 codes. As we mentioned in Section 8.3.9.1, an AutoPostBack property will be set to True whenever the user clicked and selected an item from the list box control, and this property will be sent to the server to indicate that an action has been taken by the user to this list box. After the server received this property, it will send back a refreshed Course page to the client; therefore, the Page_Load event procedure 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 procedure is to attach another copy of all faculty members to the end of those faculty members that have been already added into the

Faculty Name combo box control when the Course page is displayed in the first time. As the number of times you clicked on an item from the **CourseList** box increases, the number of copies of all faculty members will also be increased and displayed in the Faculty Name combo box. To avoid these duplications, we only need to add all faculty members in the first time as the Course page is displayed, but do nothing if an **AutoPostBack** property occurred.

The codes for the **Back** button's Click event procedure are similar to that for the **Back** button's Click procedure in the Faculty page. When this button is clicked 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 function. Double-click on the **Back** button from the Course page window and enter the following codes into this procedure:

```
Response.Redirect("Selection.aspx")
```

Let's continue to develop the codes for the **Select** button's click event procedure.

8.3.10.2 Coding for the Select Button's Click Event Procedure

As we mentioned at the beginning of this section, the function of this event procedure is: when the user selected the desired faculty member from the Faculty Name combo box 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 list box control **CourseList** in the Course page.

Double-click on the **Select** button from the Course page window to open this event procedure, and enter the codes that are shown in Figure 8.26 into this procedure.

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

- A. The joined table query string is declared at the beginning of this event procedure. Here, two columns are queried. The first one is the **course_id**, and the second is the course name.

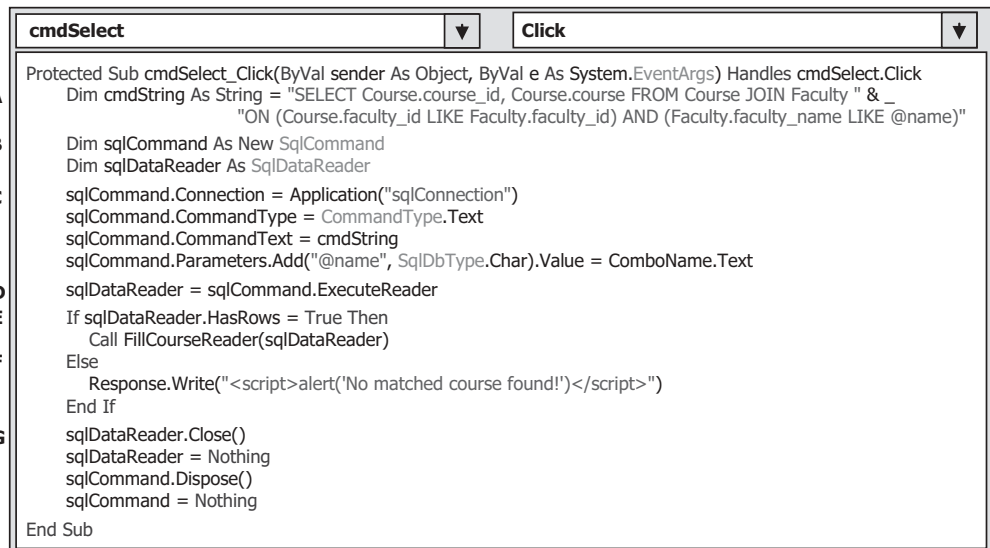


Figure 8.26. The codes for the **Select** button's Click event procedure.

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 clicked and selected 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 equal symbol for the criteria in the `ON` clause in the query string, and this is required by SQL Server database operation. For a more detailed discussion about the joined table query, refer to Section 5.19.6 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 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 parameter `@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 combo box, which should be selected by the user as the project runs.
- 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 subroutine `FillCourseReader()` 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 jobs are preformed to release objects used for this query.

Now let's develop the codes for the user-defined subroutine `FillCourseReader()`, which is shown in Figure 8.27. Open the code page of the Course Web form and enter the codes that are shown in Figure 8.27 to create this procedure inside the Course class.

Let's see how this piece of codes works.

- A.** 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 data from the Course table.
- B.** We need to clean up the CourseList box before it can be filled.
- C.** 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

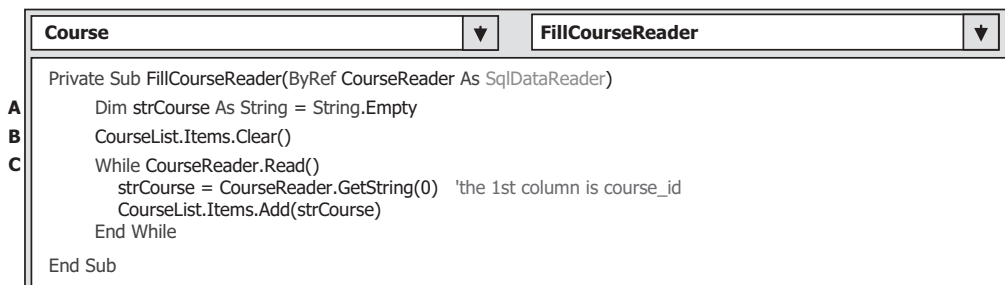


Figure 8.27. The codes for the subroutine `FillCourseReader()`.

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 procedure of the list box control `CourseList`. The function of this event procedure is: when the user clicks any `course_id` from the list box control `CourseList`, the detailed course information related to the selected `course_id`, such as the course title, schedule, credit, classroom, and enrollment, will be retrieved from the database and displayed in six textboxes on the Course page form.

8.3.10.3 Coding for the `SelectedIndexChanged` Event Procedure of the `CourseList` Box

Double-click on the list box control `CourseList` from the Course Web form to open this event procedure, and enter the codes that are shown in Figure 8.28 into this procedure.

Let's take a closer look at this piece of codes to see how it works.

- A.** The query string is created with six queried columns, such as `course_id`, `course`, `credit`, `classroom`, `schedule`, and `enrollment`. The query criterion is `course_id`, which is obtained from the `CourseList` box control. The comparator `LIKE` is used to replace the original equal symbol for the criteria in the `WHERE` clause in the query string, and this is required by SQL Server database operation.
- 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.

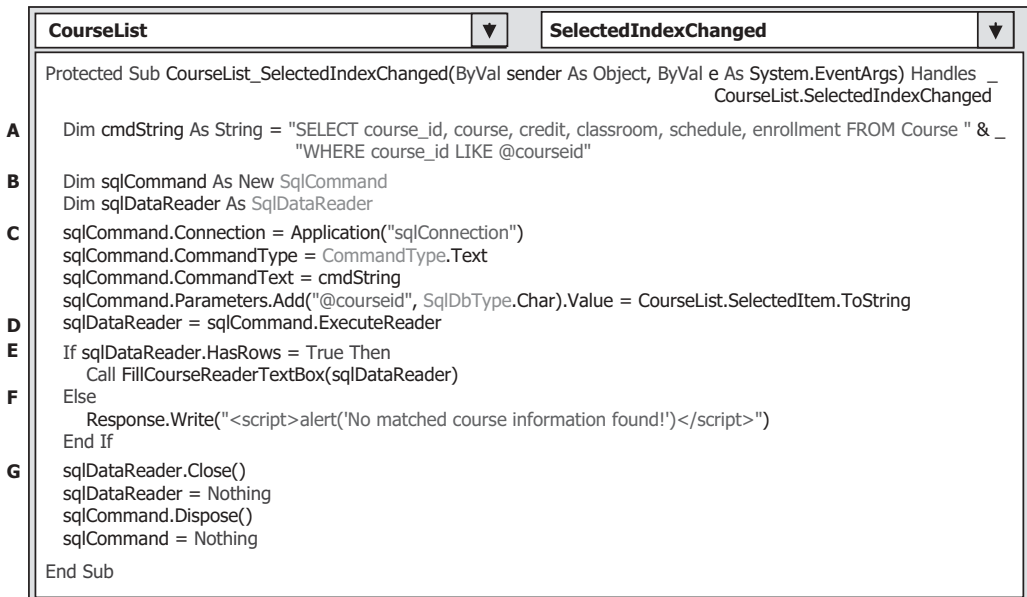


Figure 8.28. The codes for the `SelectedIndexChanged` event procedure.

- 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.
- 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 subroutine procedure `FillCourseReaderTextBox()` is called to fill detailed course information into six 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 subroutine procedures includes the coding for the user-defined subroutine procedures `FillCourseReaderTextBox()` and `MapCourseTable()`.

8.3.10.4 Coding for Other User-Defined Subroutine Procedures

First, let's develop the codes for the subroutine `FillCourseReaderTextBox()`. On the opened code page of the Course Web form, enter the codes that are shown in Figure 8.29 into the Course class to create this user-defined subroutine procedure.

Let's take a closer look at this piece of codes 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 loop is used to create the textbox object array and perform the initialization for those objects.
- C. The user-defined subroutine `MapCourseTable()` is executed to set up a one-to-one relationship between each textbox control in the Course page and each queried column in the query string. This step is necessary since the distribution order of six textbox controls in the Course page may be different with the column order in the query string.

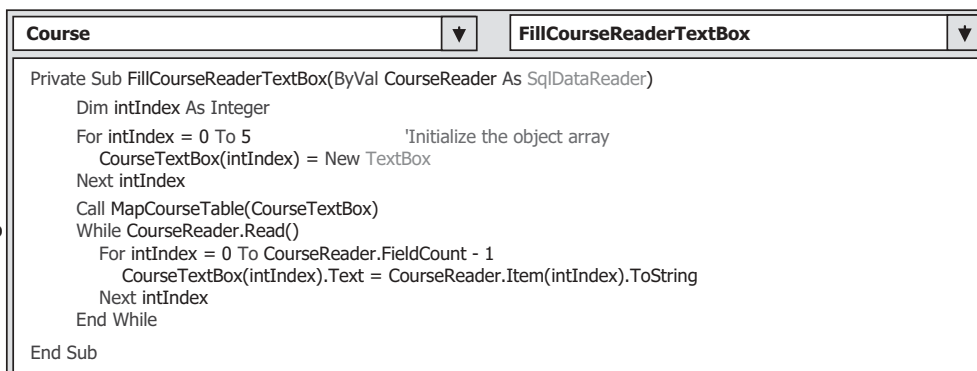


Figure 8.29. The codes for the subroutine `FillCourseReaderTextBox()`.

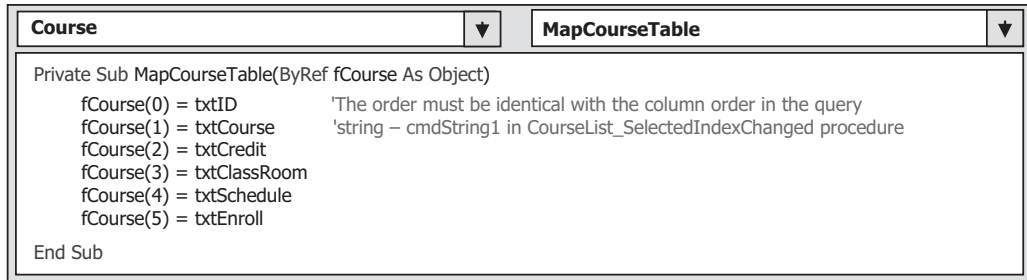


Figure 8.30. The codes for the subroutine MapCourseTable().

- D.** A While and a For...Next loop are used to pick up all six 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, 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...Next 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 codes for the subroutine MapCourseTable() are shown in Figure 8.30.

The function of this piece of codes is straightforward with no trick. The order of the textboxes on the right-hand side of the equal operator is the column order in the query string—cmdString. By assigning each column of required 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 in the Course page is built, and the data retrieved from the DataReader can be mapped exactly to the associated textbox control in the Course page, and can be displayed in there.

At this point, we have finished all coding developments for the Course Web form. Now let's run the project to test the function 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 Course page, select a faculty member from the Faculty Name combo box control and click on the Select button to retrieve all courses (course_id) taught by that selected faculty. Immediately, all courses (course_id) are retrieved and displayed in the CourseList box. Your running result should match the one that is shown in Figure 8.31.

Click on any course_id from the CourseList box to select it; immediately, the detailed course information related to that selected course_id is displayed in six 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.

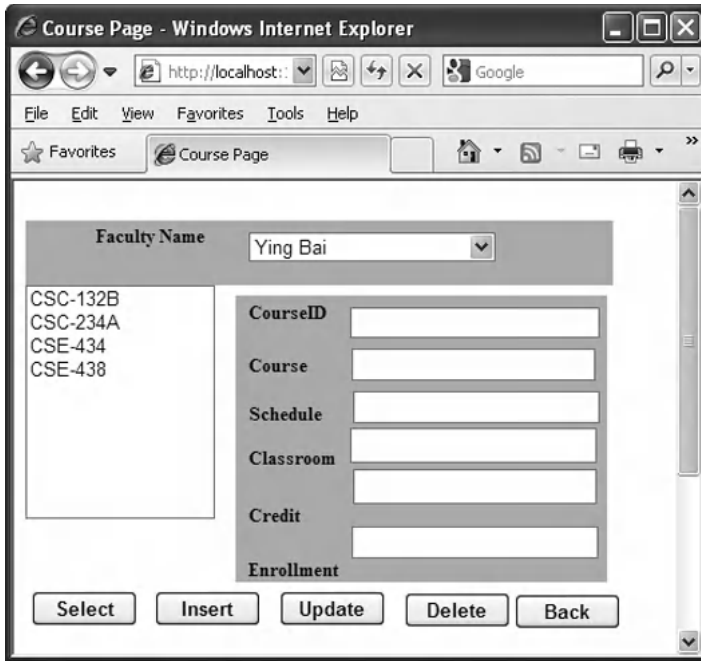


Figure 8.31. The running status of the Course page.

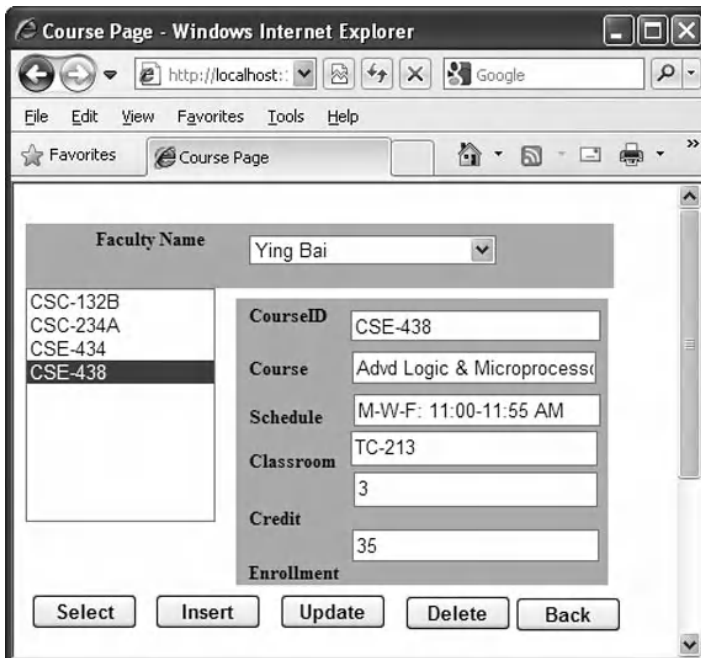


Figure 8.32. The detailed course information.

A complete Web application project **SQLWebSelect**, which is used for data query from the SQL Server database, can be found in the folder **DBProjects\Chapter 8** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Next, let's discuss how to insert data into our sample database via Web applications.

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 do not need to create any new Web application; instead, we can modify an existing Web project **SQLWebSelect** we built in the last section to make it our new Web application **SQLWebInsert**. Perform the following operations to create this new Web application project **SQLWebInsert**:

1. Open the Windows Explorer and create a new folder **Chapter 8** if you have not done that.
2. Copy the project **SQLWebSelect** from the folder **DBProjects\Chapter 8** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1), and paste it to our new folder **C:\Chapter 8**.
3. Rename this project to **SQLWebInsert**.

Recall that we built five buttons on the Faculty page in the project **SQLWebSelect**. In this section, we will concentrate on the coding development for the **Insert** button on the Faculty page to perform the faculty data insertion action to our sample database.

8.4.1 Develop the Codes to Perform the Data Insertion Function

The function of this **Insert** button's Click event procedure is:

1. During the project runs, you need to open the Faculty page by selecting the **Faculty Information** from the Selection page.
2. To insert a new faculty record into the database, you need to enter seven pieces of new information into seven textboxes in the Faculty page. The information includes the **faculty_id**, **faculty_name**, **title**, **office**, **phone**, **college**, and **email**.
3. The Faculty Image 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 faculty image will be adopted and displayed when this new record is validated.
4. After all pieces of information has been filled into all textboxes, you can click on the **Insert** button to insert this new record into the Faculty table in our database via the Web page.

Now let's start creating the codes for this **Insert** button's Click event procedure.

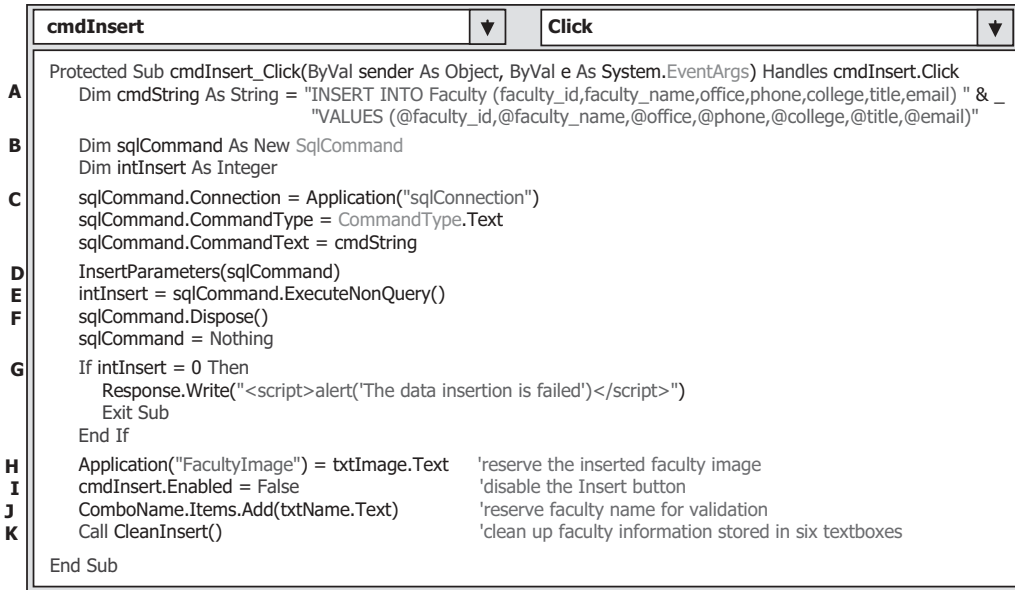


Figure 8.33. The codes for the Insert button's Click event procedure.

8.4.2 Develop the Codes for the Insert Button Click Event Procedure

Open the Insert button's Click event procedure by double-clicking on the Insert button from the Faculty Web form, and enter the codes that are shown in Figure 8.33 into this event procedure.

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

- A.** The insert query string is declared first, and it contains seven pieces of information that is related to seven columns in the Faculty table in the database.
- B.** The data components and local variables used in the procedure are declared here. The local integer variable `intInsert` is used to hold the returned running result from the execution of the data insertion command.
- C.** The Command object is initialized by assigning it with the connection object stored in the Application state, the command type and the command text objects, respectively.
- D.** The user-defined subroutine `InsertParameters()` is executed to assign all seven input parameters to the Parameters collection of the command object.
- E.** The `ExecuteNonQuery()` method of the command object is called to run the insert query to perform this data insertion.
- F.** A cleaning job is performed to release all objects used in the procedure.
- G.** The `ExecuteNonQuery()` method will return an integer 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 returned, which means that no any row has been inserted into the database, a warning message is

displayed to indicate this situation, and the procedure is exited. Otherwise, the data insertion is successful.

- H.** A global variable `FacultyImage` is created and initialized with the faculty image file name stored in the Faculty Image textbox. In some cases, the user may want to add a faculty image with that faculty record insertion. In order to save this image file for the data validation, we need this step.
- I.** 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.
- J.** The newly inserted faculty name is added into the Faculty Name combo box by using the `Add()` method, and this faculty name will be used later for the validation purpose.
- K.** The user-defined subroutine procedure `CleanInsert()` is executed to clean up six textboxes in the Faculty page (except the Faculty ID textbox).

When the content of the `faculty_id` textbox is changed (a `TextChanged` event of the `faculty_id` textbox will be triggered), which means that a new faculty record should be inserted, we need to enable the `Insert` button if this situation happened. To do this piece of codes, double-click on the `faculty_id` textbox from the Faculty page to open its `TextChanged` event procedure and enter `cmdInsert.Enabled = True` into this procedure.

The detailed codes for the user-defined subroutine `InsertParameters()` are shown in Figure 8.34.

This piece of codes is straightforward and easy to be understood. 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.

The codes for the user-defined subroutine `CleanInsert()` are shown in Figure 8.35.

The function of this piece of codes is to clean up contents of six textboxes, except the `faculty_id` textbox. The reason for that is: the `Insert` button would be enabled if the content of the `faculty_id` textbox is cleaned up (changed) since a `TextChanged` event will be triggered. However, this cleaning up action has nothing to do with inserting a new record. Therefore, in order to avoid this confusing operation, we will not clean up the `faculty_id` textbox.

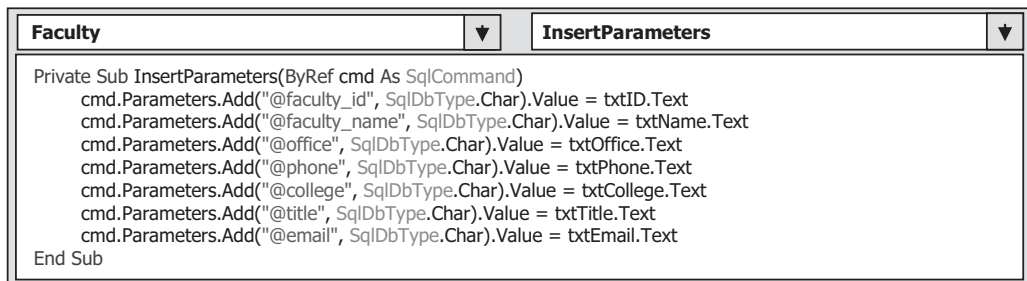


Figure 8.34. The codes for the subroutine `InsertParameters()`.

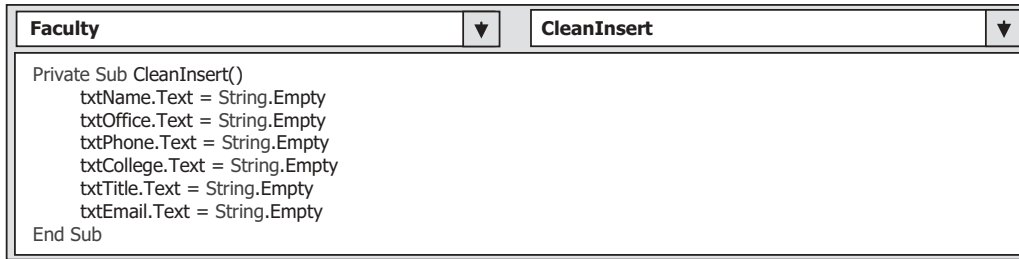


Figure 8.35. The codes for the subroutine CleanInsert().

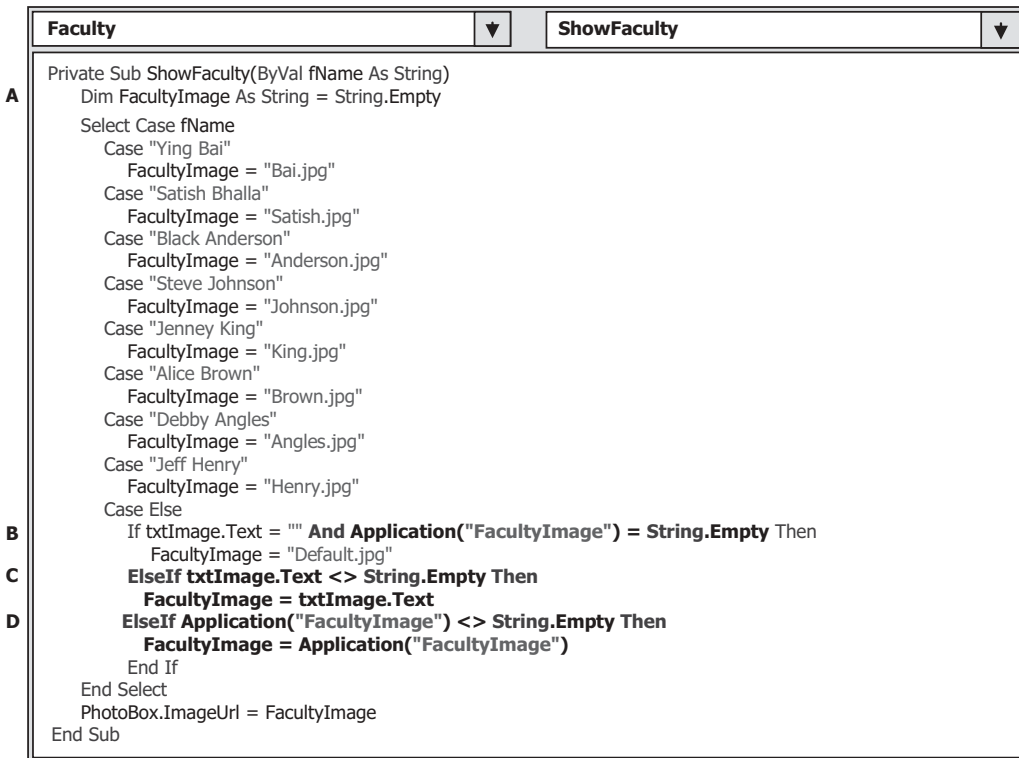


Figure 8.36. The modified codes for the subroutine ShowFaculty().

8.4.3 Modify the Codes in the Subroutine ShowFaculty() for the Data Validation

In order to validate this data insertion action, we need to modify some codes inside the user-defined subroutine ShowFaculty() to enable a newly inserted faculty image to be retrieved and displayed in this page if the user wants to add a new faculty image for that data insertion.

Open this subroutine and perform the modifications, which are shown in Figure 8.36, to this procedure. The modified parts have been highlighted in bold.

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

- A.** The local variable **FacultyImage** is initialized with an empty string.
- B.** To check whether a new faculty image has been inserted or no matched faculty image has been found, we use an **And** logic operator to combine both conditions together. If both of them are empty, which means that no matched faculty image can be found, a default faculty image is used.
- C.** If the Faculty Image textbox contains a valid faculty image file's name, it is assigned to the local String variable **FacultyImage** and displayed later.
- D.** If the global variable **FacultyImage** is not empty, which means that a valid faculty image's name has been assigned to it by the user, and this faculty image will be used and displayed later.

Now we have finished all coding development for this data insertion action, and we can run the project to test the data insertion function via the Web page. However, before we can start the project, make sure that all faculty image files, including a default faculty photo file named **Default.jpg**, have been stored in our default folder in which our project is located since we need to use those photo files to run our project. Also, make sure that the start page is the LogIn page by right-clicking on the LogIn page from the Solution Explorer window and selecting the item **Set As Start Page**.

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. Enter the following data as the information for a new faculty record:

- B55880 Faculty ID textbox
- Susan Bai Name textbox
- Professor Title textbox
- MTC-335 Office textbox
- 750-378-2355 Phone textbox
- Duke University College textbox
- sbai@college.edu Email textbox
- Default.jpg Faculty Image textbox

Your finished new faculty information page is 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 six textboxes are cleaned up.

8.4.4 Validate the Data Insertion

To confirm and validate this faculty record insertion, go to the Faculty Name combo box control, and you can find that the newly inserted faculty name **Susan Bai** is already there. Click on this name to select this faculty and then click on the **Select** button to retrieve this newly inserted record from the database and display it in this page. The inserted record is displayed in this page, which is shown in Figure 8.38.

Our data insertion is successful.

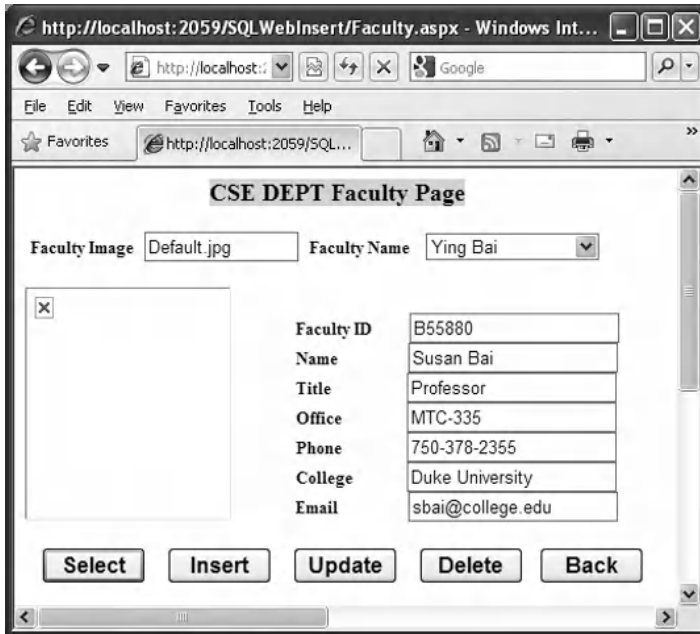


Figure 8.37. The running status of the Faculty page.

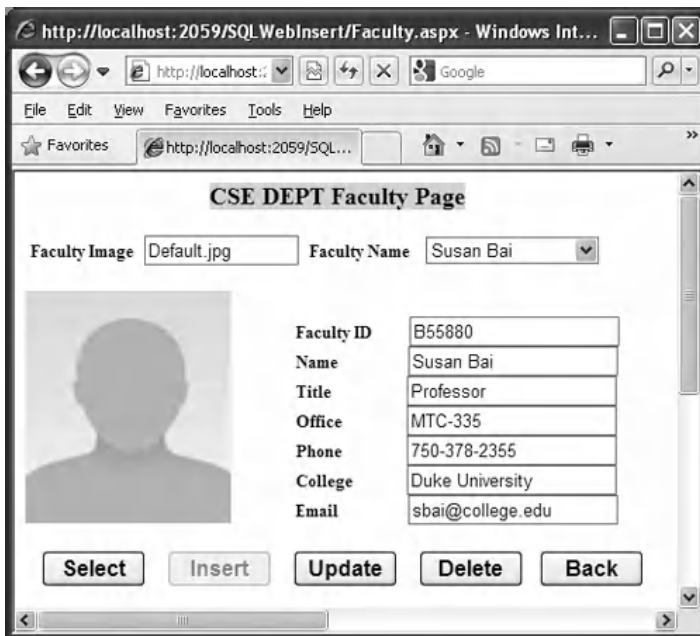


Figure 8.38. The data validation process.

Click on the **Back** button, and then the **Exit** button to close our project.

A complete Web application project **SQLWebInsert** can be found in the folder **DBProjects\Chapter 8** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

In the next section, we will discuss how to perform the data updating and deleting actions against the SQL Server database via the Web pages.

8.5 DEVELOP WEB APPLICATIONS TO UPDATE AND DELETE DATA IN SQL SERVER DATABASES

Updating or deleting data against the relational databases is a challenging topic. We have provided a very detailed discussion and analysis for this topic in Section 7.1.1. Refer to that section to get more detailed discussion for these data actions. Here, we want to emphasize some important points related to the data updating and deleting.

1. When updating or deleting data against 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 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 an existing data against 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 tables operations listed above. In practice, it is very rare to update a primary key for an existing record against the database in the real applications. So in this section, we concentrate our discussion on updating the existing record 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, which 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 delete the data with the primary key from the database.

Keep these three points we discussed above in mind; now let's begin our project.

We need to modify our existing project **SQLWebInsert** and make it as our new project **SQLWebUpdateDelete**. To do that, open the 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** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Rename this project to **SQLWebUpdateDelete**.

To update or delete an existing record against 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 those data actions. To meet our data actions' requirements, we need to perform some modifications to the codes in the Faculty page.

First, let's handle the data updating action to the Faculty table in our sample database via the Faculty page.

8.5.1 Modify the Codes for the Faculty Page

The code modifications for this data action can be divided into two parts: the code modifications to the **Select** button's Click event procedure and code creation for the **Update** button's Click event procedure. First, let's handle the code modifications to the **Select** button's Click event procedure.

The only modification to this event procedure is to add one more statement, which is shown in step **A** in Figure 8.39.

The newly added statement has been highlighted in bold. The purpose of this statement is to store the current selected faculty name that is located at the Faculty Name combo box control into the Application state as a global variable. During the data updating process, the faculty name may be updated by the user. If this happened, the updated faculty name that is stored in the **txtName** textbox will be added into the Faculty Name combo box, and the original faculty name will 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 when the server posts back a refreshed Faculty page

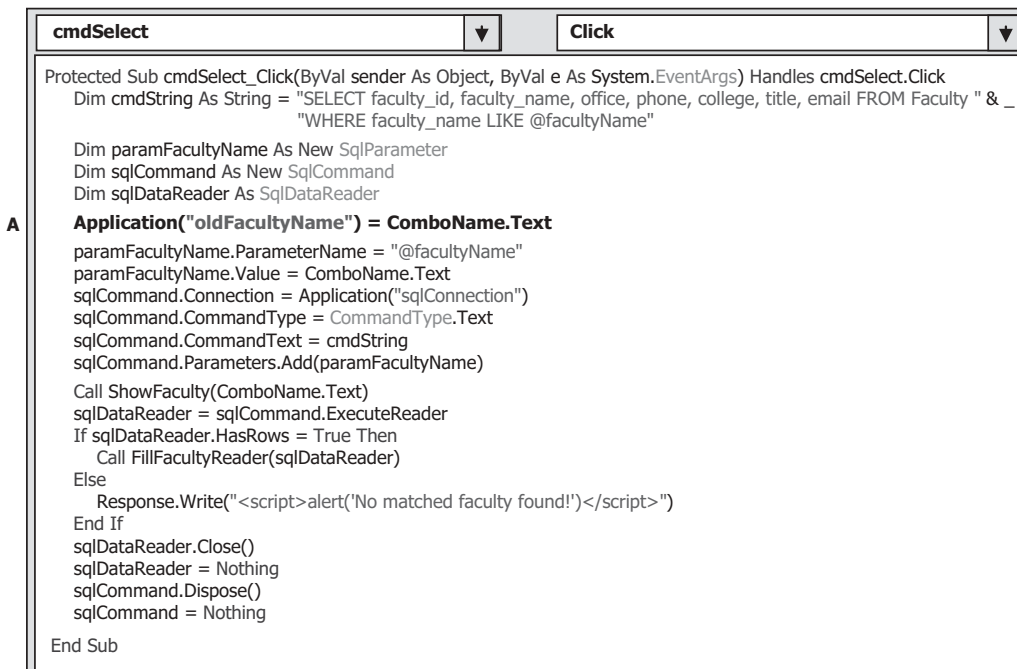


Figure 8.39. The modified Select button's event procedure.

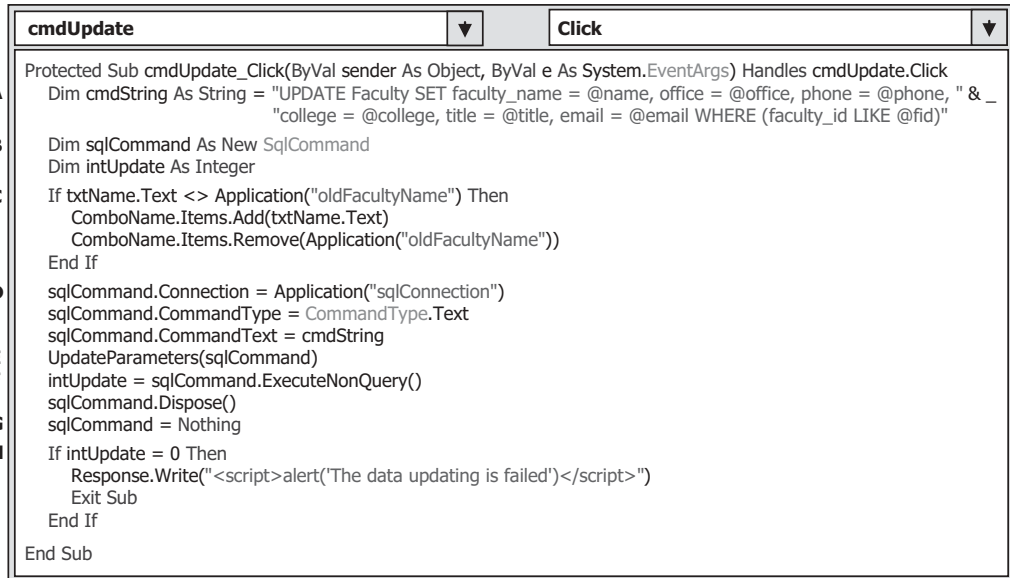


Figure 8.40. The codes for the Update button's click event procedure.

based on the client's request, all contents in all controls on that page will be refreshed and all old staff will be lost.

Now let's develop the codes for the Update button's click event procedure.

8.5.2 Develop the Codes for the Update Button Click Event Procedure

Open this event procedure by double-clicking on the Update button from the Faculty Web form window and enter the codes that are shown in Figure 8.40 into this procedure.

Let's take a closer look at this piece of codes to see how it works.

- A.** An updating query string is declared first with the `fid` as the name of the dynamic parameter. This is because we want to update all other columns in the Faculty table based on the `faculty_id` that will be kept unchanged.
- B.** All data objects used in this procedure are created here, and a local integer variable `intUpdate` is also created, which is used as a value holder to keep the returned data from executing the `ExecutNonQuery()` method.
- C.** Now we need to check whether the user wants to update the faculty name or not. To do that, we need to compare the global variable `oldFacultyName` that is stored in the Application state during the data selection process in the Select button's click event procedure with the current or updated faculty name that is stored in the `txtName` textbox. If both names are different, this means that the user has updated the faculty name. In that case, we need to add the updated faculty name into the Faculty Name combo box and remove the old faculty name from that box to allow users to select this updated faculty to perform the data validation later.

- D. The Command object is initialized with the connection object, command type, and command text.
- E. The user-defined subroutine `UpdateParameters()`, whose detailed codes are shown below, is called to assign all input parameters to the command object.
- F. 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 data is stored to the local integer variable `intUpdate`.
- G. A cleaning job is performed to release all data objects used in this procedure.
- H. 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 any row has been updated and this data updating has failed, a warning message is displayed and the procedure is exited. Otherwise, if this value is nonzero, which means that this data updating is successful.

The detailed codes for the subroutine `UpdateParameters()` are shown in Figure 8.41.

Seven input parameters are assigned to the `Parameters` collection property of the command object using the `Add()` method. One point for this parameters assignment is the last input parameter `@fid`. Since we want to update all other columns in the `Faculty` table except the `faculty_id`, therefore, we will use the original `faculty_id` without any modification.

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, make sure that the starting page is the `LogIn` page, and all faculty image files, including a default faculty image file `Default.jpg`, have been stored in our default folder. To check the starting page, perform the following operations:

1. 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 Start action wizard.
2. Then check on the **Specific page** radio button and click on the expansion button to open the Select Page to Start wizard.
3. Select the `LogIn.aspx` page as the start page by clicking on it.
4. Click on two OK buttons to close this starting page setup.

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,

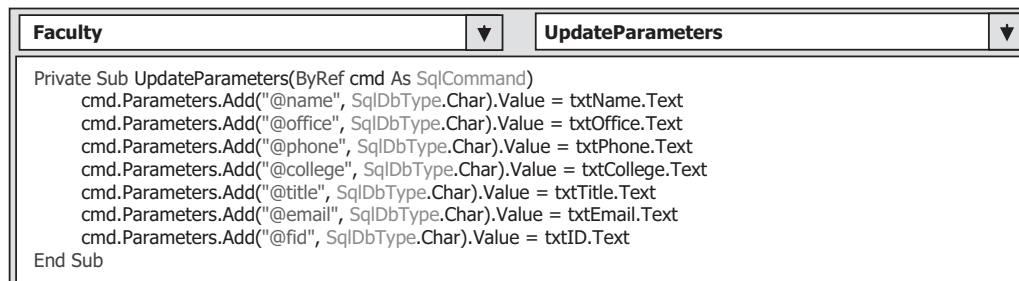


Figure 8.41. The codes for the subroutine `UpdateParameters()`.

and select the **Faculty Information** item from the Selection page to open the Faculty page. Keep the default faculty name **Ying Bai** selected from the Faculty Name combo box, and click on the **Select** button to retrieve the information for this selected faculty from the database and display it in this page.

Now let's test the data updating actions by entering the following data into the associated textboxes to update this faculty record:

- Susan Bai Name textbox
- Professor Title textbox
- MTC-353 Office textbox
- 750-378-3300 Phone textbox
- Duke University College textbox
- sbai@college.edu Email textbox
- Default.jpg Faculty Image textbox

Click on the **Update** button to perform this data updating. To confirm this data updating, first select another faculty from the Faculty Name combo box and click on the **Select** button to retrieve and display that faculty information. Then select the updated faculty **Susan Bai** from the combo box 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 8.42.

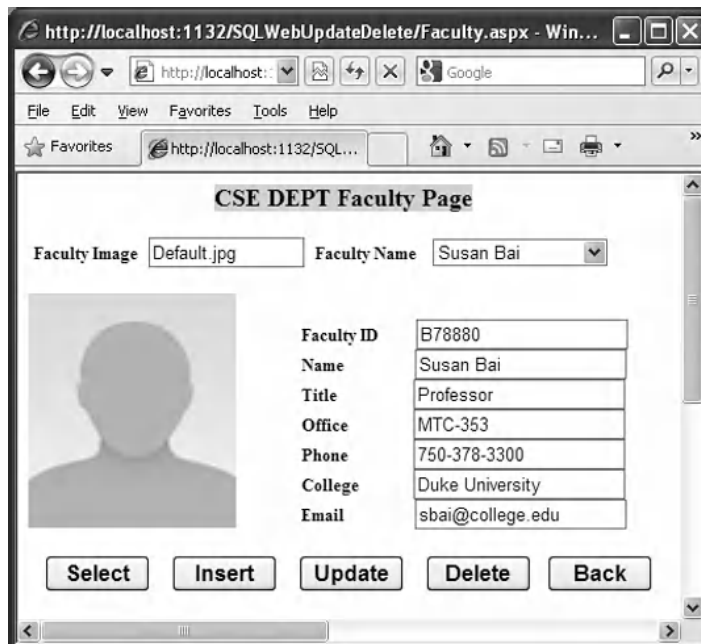


Figure 8.42. The data updating process.

Table 8.6. The original data for the faculty member Ying Bai 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

Our data updating action is very successful. Click on the **Back** and then the **Exit** button to terminate our project.

It is highly recommended to recover this updated faculty record to the original one. Refer to the original faculty record shown in Table 8.6 to complete this recovery job. You can do this recovery using either the Server Explorer window or the SQL Server Management Studio. Next, let's take care of the data deleting action against the SQL Server database.

8.5.3 Develop the Codes for the Delete Button Click Event Procedure

Since deleting a record from a relational database is a complex issue, we have provided a detailed discussion about this data action in Section 8.5. Refer to that part to get more detailed information for this data action. In this section, we divide this data deleting action discussion into the following five sections:

1. Relationships between five tables in our sample database
2. Data deleting sequence
3. Use the Cascade deleting option to simplify the data deleting
4. Create a stored procedure to perform the data deleting
5. Call the stored procedure to perform the data deleting action

8.5.3.1 Relationships between Five Tables in Our Sample Database

As we discussed in Section 8.5, 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. 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, which is Figure 8.43 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 relationships exist between the five tables.

For example, if one wants to delete a faculty record from the Faculty table, one must perform the following deleting jobs:

- 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

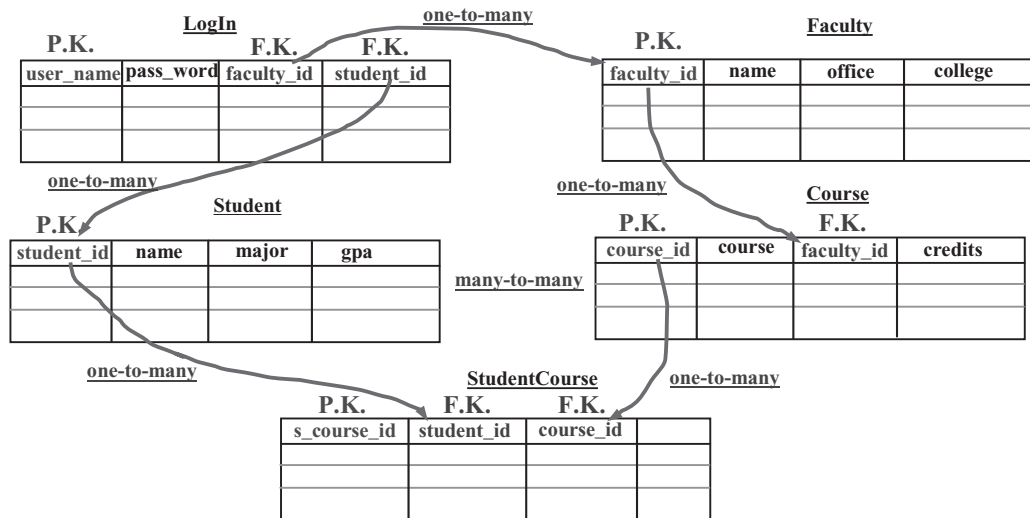


Figure 8.43. The relationships between five tables.

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 now is a parent table relative to the StudentCourse table since the `course_id` is a primary key in the Course table, but a foreign key in the StudentCourse table. 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, so 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 have been deleted, finally, the faculty member can be deleted from the parent table, Faculty table.

8.5.3.2 Data Deleting Sequence

To summarize, to delete a record from the Faculty table, one needs to perform the following deleting jobs in the **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 it is in these operations to delete one record from the relational database from this example.

8.5.3.3 Use the Cascade Deleting Option to Simplify the Data Deleting

To simplify the data deleting operations, we can use the cascade deleting option provided by the SQL Server 2008 Database Management Studio.

Recall that when we created and built the relationship between our five tables, the following five **relationships** are built between tables:

1. A relationship between the LogIn and the Faculty tables is 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 is 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 is 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 is 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 is 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 Section 8.5.3.2; to delete a record from the Faculty table, one needs to perform four deleting operations in that sequence. Compared with all four deleting operations, the first one is the most difficult and the reason for that is:

To perform the first data deleting, one must first find all **course_id** that use the **faculty_id** as the foreign key from the Course table, and then based on those **course_id**, one needs to delete all records that use those **course_id** as the foreign keys from the StudentCourse table. For deleting operations in sequences 3 and 4, they are easy, and each deleting operation only needs one deleting query. The question 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 for the data deleting and updating setup dialog provided by the SQL Server 2008 Database Management Studio. This Cascade option allows the SQL Server 2008 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 2008|SQL Server Management Studio**. On the opened Studio Express wizard, click on **Database** and expand our sample database **CSE_DEPT**. Then expand that database to display all five tables. Since we only have our interest on relationships 4 and 5, expand the **dbo.StudentCourse** table and expand the **Keys** folder to display all **Keys** we set up before. Double-click on the **FK_StudentCourse_Course** key to open it, which is shown in Figure 8.44.

On the opened wizard, keep our desired foreign key **FK_StudentCourse_Course** selected in 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 8.44.

For the foreign key **FK_StudentCourse_Student**, the same cascade mode has been set up for **Student** and the **StudentCourse** tables.

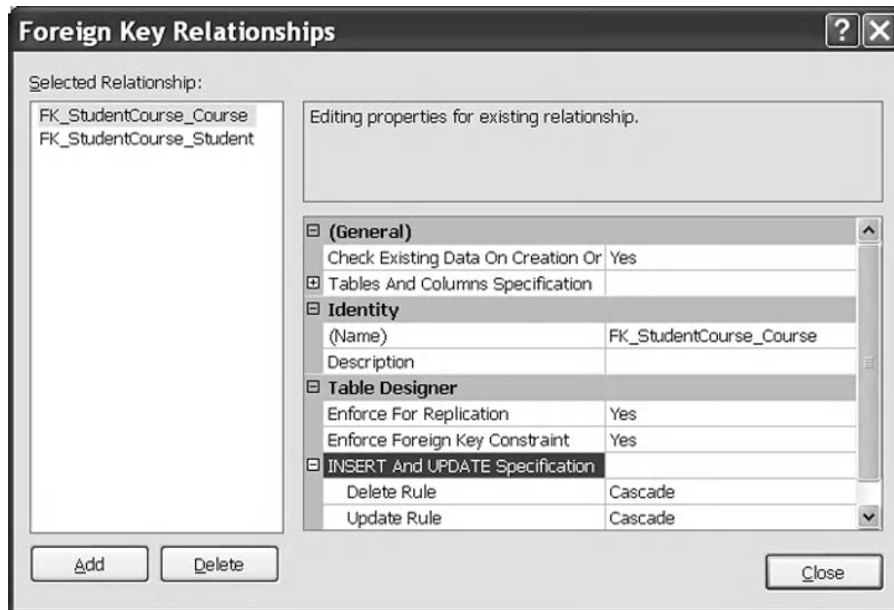


Figure 8.44. The Foreign Key Relationship wizard.

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. So now you can see how easy it is to perform the data deleting in sequence 1.

After the first data deleting operation listed in the deleting sequence in Section 8.5.3.2 is performed, we can do the following three operations by executing three deleting queries. But we want to integrate those three queries into a single stored procedure to perform this data deleting operation.

Well, wait a moment before we can start to create our stored procedure. One question is that is it possible for us to set up Cascade options for relationships 1, 2, and 3 to allow the SQL Server engine to help us to perform those data deleting operations? If it is, can we only use one query to directly delete the faculty member from the `Faculty` table? The answer is Yes! We prefer to leave this as homework to allow students to handle this issue themselves.

Now let's create our stored procedure for this data deleting operation.

8.5.3.4 Create the Stored Procedure to Perform the Data Deleting

This stored procedure contains three deleting queries that can be mapped to three sequences listed in Section 8.5.3.2, which are sequences 2, 3 and 4.

Open the Visual Studio.NET 2010 and open the Server Explorer window, expand our database `CSE_DEPT.mdf`, and right-click on the **Stored Procedures** folder. Select the **Add New Stored Procedure** item from the pop-up menu and enter the codes that are shown in Figure 8.45 into this newly stored procedure.

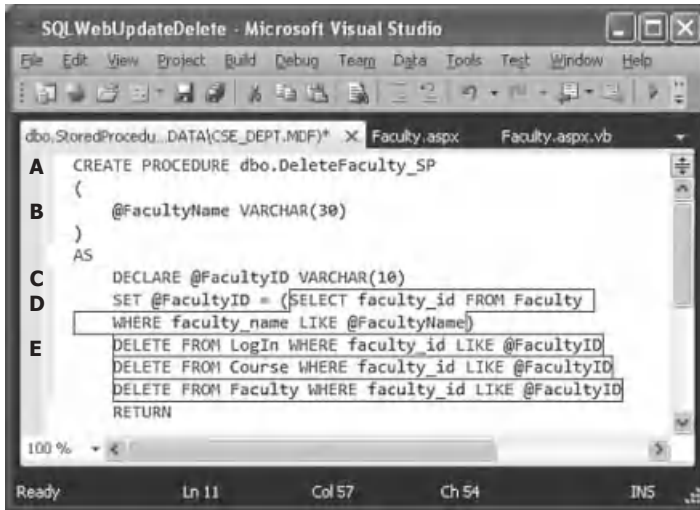


Figure 8.45. The stored procedure `dbo.DeleteFaculty`.

Let's take a closer look at this piece of codes to see how it works.

- A.** The stored procedure's name is `dbo.DeleteFaculty_SP`, and the prefix `dbo` is required by the SQL Server database to create any stored procedure.
- B.** This stored procedure has only one input parameter, which is the faculty name. So a nominal input parameter `@FacultyName` is defined in the input/output parameter list at the beginning of this stored procedure.
- C.** A local variable `@FacultyID` is declared, and it is used to hold the returned `faculty_id` from the execution of the data query to the Faculty table in step **D**.
- D.** A data query is executed to pick up the matched `faculty_id` from the Faculty table based on the input parameter `@FacultyName`.
- E.** After the `faculty_id` is obtained from the data query, three deleting queries are executed in the order that is shown in Figure 8.43 to perform three deleting operations. The order is: first, one must delete all records that use the `faculty_id` as the foreign keys from the child tables, such as the LogIn and the Course tables. Then one can delete the record that uses the `faculty_id` as the primary key from the parent table, such as the Faculty table.

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 newly stored procedure `dbo.DeleteFaculty_SP` from the Server Explorer window, and click on the **Execute** item from the pop-up menu to open the Run Stored Procedure wizard. Enter the input parameter **Ying Bai**, which is the faculty to be deleted from the Faculty table into the **Value** box, and your finished parameters wizard is shown in Figure 8.46.

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 8.47.

One point to be noted is the number of rows that are affected in Figure 8.47. It indicates that six rows are affected or deleted from our sample database, but this number is

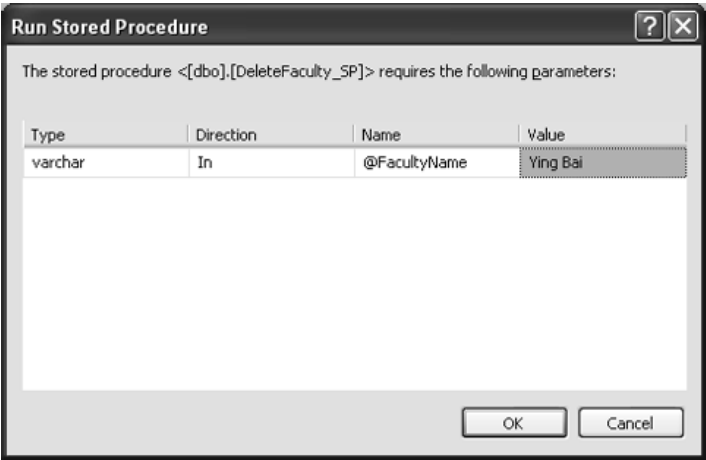


Figure 8.46. The finished Run Stored Procedure wizard.

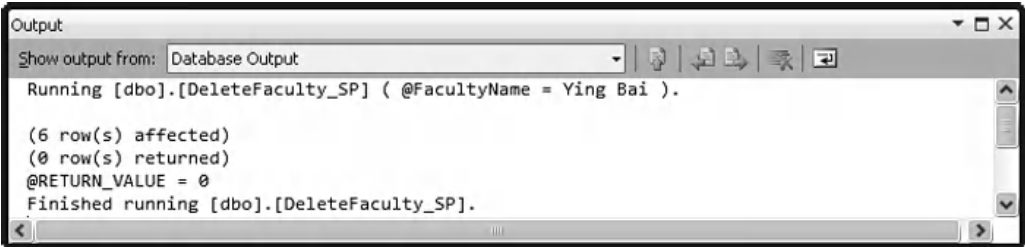


Figure 8.47. The running result of the stored procedure.

not the total number of rows that have been deleted from our database. According to the records built in our sample database, in total, there should be 11 rows that have been deleted from our database, which is shown in Table 8.7.

The reason for that is sometimes, the cascaded rows are not counted by this data deleting. In other words, some rows that are deleted by the SQL Server database engine are not included with this total number of affected rows, and this is a design deficiency.

To confirm this data deleting, open the following data tables from the Server Explorer window (Sometime you need to close the Visual Studio.NET and reopen the project to see these deleting results in these tables):

- LogIn table
- Faculty table
- Course table
- StudentCourse table

It can be found that all records listed in the Rows Affected in Table 8.7 have been deleted from the associated tables.

Table 8.7. The total number of rows affected or deleted

Table	Rows Affected	Number of Rows Affected
LogIn	user_name = ybai (faculty_id = B78880)	1
Course	course_id = CSC-132B (faculty_id = B78880) course_id = CSC-234A (faculty_id = B78880) course_id = CSE-434 (faculty_id = B78880) course_id = CSE-438 (faculty_id = B78880)	4
StudentCourse	s_course_id = 1005 (course_id = CSC-234A) s_course_id = 1009 (course_id = CSE-434) s_course_id = 1014 (course_id = CSE-438) s_course_id = 1016 (course_id = CSC-132B) s_course_id = 1017 (course_id = CSC-234A)	5
Faculty	faculty_id = B78880	1

Table 8.8. The 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 8.9. The data to be added into the LogIn table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	NULL

Another point to be noted is that we do not have to put all of three DELETE queries in this stored procedure to perform these data deleting actions; instead, we can only use one DELETE query: **DELETE FROM Faculty WHERE faculty_name LIKE @FacultyName**, to do the same function as these three queries did. The SQL Server engine can handle the data deleting actions from the child tables because of the cascaded deleting mode we have built for these tables in Chapter 2. For illustration purposes, we provide a complete picture with these deleting queries to show readers the details of this deleting function.

Next, we need to develop the codes in the ASP.NET environment to call this stored procedure to delete the selected faculty record from the database via our Web application project. However, before we can develop our codes, it is highly recommended to recover all deleted records from our sample database to make our database neat and complete.

To do that recovery job, you need to close the Visual Studio.NET and open the SQL Server Management Studio Express, and take the following actions in the following orders:

1. Recover the Faculty table by adding the deleted faculty record, which is shown in Table 8.8, into the Faculty table.
2. Recover the LogIn table by adding the deleted login record, which is shown in Table 8.9, into the LogIn table.

Table 8.10. The 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 8.11. The 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

3. Recover the Course table by adding the deleted courses taught by the deleted faculty member, which are shown in Table 8.10, into the Course table.
4. Recover the StudentCourse table by adding the deleted courses taken by the associated students, which are shown in Table 8.11, into the StudentCourse table.

An easy way to add data into the related tables in our sample database is to copy all rows from Tables 8.8 to 8.11 and paste them into the last line of the associated tables.

Save these changes and now we can close the SQL Server Management Studio and open the Visual Studio.NET to develop our codes to call the stored procedure to perform the data deleting actions against the SQL Server database.

8.5.3.5 Develop the Codes to Call the Stored Procedure to Perform the Data Deleting

On the opened Visual Studio.NET, go to the File|Open Web Site menu item to open our Web application project **SQLWebUpdateDelete**. Then open the **Delete** button's Click event procedure and enter the codes that are shown in Figure 8.48 into this event procedure.

Let's take a closer look at this piece of codes to see how it works.

- A. The content of the query string now is equal to the name of the stored procedure we developed in the Server Explorer window. This query string will be assigned to the CommandText property of the Command object later to inform it that a stored procedure needs to be executed to perform this data deleting action. Here, the name assigned to the

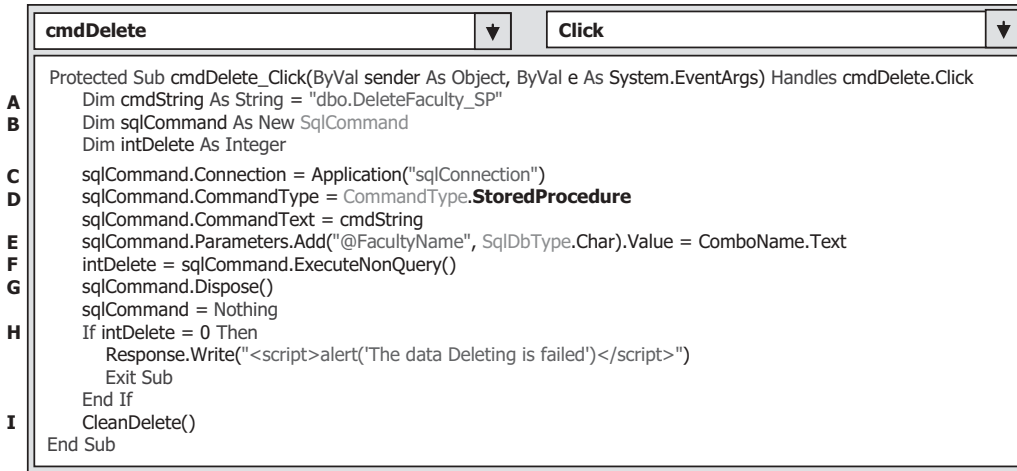


Figure 8.48. The codes for the Delete button's Click event procedure.

query string must be exactly identical with the name of the stored procedure we developed in the Server Explorer window; otherwise, an error would be encountered as the project runs since the page cannot identify the stored procedure if no matched name can be found.

- B.** The data object and local variable used in this procedure are declared here. The integer variable `intDelete` is used to hold the returned value from calling of the `ExecuteNonQuery()` method of the `Command` class later.
- C.** The `Command` object is initialized by assigning the connection object, which is a global variable and stored in the `Application` state, to the `Connection` property.
- D.** The `CommandType` property must be assigned to the `StoredProcedure` to inform the `Command` object that a stored procedure needs to be called when this `Command` object is executed. This is very important and should be distinguished with the general query text string.
- E.** The input parameter `@FacultyName`, which is the only input to the stored procedure, is assigned with the real parameter's value, and it is the faculty name stored in the Faculty Name combo box control in the Faculty page. Similarly, the name of this input parameter must be identical with the name of the input parameter used in the stored procedure we built earlier.
- F.** After the `Command` object is initialized, the `ExecuteNonQuery()` method of the `Command` class is called to run the stored procedure to perform the data deleting actions. This method will return a data value and assign it to the local variable `intDelete`.
- G.** A cleaning job is performed to release all objects used in this procedure.
- H.** 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 and the procedure is exited. Otherwise, if a nonzero value returned, which means that at least one row in our database has been deleted (all rows should be also deleted) from our database and this data deleting is successful.

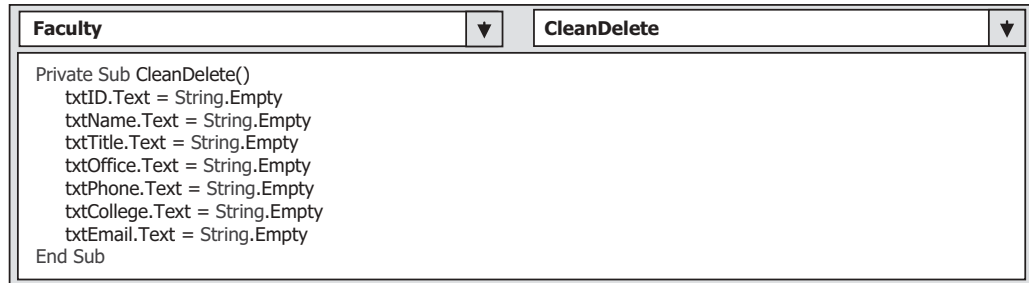


Figure 8.49. The codes for the subroutine CleanFaculty().

- I. A user-defined subroutine `CleanDelete()`, whose detailed codes are shown below, is executed to clean up the contents of all textboxes that stored the deleted faculty information.

The codes for the subroutine `CleanDelete()` are shown in Figure 8.49.

This piece of codes is easy to understand. All textboxes are cleaned up by assigning an empty string to their `Text` property.

At this point, we finished all coding jobs for deleting data against the SQL Server database using the stored procedure. 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, open the `Faculty` page, and keep the default faculty name `Ying Bai` selected in the `Faculty Name` combo box, and click on the **Select** button to retrieve and display this faculty's record. Click on the **Delete** button to run the stored procedure `dbo.DeleteFaculty_SP` to delete this faculty record from our database. Immediately, all pieces of information stored in seven textboxes are deleted.

To confirm this data deleting, open the SQL Server 2008 Management Studio and our sample database. You can find that all records related to that default faculty have been deleted from our database. Yes, our data deleting is successful.

Before you can close the SQL Server Management Studio, we highly recommend that you recover all deleted records in the associated tables. Refer to Tables 8.8–8.11 in the last section to add those records back the associated tables.

A complete Web application project `SQLWebUpdateDelete` can be found in the folder `DBProjects\Chapter 8` that is located at the Wiley ftp site (refer to Figure 1.2 in 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 selecting, adding, modifying, and deleting data against our sample database via Web pages. You know, LINQ to SQL queries can perform not only the data selection, but also the data insertion, updating, and deletion actions. The standard LINQ to SQL queries include:

- Select
- Insert
- Update
- Delete

To perform any of these operations or queries, we need to use the 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`. In this section, we want to create a Web-based project `SQLWebLINQ` by adding a graphic user interface to perform the data selection, data insertion, data updating, and deleting actions against our sample database `CSE_DEPT.mdf` using the LINQ to SQL query via Web pages. Let's perform the following steps to create our new project `SQLWebLINQ`:

1. Create a new Visual Basic.NET website project and name it as `SQLWebLINQ`.
2. Add an existing Web form page `FacultyLINQ.aspx` that can be found in the folder `VB Forms\Web` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).
3. Add the `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 scrolling down the `.NET` list, then selecting the item `System.Data.Linq` from the list and clicking on the OK button.
4. Add the following imports commands to the top of the `FacultyLINQ.aspx` page file:
 - `Imports System.Data.Linq`
 - `Imports System.Data.Linq.Mapping`
5. Follow 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 can be found in the folder `Database\SQLServer` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). 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 event procedure to validate our data insertion, data updating, and deleting actions, we need to divide our coding process into the following five parts:

- A. Create a new object of the `DataContext` class and do some initialization coding.
- B. Develop the codes for the **Select** button's Click event procedure to retrieve the selected faculty information using the LINQ to SQL query.
- C. Develop the codes for the **Insert** button's Click event procedure to insert a new faculty member using the LINQ to SQL query.

- D. Develop the codes for the **Update** button's Click event procedure to update the selected faculty member using the LINQ to SQL query.
- E. Develop the codes for the **Delete** button's Click event procedure to delete the selected faculty member using the LINQ to SQL query.

Before we can start the coding process, first, let's add an existing Web page **FacultyLINQ.aspx** as our graphic user interface to perform those data actions.

8.6.1 Add an Existing Web Page **FacultyLINQ.aspx**

Perform the following operations to add this existing Web page into our project:

1. Right-click on our new website project **SQLWebLINQ** from the Solution Explorer window and select the **Add Existing Item** to open the Add Existing Item wizard.
2. Browse to the folder **VB Forms\Web** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1) to find the Web page **FacultyLINQ.aspx**. Click this page to select it, and click on the **Add** button to add it into our project.
3. Right-click on this added page **FacultyLINQ.aspx** in the Solution Explorer window and select the **Set As Start Page** item to make this page as the starting page for our project.

Your added Web page **FacultyLINQ.aspx** is shown in Figure 8.50.

Perform steps 3–6 listed in the last section to add the **System.Data.Linq** reference and build five entity classes for this project. Now let's start our coding process for this page. First, let's handle creating a new object of the **DataContext** class in our project.

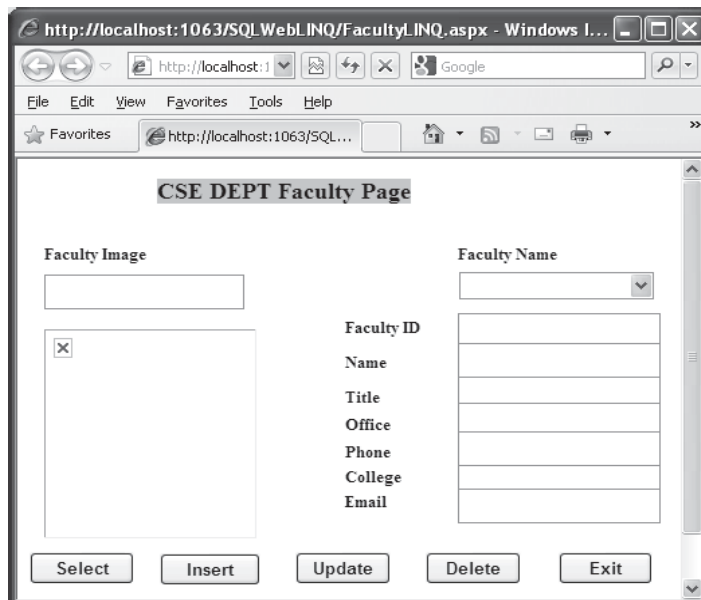


Figure 8.50. The added **FacultyLINQ.aspx** Web page.

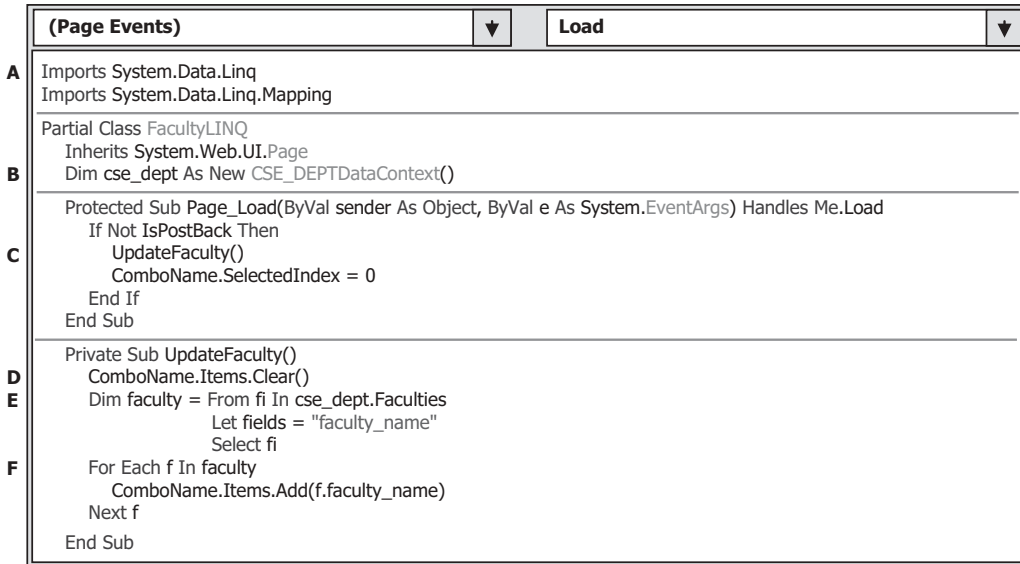


Figure 8.51. Initialization codes for the Faculty Web page.

8.6.2 Create a New Object of the 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 5 in Section 8.6, and a 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 codes include retrieving all updated faculty members from the Faculty table in our sample database using the LINQ to SQL query and displaying them in the Faculty Name combo box.

Open the code window and the Page_Load() event procedure of the Faculty Web page, and enter the codes that are shown in Figure 8.51 into this procedure.

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

- A.** Two LINQ-related namespaces are imported at the beginning of this page since we need to use some LINQ objects to perform data queries later.
- B.** A new form-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.
- C.** A user-defined subroutine `UpdateFaculty()` is executed to retrieve all updated faculty members from our sample database and display them in the Faculty Name combo box to allow the user 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 Faculty Name combo box 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 a request is sent to the server.

- D. Before we can update the Faculty Name combo box control 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.
- E. 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**.
- F. The LINQ query is executed to pick up all queried faculty members and add them into the Faculty Name combo box control in the Faculty Form.

The codes for the Exit button's Click event procedure are shown in Figure 8.52.

The function of this piece of codes is to close the entity object and the Web project.

8.6.3 Develop the Codes for the Data Selection Query

Double-click on the **Select** button to open its Click event procedure and enter the codes that are shown in Figure 8.53 into this procedure. The function of this piece of codes is to retrieve detailed information for the selected faculty member from the Faculty table in our sample database and display them in seven textbox controls in the Faculty Form page as this **Select** button is clicked by the user.

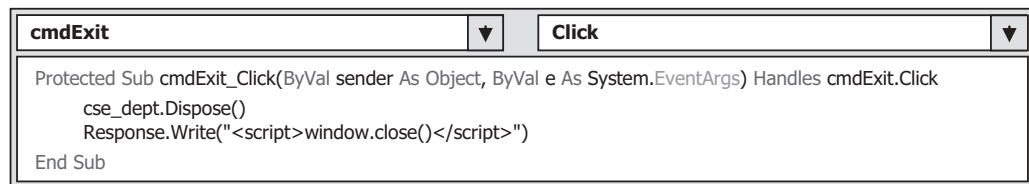


Figure 8.52. The codes for the Exit button Click event procedure.

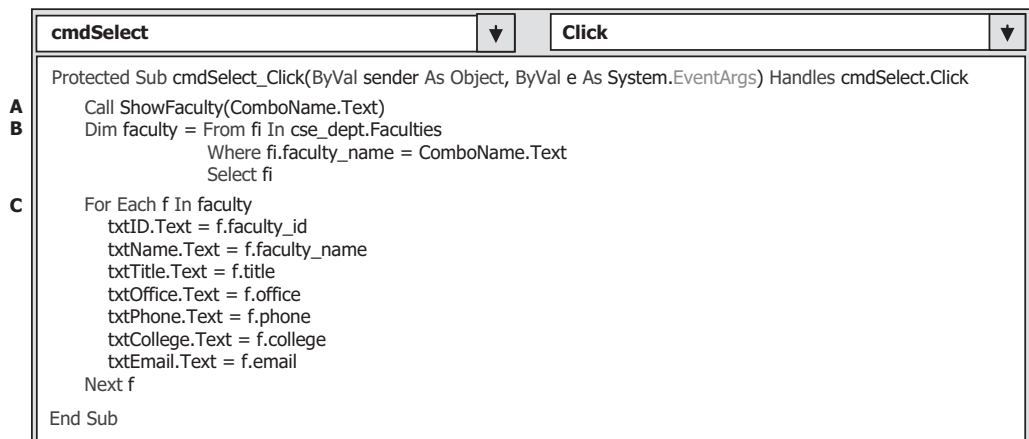


Figure 8.53. The codes for the Select button Click event procedure.

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

- A.** A user-defined subroutine procedure **ShowFaculty()** is executed to identify and display a matched faculty image for the selected faculty member.
- 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 in the associated textbox in the **Faculty Form** page.

The codes for the user-defined subroutine **ShowFaculty()** are shown in Figure 8.54.

The function of this piece of codes is simple. A **Select Case** structure is used to find the matched faculty image file, and display it in the **PhotoBox** with the **ImageUrl** property.

Now, let's concentrate on the coding development for the data insertion actions.

8.6.4 Develop the Codes for the Data Insertion Query

Double-click on the **Insert** button from our **Faculty Form** page to open its **Click** event procedure and enter the codes that are shown in Figure 8.55 into this procedure.

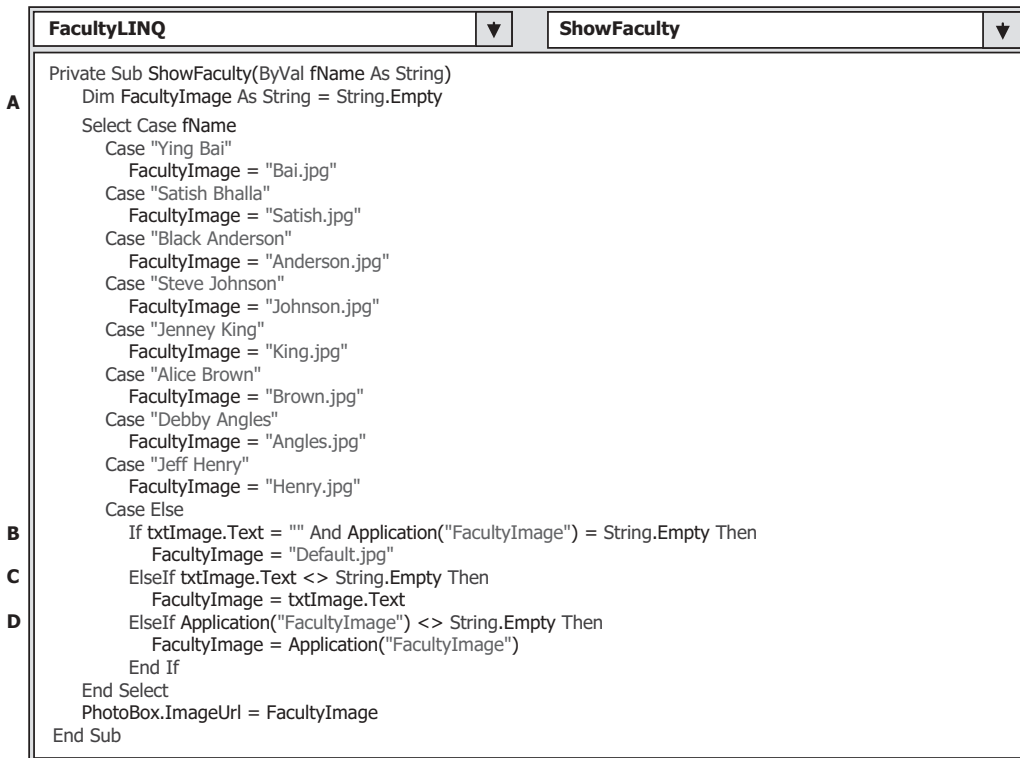


Figure 8.54. The codes for the user-defined subroutine **ShowFaculty()**.

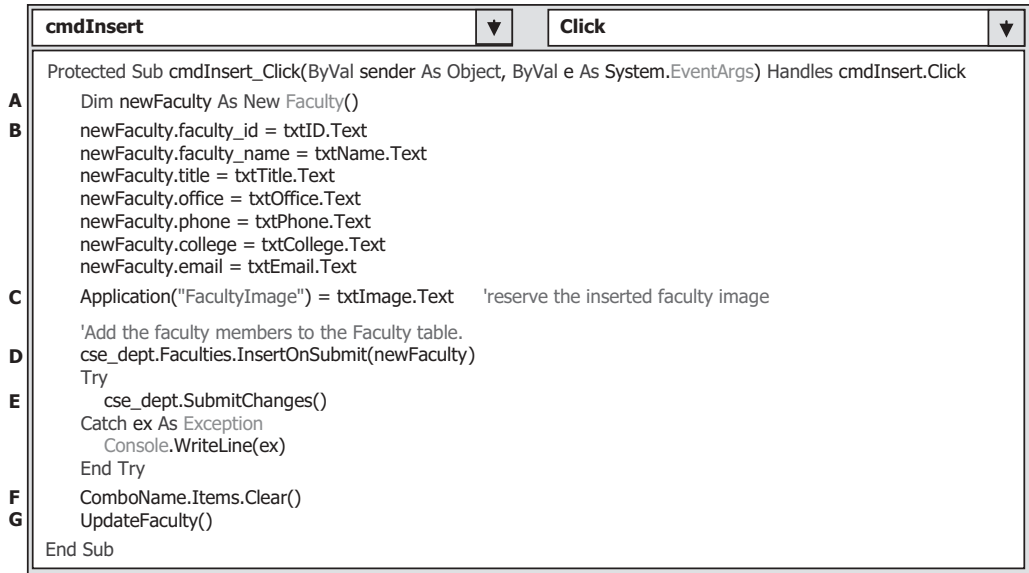


Figure 8.55. The codes for the Insert button Click event procedure.

Let's have a closer look at this piece of codes 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 the 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.** The newly inserted faculty image file is assigned to a global variable **FacultyImage** stored in the Application state function, and this will be used later for the insertion validation purpose. If no new faculty image is used for this insertion, the **txtImage.Text** will be an empty string that is assigned to the **FacultyImage**.
- D.** A system method **InsertOnSubmit()** is executed to send our newly created Faculty instance to our Faculty table via the **DataContext** class.
- E.** Another system method **SubmitChanges()** is executed to perform this data insertion. The point is that this method must be included in a **Try...Catch** block to avoid some possible unnecessary exceptions during the execution of this method.
- F.** After a new record has been inserted into our database, we need to update our Faculty Name combo box control to reflect that insertion. First, we need to clean up all original contents from this control to avoid multiple updating.
- G.** Then, the user-defined subroutine **UpdateFaculty()** is called to complete this faculty data updating.

Now let's begin the coding development for our data updating and deleting actions.

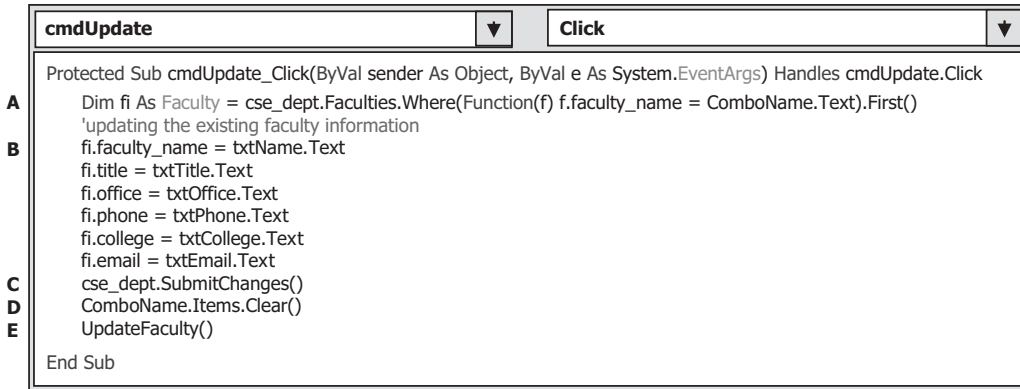


Figure 8.56. The codes for the Update button Click event procedure.

8.6.5 Develop the Codes for the Data Updating and Deleting Queries

First, let's build the codes for the data updating actions to the Faculty table in our sample database. Double-click on the **Update** button from our Faculty page window to open its Click event procedure and enter the codes that are shown in Figure 8.56 into this procedure.

Let's have a closer look at this piece of codes 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 matched record. This method does not have any effect to our application since we have only one record that is matched to this specified `faculty_name`.
- B.** All six columns, except the `faculty_id`, 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 Faculty Name combo box is cleaned up to make it ready to be updated.
- E.** The user-defined subroutine `UpdateFaculty()` is executed to refresh the updated faculty members stored in the Faculty Name combo box control.

Before we can run our Web project to test these data actions, let's complete the last coding development for our data deleting action.

Double-click on the **Delete** button from our Faculty page window to open its Click event procedure and enter the codes that are shown in Figure 8.57 into this procedure.

Let's have a closer look at this piece of codes 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 an instance of our entity class Faculty, and the `faculty_name` works as the query criterion for this query. All pieces of information related to the

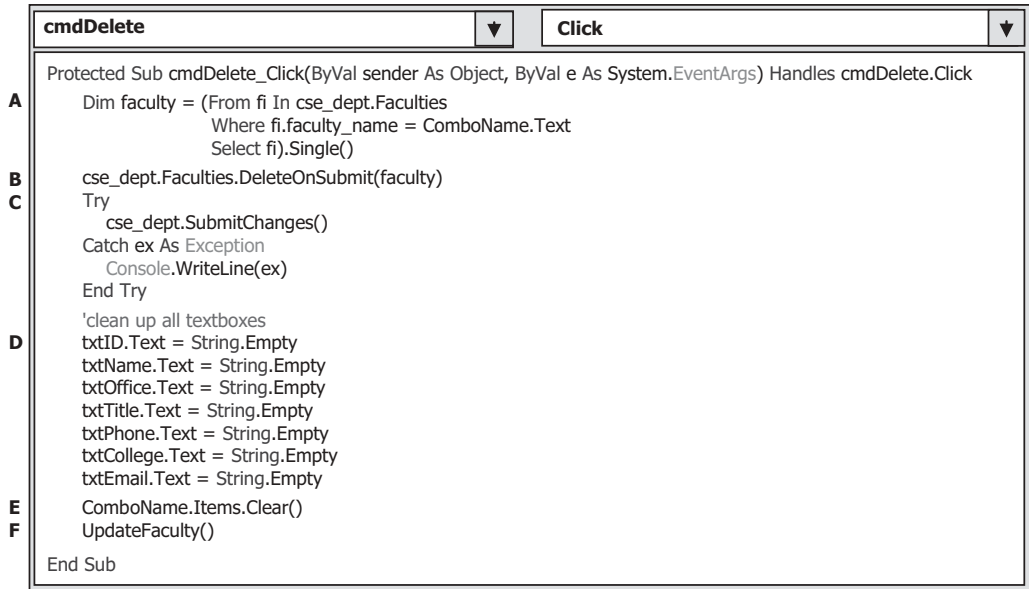


Figure 8.57. The codes for the Delete button Click event procedure.

selected faculty member (**faculty_name**) will be retrieved and stored in the query variable **faculty**. The **Single()** method 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.** A **Try...Catch** block is used to execute another system method **SubmitChanges()** to exactly perform this deleting action against the data table in our sample database. The point is that this method must be included in this block to avoid some unnecessary exceptions during the execution of this method. Only after this method is executed can the selected faculty record be 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 Faculty Name combo box is cleaned up to make it ready to be updated.
- F.** The user-defined subroutine **UpdateFaculty()** is executed to reflect this faculty record deleting for all faculty members stored in the Faculty Name combo box.

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**. You can find all faculty and student image files from the folder **Images** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Click on the **Start Debugging** button to run our project. Click on the **Yes** button on the pop-up message box to enable the debugging function as the project runs. Click on the **Select** button to test the faculty data query function.

Now let's test the data insertion action. Enter the following eight pieces of information into eight textboxes as a new faculty record:

- P77777 Faculty ID textbox
- Peter Tom 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
- Default.jpg Faculty Image textbox

Then click on the **Insert** button to perform this data insertion.

To confirm this data action, first select another faculty member from the Faculty Name combo box and click on the **Select** button to retrieve and display that faculty's record. Then select the newly inserted faculty **Peter Tom** from the Faculty Name combo box, and click on the **Select** button to try to retrieve this newly inserted faculty's record and display it in this page. Your data insertion confirmation page should match the one that is shown in Figure 8.58.

A default faculty image is displayed for this data insertion, since we placed a default faculty image file **Default.jpg** into the **Faculty Image** textbox for this insertion. You can

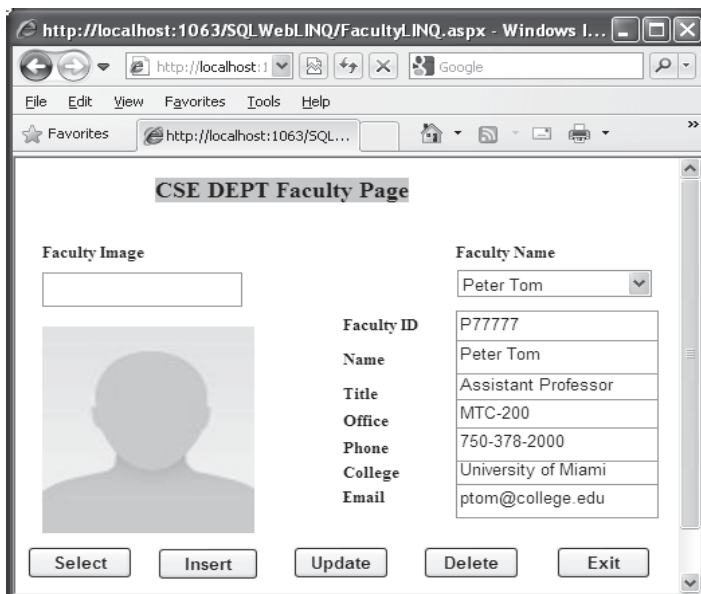


Figure 8.58. The testing status of the data insertion action.

test to insert a new faculty with a selected faculty image by entering the name of that faculty image file into the **Faculty Image** textbox if you like.

You can continue to test the data updating and deleting functions for this project. However, one point to be noted is 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 sample database neat and complete. Refer to Tables 8.8–8.11 in Section 8.5.3.4 in this chapter to recover the deleted records in our sample database. You can do this recovery job using either the Server Explorer in Visual Studio.NET 2010 IDE or the SQL Server Management Studio.

A complete Web page application project **SQLWebLINQ** can be found in the folder **DBProjects\Chapter 8** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

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 the SQL Server and the Oracle database actions. Also, in order to save time and space, we will modify an existing Web application project **SQLWebSelect** we developed in the last section to make it as our new project **OracleWebSelect** in this section.

The main coding differences that exist between these two database operations are:

1. Oracle Database reference and Imports commands in all pages
2. Connection string in the LogIn page
3. LogIn query string in the LogIn page
4. Query string in the Faculty page
5. Query strings in the Course page, which include the query string in the **Select** button's click event procedure and the query string in the **SelectedIndexChanged** event procedure of the **CourseList** box control.
6. Data objects used in the Selection page
7. Prefix for all data objects and classes used for the Oracle database operations.
8. Data type of the passed arguments in all user-defined subroutine procedures for Oracle database operations

Now let's begin to modify the project **SQLWebSelect** based on the eight differences listed above to make it our new project **OracleWebSelect**. Open the Windows Explorer and create a new folder **Chapter 8** if you have not created it. Copy the project **SQLWebSelect** from the folder **DBProjects\Chapter 8** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1) and paste it into the folder **Chapter 8**. Rename the project to **OracleWebSelect**.

Open Visual Studio.NET, go to the **File/Open Web Site** menu item, and browse to our new project **OracleWebSelect**. Then click on the **Open** button to open it.

First, let's add an Oracle database reference into our project and modify all Imports commands for all Web pages in this new project.

8.7.1 Add the Oracle Database Reference and Modify Imports Commands

In this section, we will use the Oracle Database 11g Express Edition as our database source and provider. Refer to Appendix B for detailed procedures to download, install, and configure this database on your computer.

Starting from .NET Framework 4.0, Microsoft no longer support Oracle database related operations. Therefore, we need to use an Oracle database driver provided by a third-party vendor. We will use an Oracle database driver dotConnect for Oracle 6.30 Express provided by Devart. Refer to Appendix F to get detailed information about how to download and configure this driver in your machine.

After installing Oracle Database 11g XE and dotConnect for Oracle 6.30 Express in your machine, perform the following operations to add this Oracle database reference into this new project:

1. Right-click on our new project **OracleWebSelect** from the Solution Explorer window and select the **Add Reference** item from the pop-up menu to open the Add Reference wizard.
2. With the .NET tab selected, scroll down the list until you find the items **Devart.Data** and **Devart.Data.Oracle**, then click on both to select them, and click on the OK button to add these two references to our project.

Now, let's modify all Imports commands in all Web pages in this new project to make them match to the Oracle data source. Let's start from the LogIn page. Open the code window of the LogIn page and change Import commands on the top of this page to:

```
Imports System.Data
Imports Devart.Data
Imports Devart.Data.Oracle
```

Perform a similar modification for these Import commands for all other pages in this project, including the Selection, Faculty and Course pages.

Next, let's modify the codes of the connection string in the LogIn page.

8.7.2 Modify the Connection String in the LogIn Page

Open the Page_Load() event procedure by selecting the (Page Events) item from the Class Name combo box and Load item from the Method Name combo box. Perform the modifications shown in Figure 8.59 to this event procedure.

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

- A. Modify all Imports commands on the top of this page to provide references for all Oracle data components used in 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 the Oracle database.

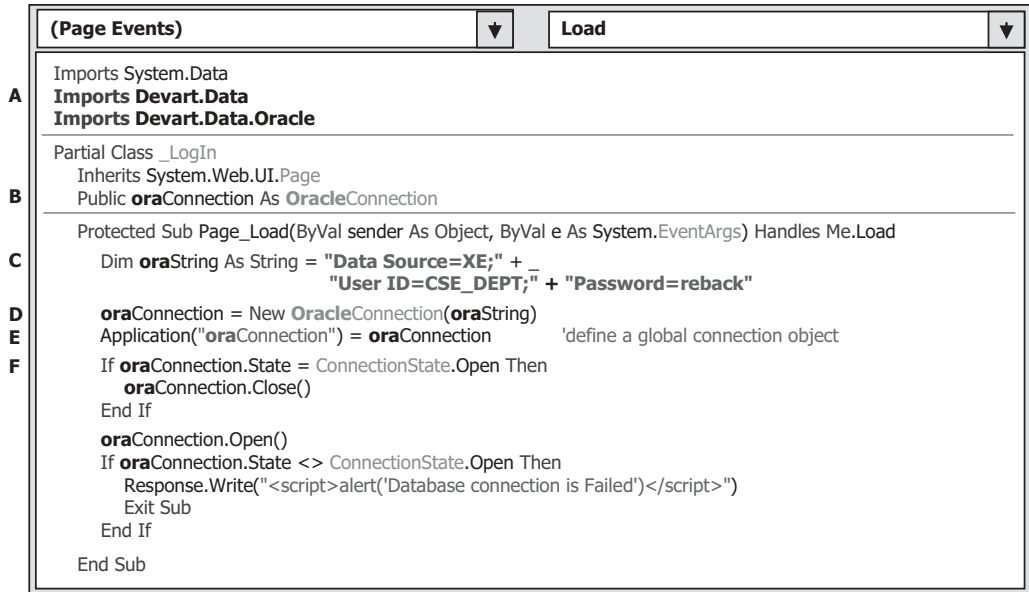


Figure 8.59. The modified connection object and connection string.

- D. Create a new instance of the Oracle connection class with the Oracle connection string `oraString` as the argument. Also, change the prefix for all Oracle data classes and objects from `Sql` to `Oracle`, and from `sql` to `ora`, respectively.
- E. Change the prefix for the global connection object stored in the Application state from `sql` to `ora`.
- F. Change the prefix for all data components from `sql` to `ora`.

Your finished modifications to the `Page_Load()` event procedure and the connection string should match those that are shown in Figure 8.59. All modified parts have been highlighted in bold.

8.7.3 Modify the Query String in the LogIn Page

Open the LogIn button's Click event procedure and perform the modifications shown in Figure 8.60 to this event procedure.

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

- A. Change the query string from the SQL Server database-based to the Oracle database-based. The Oracle database comparison operator `=:` is used to replace the SQL Server database comparison operator `LIKE @`.
- B. Change the prefix for all data objects 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.

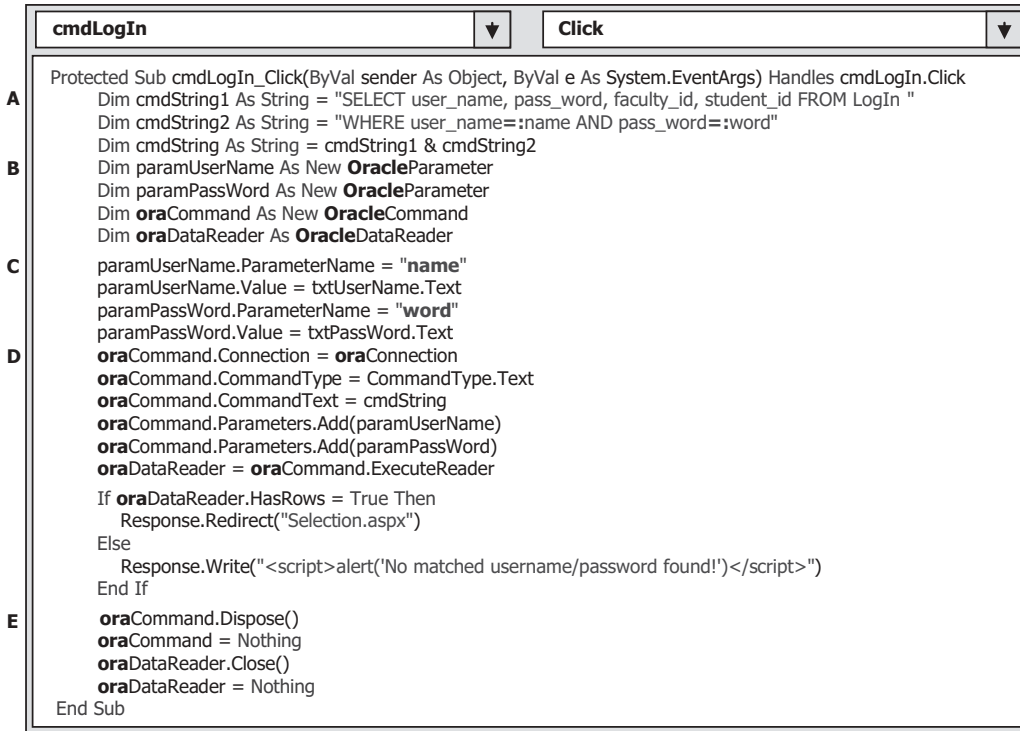


Figure 8.60. The modifications to the codes in the LogIn button event procedure.

- 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**.

Your finished modifications to this event procedure should match the one that is shown in Figure 8.60. All modified parts have been highlighted in bold.

Another modification to this page is the modifications to the codes in the Cancel button's click event procedure. This modification is simple and just changes the prefix of all data objects from **sql** to **ora**.

Go to the File|Save All menu item to save these modifications.

8.7.4 Modify the Query String in the Faculty Page

The modifications to this page include the following parts:

1. Modifications to the global connection object stored in the Application state in the Page_Load() event procedure.
2. Modifications to the codes in the Select button's click event procedure.
3. Modifications to the data type of the passed argument in the user-defined subroutine FillFacultyReader().

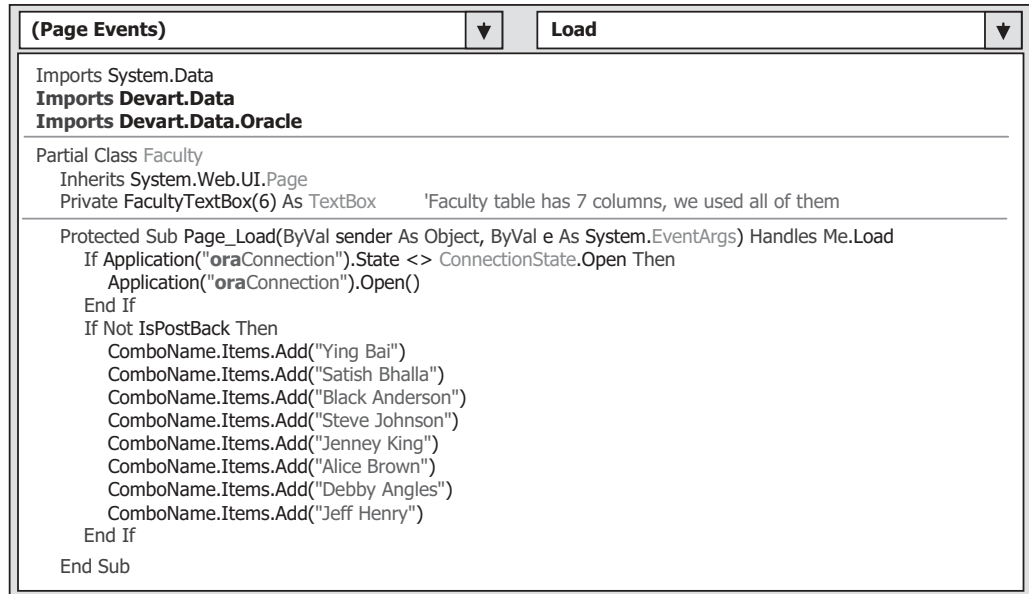


Figure 8.61. The modified Page_Load() event procedure.

Open the Page_Load() event procedure and change the connection object stored in the Application state from `sqlConnection` to `oraConnection`. Your finished modifications to this event procedure should match the one that is shown in Figure 8.61. The modified parts have been highlighted in bold.

Now open the **Select** button's click event procedure and perform the modifications shown in Figure 8.62 to this procedure.

- A.** Change the query string by replacing the SQL Server database comparison operator `LIKE @` with the Oracle comparison 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 nominal name of the dynamic parameter by removing the `@` symbol before the parameter `facultyName`.
- D.** Modify the global connection object stored in the Application state from the `sqlConnection` to the `oraConnection`.
- E.** Change the prefix for all data objects and classes from `sql` to `ora` and from `Sql` to `Oracle`.

Your finished modifications to this event procedure should match the one that is shown in Figure 8.62. All modified parts have been highlighted in bold.

The Modification to the data type of the passed argument in the user-defined subroutine `FillFacultyReader()` is simple. Just change the data type of the passed argument `FacultyReader` from the `SqlDataReader` to the `OracleDataReader`.

Next, let's handle the modifications to the query strings in the Course page.

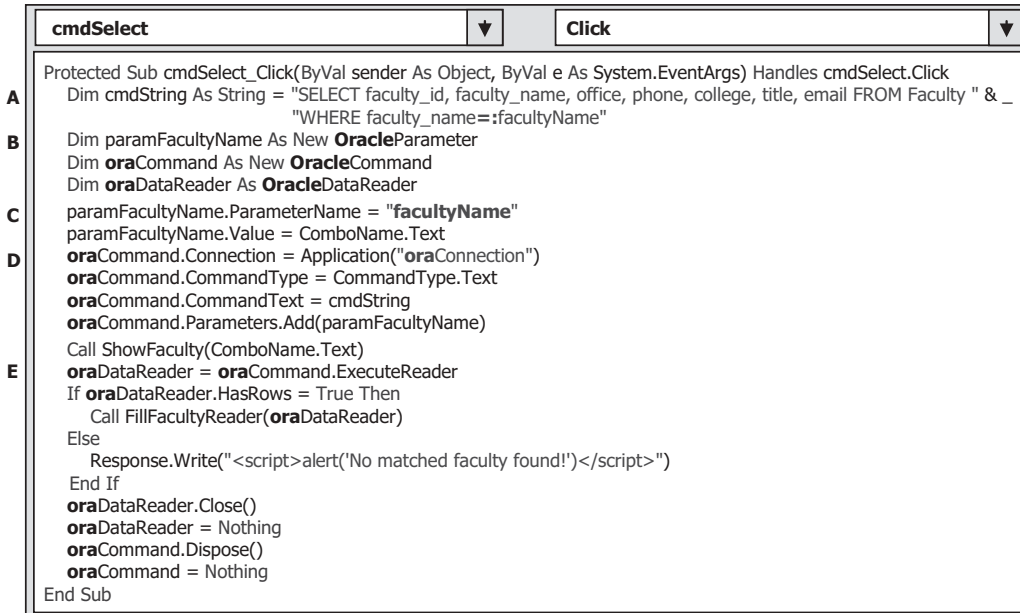


Figure 8.62. The modifications to the Select button's event procedure.

8.7.5 Modify the Query Strings in the Course Page

The modifications to this page include the following contents:

1. Modifications to the global connection object stored in the Application state in the Page_Load() event procedure.
2. Modifications to the codes in the Select button's click event procedure.
3. Modifications to the codes in the SelectedIndexChanged event procedure of the list box control CourseList.
4. Modifications to the data type of the passed argument in the user-defined subroutines FillCourseReader() and FillCourseReaderTextBox().

Open the Page_Load() event procedure and change the connection object stored in the Application state from `sqlConnection` to `oraConnection`. Your finished modifications to this event procedure should match the one that is shown in Figure 8.63. The modified parts have been highlighted in bold.

Now open the Select button's click event procedure and perform the modifications shown in Figure 8.64 to this procedure.

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

- A. The query string applied to the joined table should be modified to make it acceptable to the Oracle database actions. In Oracle database, the comparison operator is `=:`, not `LIKE@`, which used for the SQL Server query string. Modify this query string and replace `LIKE@` with `=:` in the `CString2` for this query action, which is shown in Figure 8.64.

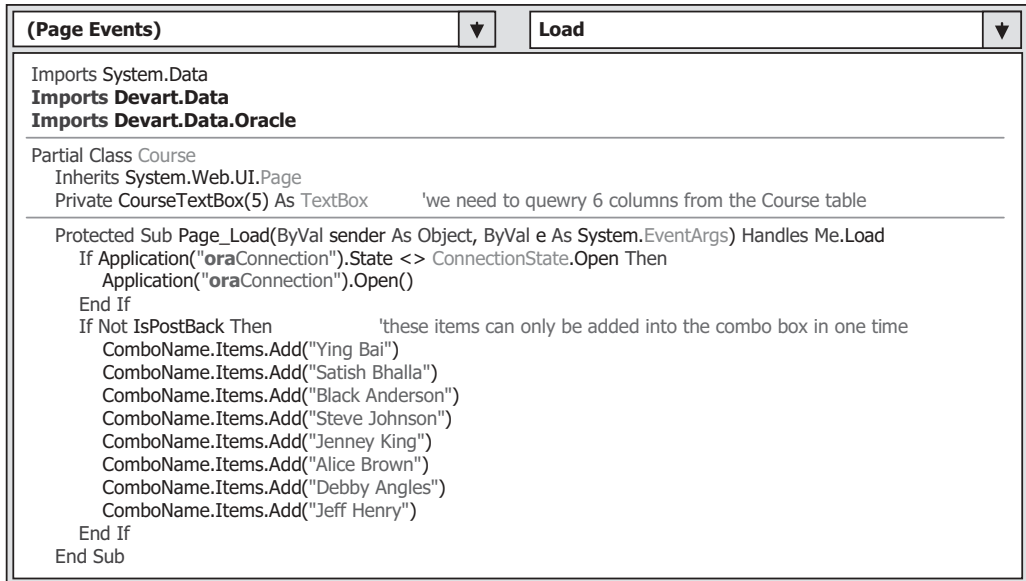


Figure 8.63. The modified Page_Load event procedure.

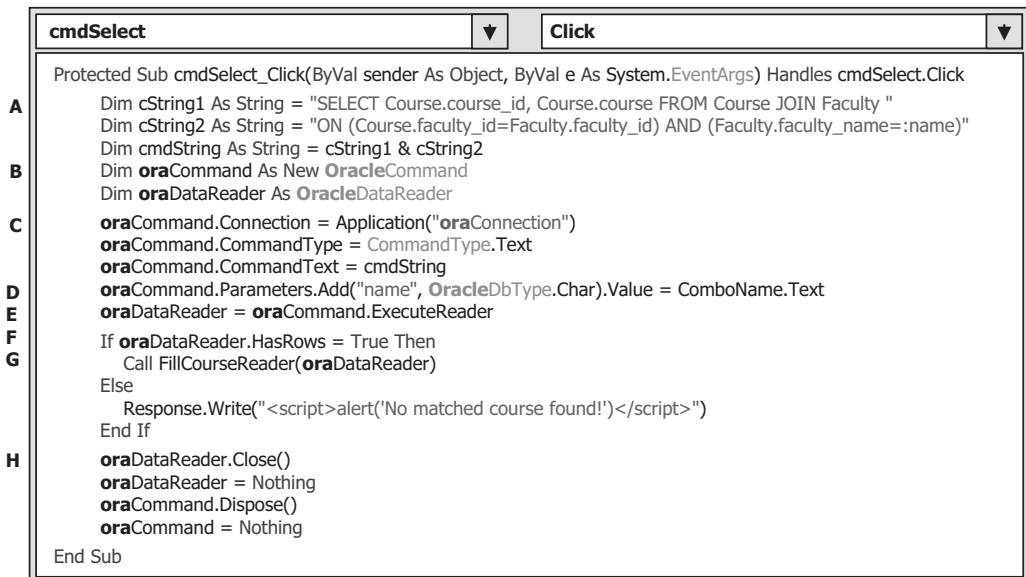


Figure 8.64. The modified Select button's Click event procedure.

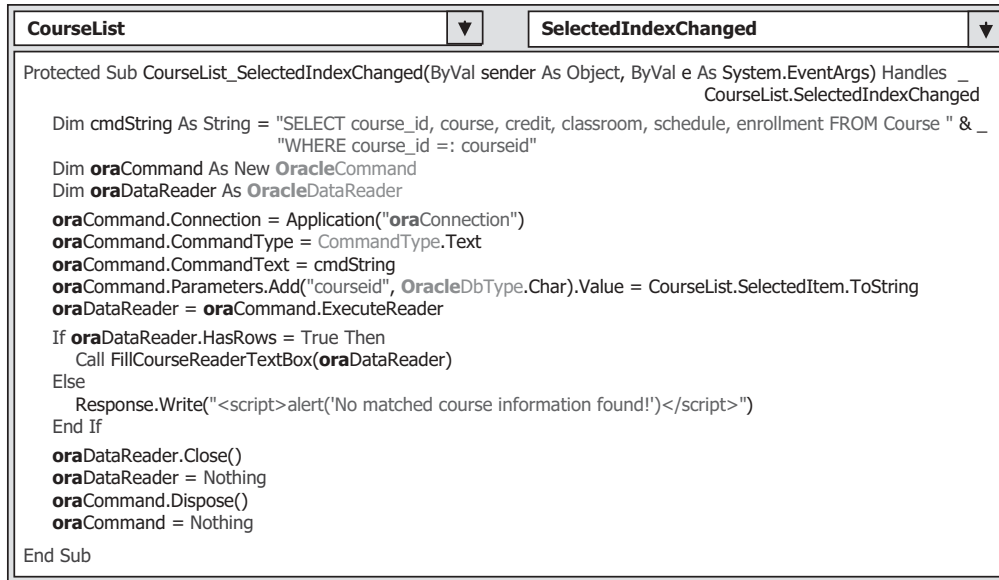


Figure 8.65. The modified SelectedIndexChanged event procedure.

- 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.** Change the prefix for all data objects from `sql` to `ora` and change the data type of the dynamic parameter name from `SqlDbType` to `OracleDbType`.
- E.** Change the prefix for all data objects from `sql` to `ora`. The steps involved in this modification include **E** through **H**.

Next, open the `SelectedIndexChanged` event procedure of the list box control `CourseList`, and perform the modifications shown in Figure 8.65 to this procedure.

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

- A.** Modify the comparison operator for the dynamic parameter `courseid` in the query string by replacing `LIKE @` with the Oracle 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 from the `sqlConnection` to the `oraConnection`.
- D.** Modify the nominal name of the dynamic parameter by removing the `@` symbol before the parameter `courseid`. Also, change the data type for this dynamic parameter from `SqlDbType` to `OracleDbType`.
- E.** Change the prefix for all data objects from `sql` to `ora`. The steps involved in this modification include **E** and **F**.

Modifications to the data type of the passed argument in the user-defined subroutines `FillCourseReader()` and `FillCourseReaderTextBox()` are simple, and just change the data type of that passed argument from the `SqlDataReader` to the `OracleDataReader`.

8.7.6 Modify the Global Connection Object in the Selection Page

The last modification is to change the global connection object stored in the Application state from the `sqlConnection` to the `oraConnection` in the Exit button's click event procedure in the Selection page.

At this point, we finished all modifications to this project. Before we can run the project to test the functions of our codes, the following two jobs must be performed:

1. Make sure that all faculty image 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 the 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 wizard. On the opened wizard, 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 wizard. In the opened wizard, click on the `LogIn.aspx` from the list and click on the OK button 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 functions of our codes in these pages.

A complete Web application project `OracleWebSelect` can be found in the folder `DBProjects\Chapter 8` that is located at the Wiley ftp site (refer to Figure 1.2 in 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 on the codes for these two databases. To save time and space, we need to modify an existing project `OracleWebSelect` to make it as our new project `OracleWebInsert`. The codes we need to add can be copied from another existing project `SQLWebInsert` with some modifications.

The main codes' differences in these two database operations are:

- A. The added codes to the Insert button's Click event procedure in the Faculty page.
- B. The added codes to the TextChanged event procedure of the Faculty ID textbox.
- C. The modified codes to the subroutine `ShowFaculty()` in the Faculty page.

Now, let's modify the project `OracleWebSelect` to make it our new project `OracleWebInsert` by performing the following operations:

1. Open the Windows Explorer and create a new folder **Chapter 8** under your **C:** drive if you have not created it.
2. Copy the project **OracleWebSelect** from the folder **DBProjects\Chapter 8** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1) and paste it into our new created folder **C:\Chapter 8**.
3. Rename the project to **OracleWebInsert**.

Open the Visual Studio.NET, go to the **File|Open Web Site** menu item and browse to our new project **OracleWebInsert**, and then click on the **Open** button to open it. First, let's build the codes for the **Insert** button Click event procedure to perform the faculty data insertion function to the **Faculty** table in our sample database via the Web page **Faculty.aspx**.

8.8.1 Create the Codes for the Insert Button Click Event Procedure

Open this event procedure and enter the codes that are shown in Figure 8.66 into this event procedure.

Let's have a closer look at this piece of newly added codes to see how it works.

- A. Change the query string from the SQL Server database style to the Oracle database style. This modification includes replacing all **@** symbols before each input parameter with the **:** operator, which is an Oracle database operator.
- B. The data components and local variables used in the procedure are declared here. The local integer variable **intInsert** is used to hold the returned running result from the execution of the data insertion command.

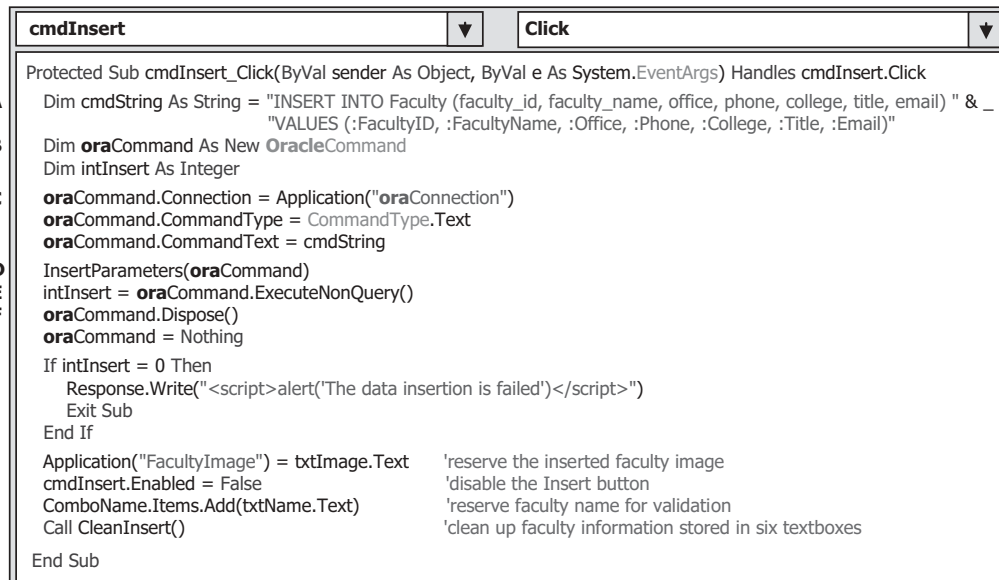


Figure 8.66. The added codes to the Insert button's event procedure.

- C. The Command object is initialized by assigning it with the connection object stored in the Application state, the command type, and the command text objects, respectively.
- D. The user-defined subroutine `InsertParameters()` is executed to assign all seven input parameters to the Parameters collection of the command object.
- E. The `ExecuteNonQuery()` method of the command object is called to run the insert query to perform this data insertion.
- F. A cleaning job is performed to release all objects used in the procedure.
- G. The `ExecuteNonQuery()` method will return an integer 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 returned, which means that no any row has been inserted into the database, a warning message is displayed to indicate this situation and the procedure is exited. Otherwise, the data insertion is successful.
- H. A global variable `FacultyImage` is created and initialized with the faculty image file name stored in the Faculty Image textbox. In some cases, the user may want to add a faculty image with that faculty record insertion. In order to save this image file for the data validation, we need this step.
- I. 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.
- J. The newly inserted faculty name is added into the Faculty Name combo box by using the `Add()` method, and this faculty name will be used later for the validation purpose.
- K. The user-defined subroutine procedure `CleanInsert()` is executed to clean up six textboxes in the Faculty page (except the Faculty ID textbox).

Next, let's build the subroutine `InsertParameters()`. Create this subroutine by entering the codes that are shown in Figure 8.67 into this page.

The function of this subroutine is straightforward, which is to assign all input parameters to the associated `VALUES` columns in the insert query.

The codes for the user-defined subroutine `CleanInsert()` are shown in Figure 8.68.

The function of this piece of codes is to clean up contents of six textboxes, except the `faculty_id` textbox. The reason for that is: the Insert button would be enabled if the content of the `faculty_id` textbox is cleaned up (changed), since a `TextChanged` event will

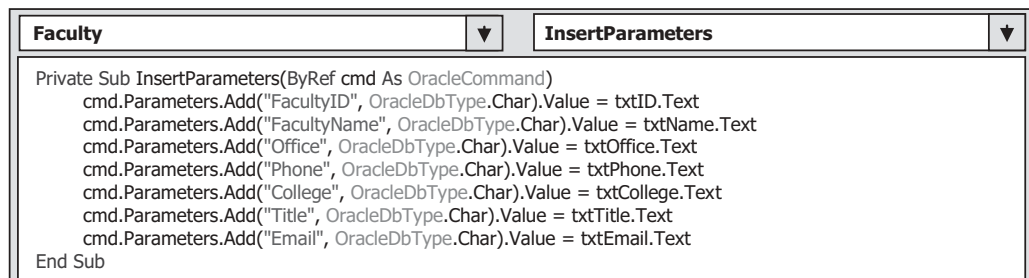


Figure 8.67. The codes for the subroutine `InsertParameters()`.

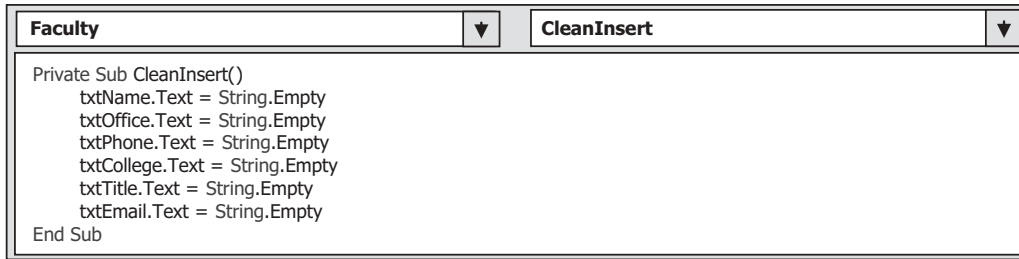


Figure 8.68. The codes for the subroutine CleanInsert().

be triggered. However, this cleaning up action has nothing to do with inserting a new record. Therefore, in order to avoid this confusing operation, we will not clean up the `faculty_id` textbox.

Next, let's handle the coding process for the `TextChanged` event procedure of the Faculty ID textbox.

8.8.2 Create the Codes for the TextChanged Event Procedure of the Faculty ID Textbox

When the content of the `faculty_id` textbox is changed (a `TextChanged` event of the `faculty_id` textbox will be triggered), which means that a new faculty record should be inserted, we need to enable the `Insert` button if this situation happened. To do this piece of codes, double-click on the `faculty_id` textbox from the Faculty page to open its `TextChanged` event procedure and enter `cmdInsert.Enabled = True` into this procedure.

8.8.3 Modify the Codes in the Subroutine ShowFaculty() for the Data Validation

In order to validate this data insertion action, we need to modify some codes inside the user-defined subroutine `ShowFaculty()` to enable a newly inserted faculty image to be retrieved and displayed in this page if the user wants to add a new faculty image for that data insertion.

Open this subroutine and perform the modifications, which are shown in Figure 8.69, to this procedure. The modified parts have been highlighted in bold.

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

- A. The local variable `FacultyImage` is initialized with an empty string.
- B. To check whether a new faculty image has been inserted or no matched faculty image has been found, we use an `And` logic operator to combine both conditions together. If both of them are empty, which means that no matched faculty image can be found, a default faculty image is used.
- C. If the Faculty Image textbox contains a valid faculty image file's name, it is assigned to the local String variable `FacultyImage` and displayed later.

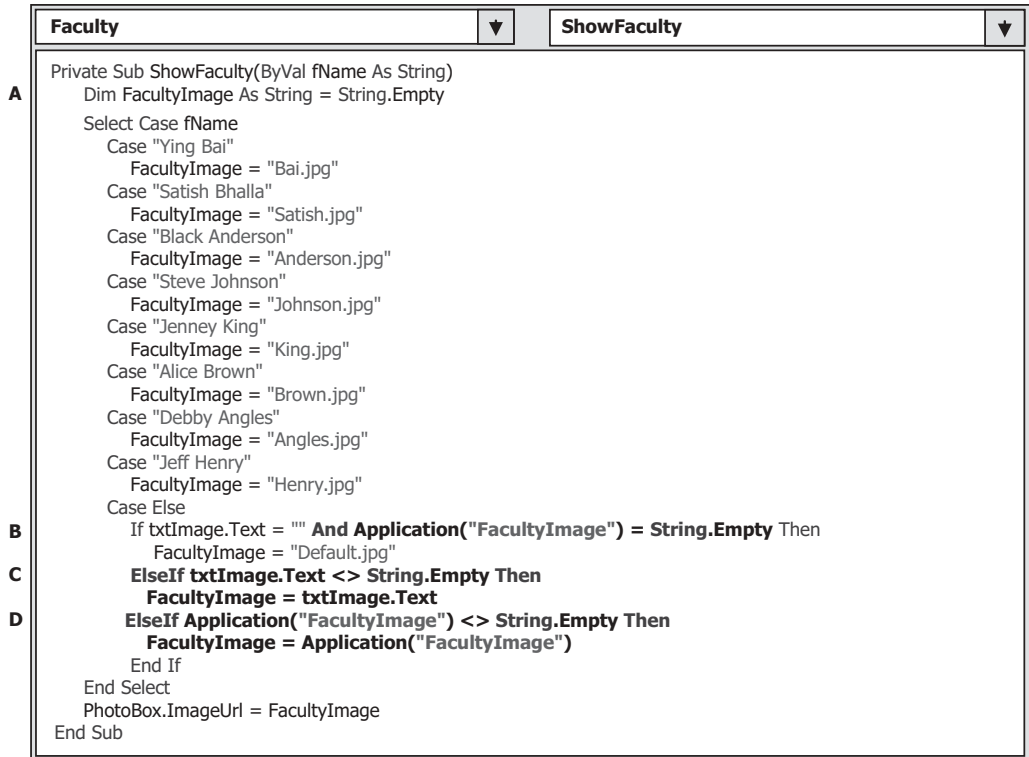


Figure 8.69. The modified codes for the subroutine ShowFaculty().

- D.** If the global variable **FacultyImage** is not empty, which means that a valid faculty image's name has been assigned to it by the user, and this faculty image will be used and displayed later.

At this point, we have finished all modifications to our new project. Before we can run the project to test the data insertion functionality, make sure that the following two jobs have been done:

1. Make sure that all faculty image files, including a default faculty file **Default.jpg**, have been stored in 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.aspx**. To confirm this, right-click on our project from the Solution Explorer Window, select the **Start Options** item from the pop-up menu. On the opened wizard, 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 wizard.

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. Enter the following data as the information for a new faculty member:

- M56789 Faculty ID textbox
- Ali Mhamed Name textbox
- Professor Title textbox
- MTC-353 Office textbox
- 750-378-3355 Phone textbox
- University of Main College textbox
- amhamed@college.edu Email textbox
- Mhamed.jpg Faculty Image textbox

Click on the **Insert** button to insert this new record into the database.

To confirm this faculty record insertion, go to the Faculty Name combo box control and you can find that the newly inserted faculty name **Ali Mhamed** has already been in there. Click it to select this faculty and then click on the **Select** button to retrieve this newly inserted record from the database and display it in this page. The inserted record is displayed in this page, which is shown in Figure 8.70.

Our data insertion to the Oracle database is successful. Click on the **Back**, and then the **Exit** button to close our project. A complete Web application project **OracleWebInsert** can be found at the folder **DBProjects\Chapter 8** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

In the next section, we will discuss how to perform the data updating and deleting against the Oracle database via the website.

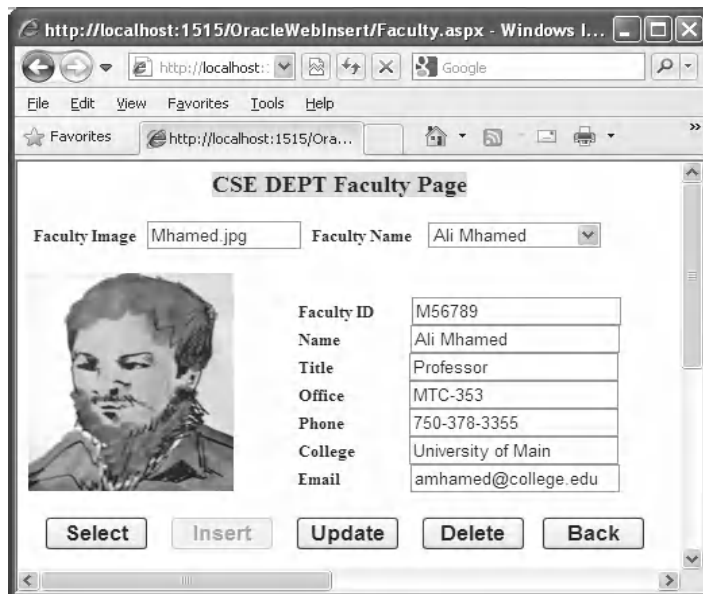


Figure 8.70. The data validation process.

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 codes 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 the Windows Explorer and create a new folder `Chapter 8` if you have not created it. Copy the project `OracleWebInsert` from the folder `DBProjects\Chapter 8` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1) and paste it to our new folder `Chapter 8`. Rename the project to `OracleWebUpdateDelete`.

We divide this section into two parts in terms of the coding functions:

1. Build the codes for the new project to perform the data updating actions against the Oracle database.
2. Build a stored procedure to perform the data deleting actions against the Oracle database.

In fact, we built a project `SQLWebUpdateDelete` to update and delete data against our SQL Server database in Section 8.5. The only difference between that project and our current project is the data source or database used for these projects. All functions and codes are similar between these projects. Therefore, you can copy some codes from the associated event procedures in that project and paste them in this project with a little modification.

Now, let's start from the first part—build the codes for the new project to make it perform the data updating against the Oracle database.

8.9.1 Build the Codes for the Project to Perform the Data Updating

Open the Visual Studio.NET, go to the `File\Open Web Site` menu item and browse to our new project `OracleWebUpdateDelete` and then click on the `Open` button to open it.

The modifications to this page can be divided into the following two parts:

1. Modify the `Select` button's click event procedure by adding one statement to reserve the original or the old faculty name stored in the `Faculty Name` combo box control for the possible faculty name updating operation later.
2. Add the codes to the `Update` button's click event procedure and the user-defined subroutine procedure `UpdateParameters()`.

Let's begin with the first modification.

8.9.1.1 Modifications to the Select Button's Click Event Procedure

Now, open the `Select` button's click event procedure and add one statement into this event procedure. Your finished modifications to this event procedure should match the

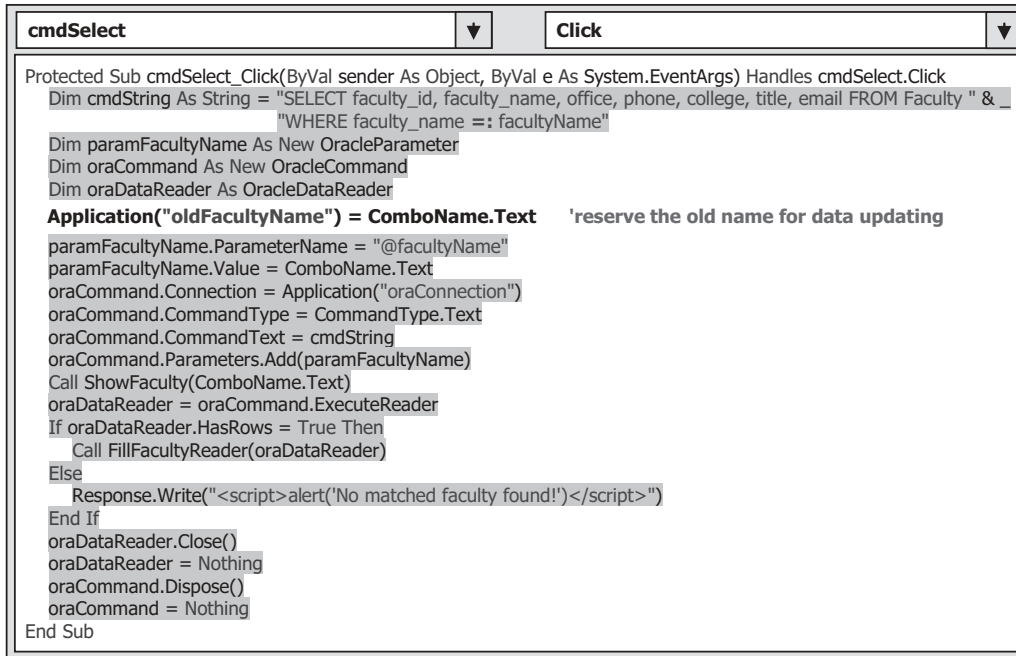


Figure 8.71. The modified Select button's event procedure.

one that is shown in Figure 8.71. The newly added statement has been highlighted in bold, and the codes we developed in the previous section have been highlighted with gray.

The purpose of this statement is to store the current selected faculty name that is located at the Faculty Name combo box control into the Application state as a global variable. During the data updating process, the faculty name may be updated by the user. If this happened, the updated faculty name that is stored in the txtName textbox will be added into the Faculty Name combo box control, and the original faculty name will 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 when 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 staff will be lost.

Now let's develop the codes for the Update button's click event procedure.

8.9.1.2 Add the Codes to the Update Button Event and UpdateParameters Procedures

In order to save the time, you can copy some codes from the Update button's Click event procedure in the Faculty page in a project SQLWebUpdateDelete we built in Section 8.5 in this chapter and paste them into the Update button Click event procedure in our current project with some modifications. You can find the project SQLWebUpdateDelete in the folder DBProjects\Chapter 8 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

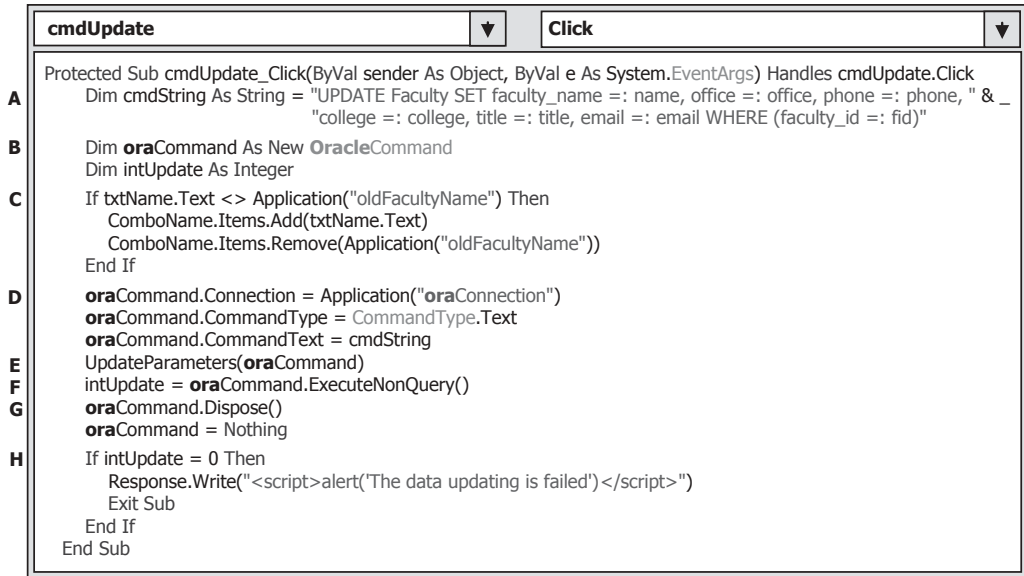


Figure 8.72. The modified Update button's event procedure.

Open the Update button's click event procedure in the Faculty page in the project SQLWebUpdateDelete, and copy all codes from that procedure, and paste them into the Update button click event procedure in the Faculty page in our current project. Perform the modifications shown in Figure 8.72 to this event procedure.

Let's take a closer look at this piece of modified codes to see how it works.

- A.** The updating query string has been modified by replacing the @ operator with the Oracle operator ::. Also, the comparison operator LIKE @ in the WHERE clause has been changed to =:, which is the Oracle comparison operator.
- B.** All data objects used in this procedure 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 executing the ExecuteNonQuery() method later. The prefix of all data objects and classes have been changed from sql to ora, and from Sql to Oracle. The modified codes have been highlighted in bold.
- C.** Now we need to check whether the user wants to update the faculty name or not. To do that, we need to compare the global variable oldFacultyName that is stored in the Application state with the current faculty name that is stored in the txtName textbox. If both names are different, this means that the user has updated the faculty name. In that case, we need to add the updated faculty name into the Faculty Name combo box control and remove the old faculty name from that control to allow users to select this updated faculty to perform the validation for this data updating later.
- D.** The Command object is initialized with the connection object, command type, and command text. Change the prefix of all data objects from sql to ora.

- E.** The user-defined subroutine `UpdateParameters()`, whose detailed codes are shown below, is called to assign all input parameters to the command object.
- F.** 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 data is stored to the local integer variable `intUpdate`.
- G.** A cleaning job is performed to release all data objects used in this procedure. Change the prefix of all data objects from `sql` to `ora`. Steps involved in this modification include **E**, **F**, and **G**.
- H.** 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 and the procedure is exited. Otherwise, if this value is nonzero, which means that this data updating is successful.

Now, let's develop the codes for the user-defined subroutine `UpdateParameters()`. You can copy this subroutine from the project `SQLWebUpdateDelete` we built in Section 8.5 and paste it into the code window of this page. Perform the modifications shown in Figure 8.73 to this subroutine after you paste this subroutine.

The function of this subroutine is straightforward. Seven input parameters are assigned to the `Parameters` collection property of the command object using the `Add()` method. Two modifications for this subroutine are:

- A.** Change the data type of the argument `cmd` for this subroutine from `SqlCommand` to `OracleCommand`.
- B.** Change the data type for all input parameters from `SqlDbType` to `OracleDbType`.

At this point, we have finished all coding developments for the data updating actions against 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 all faculty image files, including a default faculty image file `Default.jpg`, have been stored in our current project folder (`C:\Chapter 8\OracleWebUpdateDelete`).

Now you can start to run the project to test the data updating function in the Faculty page against the Faculty table in our sample Oracle database.

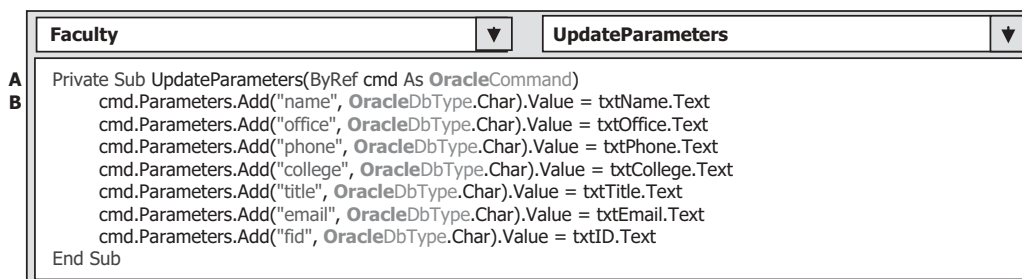


Figure 8.73. The modified subroutine `UpdateParameters()`.

8.9.2 Develop Stored Procedures to Perform the 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 for 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 delete 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. An easy way to perform these multiple deleting queries is to use the stored procedure to perform this data deleting.

8.9.2.1 Delete an Existing Record from the Faculty Table

Recall that in Section 7.8.2.3 in Chapter 7, we discussed how to develop a stored procedure in the Oracle database and call that stored procedure to perform the data deleting operation against the Oracle database. In this section, we still want to use our Faculty table as an example to discuss 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 and the Course tables. Two child tables are connected with the Faculty table by using the `faculty_id`, which is a primary key in the Faculty table and 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 that are 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. By calling and executing this stored procedure, we can easily complete this multi-table data deleting operation. The development sequence for this data deleting operation can be divided into the following three steps:

1. Develop the stored procedure in the Oracle database to perform the multi-table data deleting function.

2. Call the stored procedure from the ASP.NET Web application to perform the data deleting against the Oracle database.
3. Validate the data deleting action after the data deleting operation.

To save the time and the space, we will not provide a duplicated discussion about how to create a stored procedure in the Oracle database to perform this data deleting operation in this section, since we have discussed this topic in very detail in Section 7.8.2.3 in Chapter 7. Refer to that section to get more detailed materials about this issue. We will use the stored procedure **DeleteFacultySP**, which was developed in that section, to perform the data deleting action in this section.

In the following part, we assume that we have finished developing the stored procedure **DeleteFacultySP**, and we only take care of the coding process for steps 2 and 3.

8.9.2.2 Develop the Codes for the Delete Button's Event Procedure

Open the **Delete** button's click event procedure and enter the codes that are shown in Figure 8.74 into this event procedure.

Let's take a closer look at this piece of codes to see how it works.

- A. The content of the query string is now equal to the name of the stored procedure **DeleteFacultySP** that we created in the Oracle database in Section 7.8.2.3 in Chapter 7. 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 local variable used in this procedure are declared here. The integer variable **intDelete** is used to hold the returned value of executing the data updating method **ExecuteNonQuery()** of the command class later. Change the prefix of all data objects and classes from **sql** to **ora**, and from **Sql** to **Oracle**.
- 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.

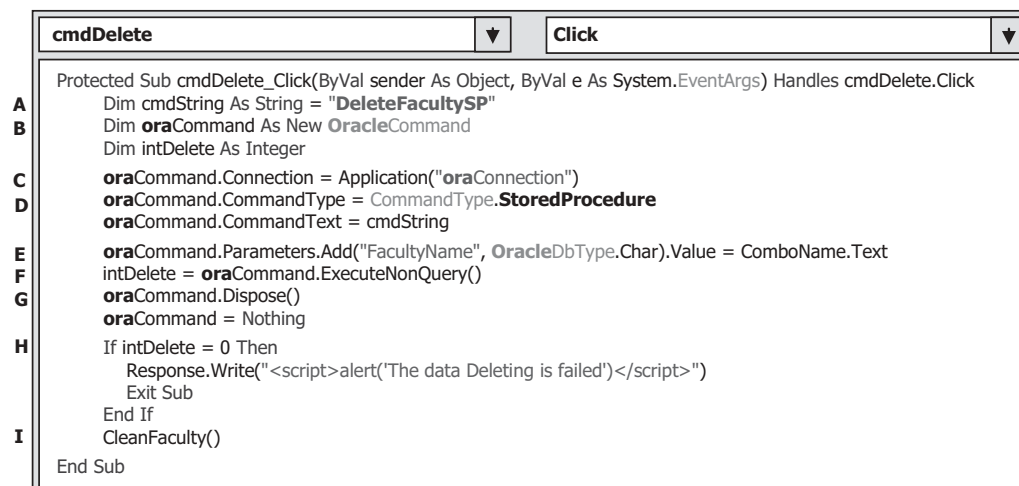


Figure 8.74. The codes for the Delete button's event procedure.

- D. The next object is the command type. The `CommandType.StoredProcedure` 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.
- E. The dynamic parameter is initialized with the real parameter faculty name that is stored in the Faculty Name combo box control. One point to be noted is that you must use the faculty name stored in this combo box control, not the faculty name stored in the faculty name textbox control, for this dynamic parameter since the latter may be an updated faculty name but not an original faculty name.
- F. 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.
- G. A cleaning job is performed to release all objects used in this event procedure.
- H. The integer value returned from the calling of `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 and the procedure is exited. Otherwise, if this value is nonzero, which means that at least one row has been deleted from the database, and this data deleting is successful.
- I. A user-defined subroutine `CleanFaculty()`, whose detailed codes are shown below, is called to clean up all faculty information stored in seven textboxes.

The codes for the subroutine `CleanFaculty()` are shown in Figure 8.75.

The function of this piece of codes is straightforward and easy to be understood. An `Empty` property of the `String` class is assigned to all textboxes to make them empty and to clean them up.

Now we have finished all coding developments for the data deleting action against 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, and all faculty image files, including a default faculty photo file `Default.jpg`, have been stored in our current project folder (`C:\Chapter 8\OracleWebUpdateDelete`).

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 keep the default

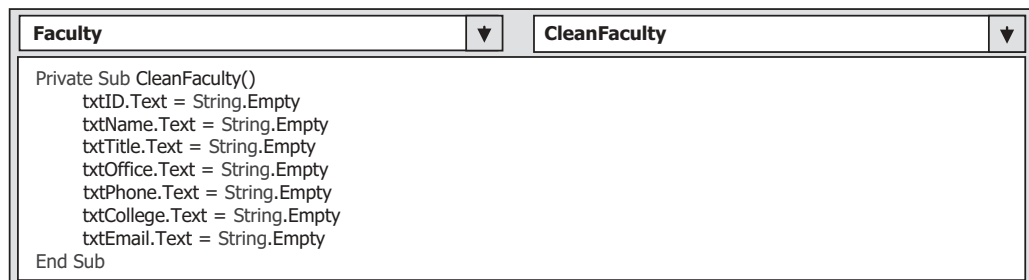


Figure 8.75. The codes for the subroutine `CleanFaculty()`.

faculty name Ying Bai to be selected from the Faculty Name combo box control and click on the **Select** button to retrieve and display this faculty record in the Faculty page.

To test the data deleting action against the Faculty table in our sample database, 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 contains the selected faculty information are cleaned up. Does that mean our data deleting is successful? Let's perform the following steps to confirm it.

8.9.2.3 Validate the Data Deleting Actions

To confirm this data deleting, there are two ways to do that. The first way is to try to retrieve this deleted faculty record from the database. The data deleting action would be successful if no such faculty record can be found and retrieved from the database. The second way is to open the database to check the associated tables to confirm this data deleting.

First, let's do this confirmation using the first way. Still in the Faculty page, keep the faculty name Ying Bai selected in the Faculty Name combo box control and click on the **Select** button to retrieve this faculty record from the database and display it in the Faculty page. A warning message "No matched faculty found!" is displayed, which means that the selected faculty record has been successfully deleted from the database.

Next, let's perform the following operations to open the Oracle database to check the associated tables to confirm this data deleting.

1. Open the Oracle Database 11g XE home page by going to the **start|All Programs| Oracle Database 11g Express Edition|Get Started** items.
2. On the opened starting page, click on the **APEX** button to open the APEX login wizard. Enter the username and password and then click on the **Login** button to open the APEX Workshop wizard.
3. Click on the **Already have an account? Login Here** button to open the Workshop login wizard. Keep the default workshop name **CSE_DEPT** unchanged, and enter the password **reback** to complete this login process.
4. Click on the **SQL Workshop** and then the **Object Browser** icon, and keep the default item **Table** selected.
5. On the opened Table page, double-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 8.76.

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. The Faculty table has some relationships with all other 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 double-clicking on each of them one by one from the left pane; you can find that those records related to the faculty member Ying Bai have been deleted from the LogIn and the Course tables. The relationship

The screenshot shows a web browser window with the title 'Object Browser - Windows Internet Explorer'. The address bar shows a URL starting with 'http://127.0.0.1:8080/asp/...'. The browser displays a table named 'FACULTY'. The table has columns: FACULTY_ID, FACULTY_NAME, OFFICE, PHONE, COLLEGE, TITLE, and EMAIL. The table contains 8 rows of data. The browser interface includes a 'Tables' list on the left and a 'Query' tab at the top of the table view.

FACULTY_ID	FACULTY_NAME	OFFICE	PHONE	COLLEGE	TITLE	EMAIL
M56789	Ali Mhamed	MTC-353	750-378-3355	University of Man	Professor	amhamed@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
K99880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu
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
B66590	Satish Bhalla	MTC-214	750-378-1051	University of Notre Dame	Associate Professor	sbhalla@college.edu

Figure 8.76. The Faculty table after the data deleting.

between the Faculty and the LogIn tables, as well as between the Faculty and the Course tables, 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 confirmed that our data deleting is successful.

But the story is not finished. As you know, the Faculty table has some relationships with all 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, and you can find that the students who are identified by the `student_id` and took these four courses have also been deleted from the StudentCourse table! Why those records are deleted and who did that? To solve this problem and find the answer to this question, we need to review our sample Oracle database building process. 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 have a closer look at those staffs to try to solve our problem in the next section.

8.9.2.4 The Constraint Property: On Delete Cascade in the Data Table

Recall that in Section 2.11.6 in Chapter 2, we used the constraint property to set up the foreign key and create the relationship 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 tables, we selected the `course_id` as the primary key for

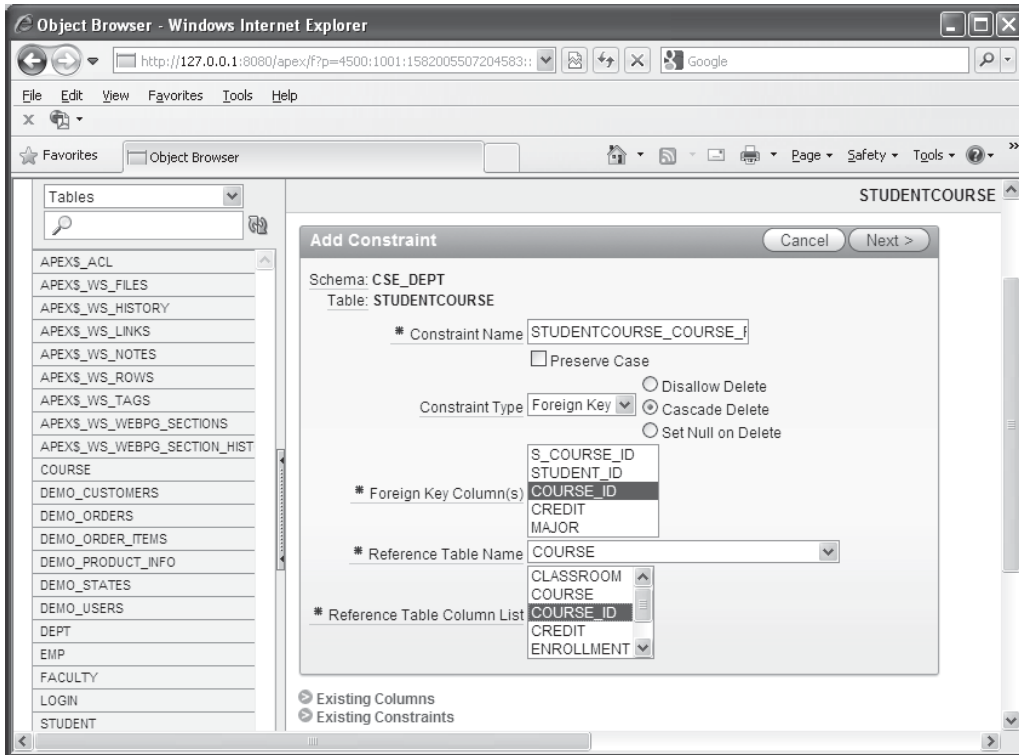


Figure 8.77. Create the foreign key between the StudentCourse and the Course table.

the Course table and 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 radio button named **Cascade Delete**, which is located at the right of the **Constraint Type** textbox. We checked this radio button when we created this foreign key for the StudentCourse table. To make this issue clear and provide readers with a global picture, we redisplay Figure 2.67, which is Figure 8.77 in this section.

It can be found that the **Cascade Delete** radio button has been 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 so-called cascaded deleting mode. The word *cascade* means series, and cascaded deleting means that if the 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 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 **Cascade Delete** radio button 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 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` that 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 **Cascade Delete** radio button was checked when the relationship is set up between tables. Similar things happened to the `student_id`, which is also a foreign key in the `StudentCourse` table.

In fact, to perform this faculty member deleting action, we do not need to build any stored procedure to include all of those three queries. We can only delete the desired faculty member from the `Faculty` table, and the Oracle database engine can perform these cascaded deleting actions to delete all other related records from all other four tables.

Before we can close the Oracle database 11g XE, it is highly recommended to recover all deleted records to the associated tables. Refer to Tables 8.12–8.15 to add those records back to the associated tables. You use the **Insert Row** button in the Object Browser to add these records into the associated tables. Now you can close the Oracle Database 11g XE.

Table 8.12. The 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 8.13. The data to be added into the `LogIn` table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	

Table 8.14. The 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 8.15. The 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

A complete Web application project `OracleWebUpdateDelete` can be found in the folder `DBProjects\Chapter 8` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

8.10 CHAPTER SUMMARY

A detailed and completed introduction to the ASP.NET and the .NET Framework is provided at the beginning of this chapter. This part is especially useful and important to readers or students who do not have any knowledge or background in the Web application project 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 from Section 8.3, the detailed development and building process 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. Seven real ASP.NET Web application projects include:

1. Develop an ASP.NET Web application to select and display data from the Microsoft SQL Server 2008 database.
2. Develop an ASP.NET Web application to insert data into the Microsoft SQL Server 2008 database.
3. Develop an ASP.NET Web application to update and delete data against the Microsoft SQL Server 2008 database.
4. Develop an ASP.NET Web application project to access and manipulate data against SQL Server 2008 database using LINQ to SQL query method.
5. Develop an ASP.NET Web application to select and display data from the Oracle 11g XE database.
6. Develop an ASP.NET Web application to insert data into the Oracle 11g XE database.
7. Develop an ASP.NET Web application to update and delete data against the Oracle 11g XE database.

The stored procedures are utilized in two projects, projects 3 and 7, to help readers or students to perform the data updating and deleting actions against two kinds of popular 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 the readers and improve their knowledge and hands-on experience on these issues.

HOMework

I. True/False Selections

- ____ 1. The actual language used in the communications 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 is a programming framework built on the .NET Framework, 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 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 List Box control to respond to a user click as the Web page runs, the AutoPostBack property of that List Box 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. Defaultty.aspx file
 - c. Config file
 - d. Application state
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 procedure, a Web-based application uses the _____ as the first event procedure.
- 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. Set default
 - d. Set Null

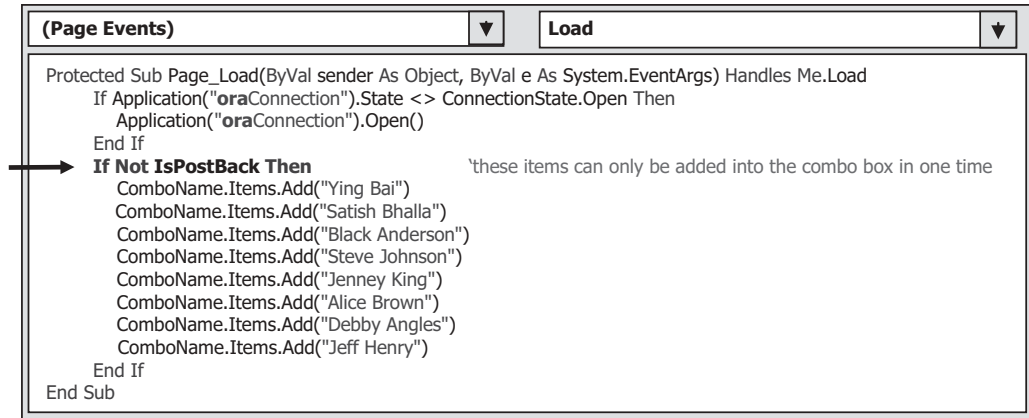


Figure 8.78. The codes for the Page_Load() event procedure.

10. To display any message on a running Web page, one must use the _____ method.
- MessageBox.Show()
 - Response()
 - Response.Redirect()
 - Response.Write()

III. Exercises

- Write a paragraph to answer and explain the following questions:
 - What is ASP.NET?
 - What is the main component of the ASP.NET Web application?
 - How is an ASP.NET Web application executed?
- 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 the data deleting from the child and the parent tables.
- Figure 8.78 shows a piece of codes developed in the Page_Load() event procedure. Explain the function of the statement **If Not IsPostBack Then** block.
- Add a Web page and develop the codes to perform the data deleting for the Student page in the SQLWebUpdateDelete project. The project file can be found in the folder DBProjects\Chapter 8 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).
- Use the Cascade options for relationships 1, 2, and 3 listed in Section 8.5.3.2 in this chapter to create only one deleting query to delete a faculty member from the Faculty table in our sample database (refer to Section 8.5.3.2 to get a detailed discussion for this issue).
- Develop a Web page Student.aspx and create a stored procedure to delete one record from the Student table by using the project OracleWebUpdateDelete. The project file can be found in the folder DBProjects\Chapter 8 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). It is highly recommended to recover those deleted records after they are deleted.

Hints: You need to delete the related records from the LogIn and StudentCourse tables, and then delete record from the Student table.

Chapter 9

ASP.NET Web Services

We provided a very detailed discussion about the ASP.NET Web applications in the last chapter. In this chapter, we will concentrate on another ASP.NET-related topic—the 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 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 the users need to create those GUIs themselves to access the Web services via the Internet.

When finished 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 the 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 the professional ASP.NET Web Service projects to insert new records into our sample database.
- Build the professional ASP.NET Web Service projects to update and delete data against our sample database.

In order to help readers to successfully complete this chapter, first, we need to provide a detailed discussion about the ASP.NET Web Services and their components.

9.1 WHAT ARE WEB SERVICES AND THEIR COMPONENTS?

Essentially, the 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 through the Internet. Those 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
3. UDDI
4. WSDL

The function of each component is listed below.

XML is a text-based data storage language, and it uses a series of tags to define and store data. 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. But the 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 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 run at any operating systems with any programming languages. Exactly, a 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, this request is first wrapped into a SOAP message and sent over the Internet to the Web server. Similarly, 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 for how a Web service method can be formatted in a SOAP message: Remote Procedure Call (RPC) and Document. Both RPC and Document style message 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

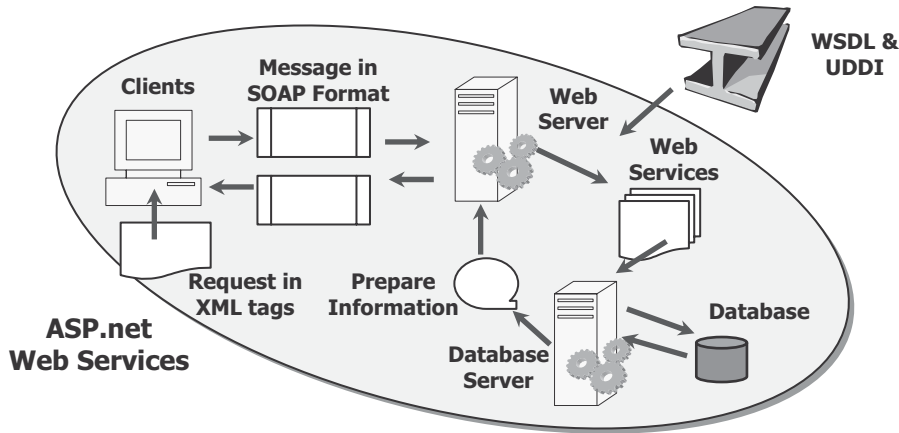


Figure 9.1. A typical process of a Web service.

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 to list themselves on the Internet, and 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 as 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.

Summarily, based on these components and their roles discussed above, we can conclude:

- The XML is used to tag the 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 to 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.

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 Basic.NET 2010. 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 The 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.vb**. This page contains the real Visual Basic.NET codes for a simple Web service. Visual Web Developer includes three default declarations to help users to develop Web services on the top of this page, which are:

```
Imports System.Web  
Imports System.Web.Services  
Imports 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**. This page 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 day. If you open the **ASMX** file on disk, you will see that it actually contains only one command line:

```
<@WebService Language = "vb" CodeBehind = "~/App_Code/Service.vb" 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 **ASMX** page through **IIS**, **ASP.NET** uses this information to generate the content displayed in the Web browser.

4. The configuration file **Web.config**, which is XML-based file, is used to set up a configuration for the newly 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 files and folders discussed above, the code-behind page is the most important file, since all Visual Basic.NET codes related to build a Web service are located in this page, and our major coding development will be concentrated on this page, too.

9.2.2 The 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 Internet Information Services (IIS) on the client computer, which works as a pseudo server. Then, the IIS will pass the request to 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, exactly 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, ASP.NET will make sure that the names and identifiers used in the SOAP message must be 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 a separate Web method to query each piece of information, and that will make our project's size terribly large if we need a lot of information. A good solution is to instantiate an object based on that class and integrate those 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, and it 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 that 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 did 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.

Starting from .NET Frameworks 4.0, a good platform, Windows Communication Foundation (WCF), is provided as support to build professional Web Services projects. First, let's have a basic understanding about this new tool.

9.2.3 Introduction to Windows Communication Foundation (WCF)

As the development of the service-oriented communications advanced, the software development has been significantly changed. Whether the message is done with SOAP or in some other ways, applications that interact through services have become the norm. For Windows developers, this change is made possible by using the WCF. First released as part of .NET Framework 3.0 in 2006, then updated in .NET Framework 3.5, the most recent version of this technology is included in the .NET Framework 4. For a substantial share of new software built on .NET, WCF is the right foundation.

9.2.3.1 *What Is WCF?*

WCF is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An end point can be a client of a service that requests data from a service end point.

WCF is a unified framework for creating secure, reliable, transacted, and interoperable distributed applications. In earlier versions of Visual Studio, there were several technologies that could be used for communicating between applications.

If you wanted to share information in a way that enabled it to be accessed from any platform, you would use a Web service (also known as an ASMX Web service). If you wanted to just move data between a client and server that are running on the Windows operating system, you would use .NET Remoting. If you wanted transacted communications, you would use Enterprise Services (DCOM), or if you wanted a queued model, you would use Message Queuing (also known as MSMQ).

WCF brings together the functionality of all those technologies under a unified programming model. This simplifies the experience of developing distributed applications.

In fact, WCF is implemented primarily as a set of classes on the top of the .NET Framework's Common Language Runtime (CLR). This allows .NET developers to build service-oriented applications in an easy way. Also, WCF allows creating clients that access services in a mutual way, which means that both the client and the service can run in pretty much the same way as any Windows process did. WCF doesn't define a required host. Wherever they run, clients and services can interact via SOAP, via a WCF-specific binary protocol, and in other ways.

9.2.3.2 *WCF Data Services*

WCF Data Services, formerly known as ADO.NET Data Services, is a component of the .NET Framework that enables you to create services that use the Open Data Protocol (OData) to expose and consume data over the Web or Intranet by using the semantics of representational state transfer (REST). Odata exposes data as resources that are addressable by URIs. Data is accessed and changed by using standard HTTP verbs of GET, PUT, POST, and DELETE. OData uses the entity-relationship conventions of the Entity Data Model to expose resources as sets of entities that are related by associations.

WCF Data Services uses the OData protocol for addressing and updating resources. In this way, you can access these services from any client that supports OData. OData enables you to request and write data to resources by using well-known transfer formats: Atom, a set of standards for exchanging and updating data as XML, and JavaScript Object Notation (JSON), a text-based data exchange format used extensively in AJAX application.

WCF Data Services can expose data that originates from various sources as OData feeds. Visual Studio tools make it easier for you to create an OData-based service by using an ADO.NET Entity Framework data model. You can also create OData feeds based on CLR classes and even late-bound or untyped data.

```

<ServiceContract(>
Public Interface IService1
<OperationContract(>
Function GetData(ByVal value As String) As String

```

Figure 9.2. The Service interface and contract.

WCF Data Services also includes a set of client libraries, one for general .NET Framework client applications and another specifically for Silverlight-based applications. These client libraries provide an object-based programming model when you access an OData feed from environments, such as the .NET Framework and Silverlight.

9.2.3.3 WCF Services

A WCF service is based on an interface that defines a contract between the service and the client. It is marked with a **ServiceContractAttribute** attribute, as shown in the codes in Figure 9.2.

You define functions or methods that are exposed by a WCF service by marking them with an **OperationContractAttribute** attribute. In addition, you can expose serialized data by marking a composite type with a **DataContractAttribute** attribute. This enables data binding in a client.

After an interface and its methods are defined, they are encapsulated in a class that implements the interface. A single WCF service class can implement multiple service contracts.

A WCF service is exposed for consumption through what is known as an **endpoint**. The endpoint provides the only way to communicate with the service; you cannot access the service through a direct reference as you would with other classes.

An endpoint consists of an address, a binding, and a contract. The address defines where the service is located—this could be a URL, an FTP address, or a network or local path. A binding defines the way that you communicate with the service. WCF bindings provide a versatile model for specifying a protocol, such as HTTP or FTP, a security mechanism, such as Windows Authentication or user names and passwords, and much more. A contract includes the operations that are exposed by the WCF service class.

Multiple endpoints can be exposed for a single WCF service. This enables different clients to communicate with the same service in different ways. For example, a banking service might provide one endpoint for employees and another for external customers, each using a different address, binding, and/or contract.

9.2.3.4 WCF Clients

A WCF client consists of a *proxy* that enables an application to communicate with a WCF service, and an endpoint that matches an endpoint defined for the service. The proxy is generated on the client side in the **app.config** file and includes information about the types and methods that are exposed by the service. For services that expose multiple endpoints, the client can select the one that best fits its needs, for example, to communicate over HTTP and use Windows Authentication.

```

Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles Button1.Click

    Dim client As New ServiceReference1.Service1Client
    Dim returnString As String

    returnString = client.GetData(TextBox1.Text)
    Label1.Text = returnString

End Sub

```

Figure 9.3. The codes in the client side to call the operation `GetData()` in the server.

After a WCF client has been created, you reference the service in your code just as what you could do for any other object. For example, to call the `GetData()` method shown in Figure 9.2, you would write the codes shown in Figure 9.3.

In most cases, you need to create a proxy to set up a reference to the server in the client to access the operations defined in the server.

9.2.3.5 WCF Hosting

From a developer perspective, WCF provides two alternatives for hosting services, which are both mostly identical under the covers. The easier of the two alternatives is to host services inside an ASP.NET application, the more flexible and more explicit alternative is to host services yourself and in whichever application process you choose.

Hosting WCF services in ASP.NET is very simple and straightforward and very similar to the ASMX model. You can either place your entire service implementation in a `*.svc` file just as with ASP.NET Web services `*.asmx` files, or you can reference a service implementation residing in a code-behind file or some other assembly. With respect to how the service implementation class is located (and possibly compiled), none of these options differ much from how you would typically create an ASMX Web service, even the attributes of the `@Service` directive are the same as those for the `@WebService` directive.

The important difference between WCF and ASMX is that the WCF service will not do anything until you specify precisely how it shall be exposed to the outside world. An ASMX service will happily start talking to the world once you place the `*.asmx` file into an IIS virtual directory. A WCF service will not talk to anybody until you tell it to do so and how to do so.

9.2.3.6 WCF Visual Studio Templates

Visual Studio.NET provides a set of WCF templates to help developers build different Web services and applications. In fact, WCF Visual Studio templates are predefined project and item templates you can use in Visual Studio to quickly build WCF services and surrounding applications.

WCF Visual Studio templates provide a basic class structure for service development. Specifically, these templates provide the basic definitions for service contract, data contract, service implementation, and configuration. You can use these templates to create a

simple service with minimal code interaction, as well as a building block for more advanced services.

Two popular templates are **WCF Service Application** template and **WCF Service Library** template. Both are located under the **New Project\Visual Basic\WCF** command folder.

9.2.3.6.1 WCF Web Service Application Template When you create a new Visual Basic.NET project using the **WCF Web Service Application** template, the project includes the following four files:

1. Service host file (**Service.svc**). The service host file indicates the general properties of this service, including the language used, service name, and the name of the code-behind file.
2. Service contract file (**IService.vb**). The service contract file is an interface that has WCF service attributes applied. This file provides a definition of a simple service to show you how to define your services, and includes parameter-based operations and a simple data contract sample. This is the default file displayed in the code editor after creating a WCF service project.
3. Service implementation file (**Service.vb**). The service implementation file implements the contract defined in the service contract file.
4. Web configuration file (**Web.config**). The configuration file provides the basic elements of a WCF service model with a secure HTTP binding. It also includes an endpoint for the service and enables metadata exchange.

The template automatically creates a Website that will be deployed to a virtual directory and hosts a service in it.

9.2.3.6.2 WCF Service Library Project Template When you create a new Visual Basic.NET project using the **WCF Service Library** template, the new project automatically includes the following three files:

1. Service contract file (**IService.vb**).
2. Service implementation file (**Service.vb**).
3. Application configuration file (**App.config**).

Now, let's start to build our Web service project using the WCF template. We prefer to use the **WCF Web Service Application** template and include our Web service in our ASP.NET application project.

9.2.4 Procedures to Build an ASP.NET Web Service

The advantages of using the WCF templates to build our Web services are obvious: for instance, the protocols of the interface and contract have been predefined. However, you must follow up those protocols to fill your codes, such as operations and methods. An easy way to do these is to directly add our Web service with our operations in our ways. In the following sections, we will not use the protocols provided by WCF and directly create our Web services and place them into an ASP.NET Web services *.asmx file.

Web service is basically composed of a set of Web methods that can be called by the computer programs in the client side. To build those methods, generally one needs to perform the following steps:

1. Create a new WCF Web Service project.
2. Add a new ASP.NET Web Service project.
3. Create a base class to handle the error checking to protect our real class.
4. Create our real Web service class to hold all Web methods and codes to response to requests.
5. Add all Web methods into our Web service class.
6. Develop the detail codes for those Web methods to perform the Web services.
7. Build a Windows-based or Web-based project to consume the Web service to pick up and display the required information on the GUI.
8. 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 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 records into the SQL Server database.
- Build a professional ASP.NET Web Service project to update and delete data 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 records into the Oracle database.
- Build a professional ASP.NET Web Service project to update and delete data against the Oracle database

Based on procedures discussed above, we can start to build our first Web service project `WebServiceSQLSelect`.

9.3 BUILD ASP.NET WEB SERVICE PROJECT TO ACCESS SQL SERVER DATABASE

To create a new ASP.NET Web Service project, perform the following operations:

1. Open the Windows Explorer to create a new folder **Chapter 9** under your root drive C.
2. Open the Visual Studio.NET 2010 and go to **File|New Web Site** item.
3. On the opened New Web Site wizard, make sure that the **Visual Basic** is selected under the **Recent Templates**, and select the **WCF Service** item from the Templates list. Enter `C:\Chapter 9\WebServiceSQLSelect` into the box that is next to the **Web Location** box, which is shown in Figure 9.4.

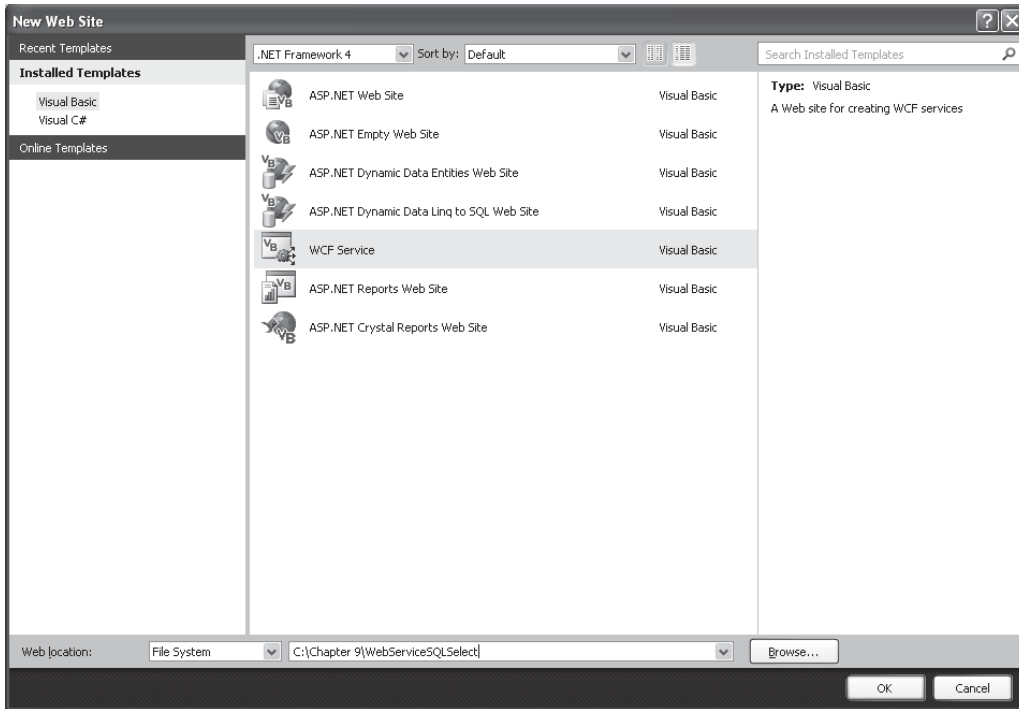


Figure 9.4. Create a new WCF Web Service project.

One point to be noted is that Visual Studio.NET 2010 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 on a local 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 9.4.

Click on the OK button to create this new WCF Web service project in our default folder C:\Chapter 9.

9.3.1 Files and Items Created in the New Web Service Project

After this new WCF Web service project is created, four items are produced in the Solution Explorer window, which are shown in Figure 9.5.

1. Service host file (**Service.svc**).
2. Service contract file (**IService.vb**).
3. Service implementation file (**Service.vb**).
4. Web configuration file (**Web.config**).

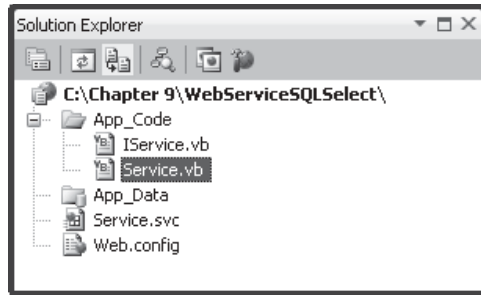


Figure 9.5. Newly created items for a WCF Web service project.

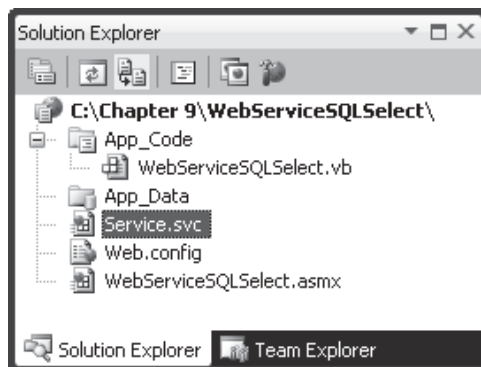


Figure 9.6. The modified Web service project.

Since we want to build our Web service in our customer way, we do not need to use files 2 and 3. Therefore, right click on these two files and select the **Delete** item to remove both of them from our project.

Perform the following operations to add a new Web service into our project:

1. Right-click on our new project **WebServiceSQLSelect** from the Solution Explorer window and select **Add New Item**.
2. On the opened **Add New Item** wizard, select the **Web Service** from the **Template** list.
3. Enter **WebServiceSQLSelect.asmx** into the **Name** box.
4. Click on the **Add** button to complete this item addition operation.

The modified Web service project is shown in Figure 9.6.

Two folders, **App_Code** and **App_Data**, are also created in this new project. The former is used to store our code-behind page **WebServiceSQLSelect.vb**, and the latter is used to save the project data. The code-behind page **WebServiceSQLSelect.vb** 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 a default Web method **HelloWorld** that is a placeholder, and we can replace it with our own method or methods later on based on the requirement of our Web service project.

The main service page file **WebServiceSQLSelect.asmx** is used to display information about the Web service's methods and provide access to the Web service's WSDL information. The configuration file **Web.config** 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 data components and Web services, and Windows authentication mode. All of these components are automatically created and added into our new project. More important, the page file **WebServiceSQLSelect.asmx** is designed to automatically create extensible WSDL, dispatch Web methods, serialize and deserialize parameters, and provide hooks for message interception within our applications. But now the default file **WebServiceSQLSelect.asmx** only contains a compile directive when a new Web service project is created and opened from the File System.

Now let's modify the Service host file **Service.svc** to make it matched to our Web service.

Double-click on this file from the Solution Explorer window to open this file. Perform the following modifications to this file:

1. Change the Service's name to **WebServiceSQLSelect**.
2. Change the name of the code-behind file to **WebServiceSQLSelect.vb**.

Your modified **Service.svc** file should match one that is shown in Figure 9.7.

Now double-click on the code-behind page **WebServiceSQLSelect.vb** to open this file that is shown in Figure 9.8, and let's have a closer look at the codes in this page.

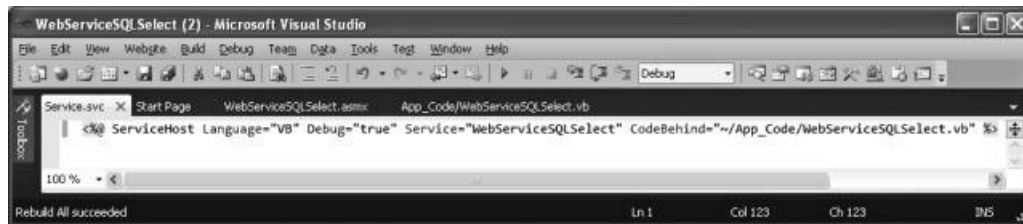


Figure 9.7. The Modified Service host file **Service.svc**.

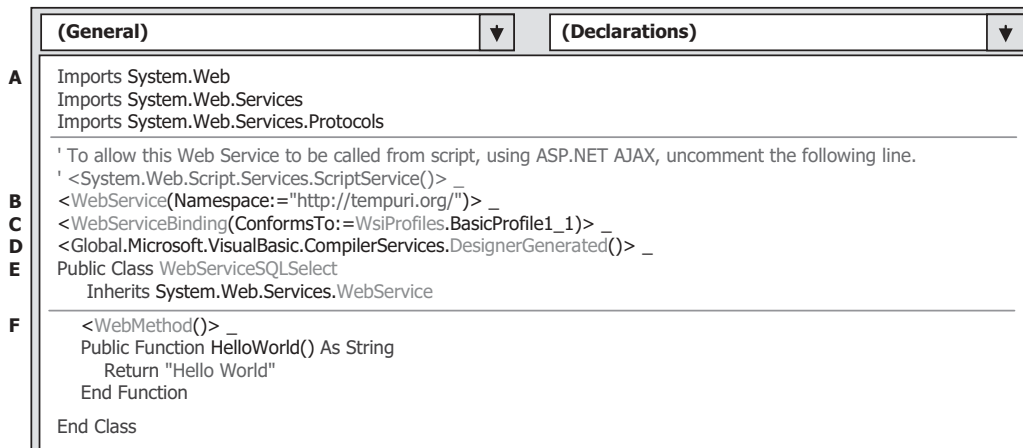


Figure 9.8. The default codes for the code-behind page **WebServiceSQLSelect.vb**.

Table 9.1. The 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.Protocols	Define the protocol used to transmit data across the wire during the communication between the Web Service clients and servers

- A. The Web services-related namespaces that contains the Web service components are imported first to allow us to access and use those components to build our Web service project. A detailed description about those namespaces and their functionalities 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 the 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 with any other SOAP messages used by other Web services. Here, in this new project, Microsoft used a default namespace **http://tempuri.org/**, which is a temporary system-defined namespace to identify all Web Services code generated 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, exactly a binding is equivalent to an interface in which it defines a set of concrete operations.
- D. This attribute indicates that the default class Service is created during the designing time by the designer.
- E. Our Web service class **WebServiceSQLSelect** is a child class that is derived from the parent class **WebService** located in the namespace **System.Web.Services**.
- F. 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, double-click on the main service page file **WebServiceSQLSelect.aspx** that is the entry point of our project to open it. Only one code line that contains a compile directive shown below is displayed since this project is created and opened using a File System.

```
<%@ WebService Language = "VB" CodeBehind = "~/App_Code/WebServiceSQLSelect.vb"
    Class = "WebServiceSQLSelect" %>
```

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. Now let’s run the default **HelloWorld** Web service project to get a feeling about what it looks like and how it works.

Click on the Start Debugging button to run the default **HelloWorld** project.

9.3.2 A Feeling of the Hello World Web Service Project As it Runs

After the project running, a message box is displayed with the following warning message displayed, which is shown in Figure 9.9.

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. But the debugging can be enabled during the development process by modifying the `Web.config` file. To do that, keep the default radio button selected and click on the OK button in this message box to continue to run our project. Our `WebServiceSQLSelect.aspx` page should be the starting page, and the following IE page is displayed as shown in Figure 9.10.

This page displays the Web service class name `WebServiceSQLSelect` and all Web methods or operations developed in this project. By default, only one method `HelloWorld` is created and used in this project.

Below the method, the default namespace in which the current method or operation is located is shown up, and a recommendation that suggests us to 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 is shown up, which is shown in Figure 9.11.

The `Invoke` button is used to test our `HelloWorld` method using the HTTP Protocol. Below the `Invoke` button, some message examples that are created by using 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 that is 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, a URL that contains the default namespace and the default `HelloWorld` method's name is activated, and a new browser window that is shown in Figure 9.12 is displayed. When the default method `HelloWorld` is executed, the main

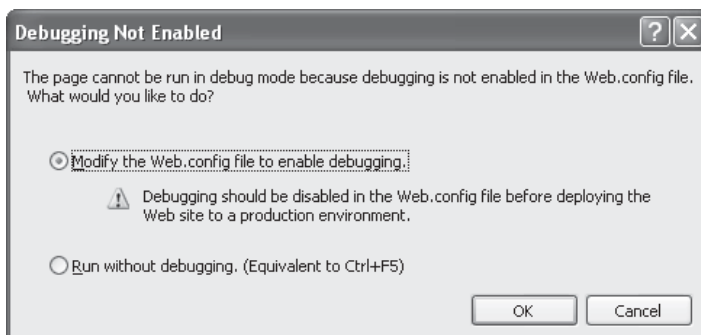


Figure 9.9. The Debugging Not Found message box.

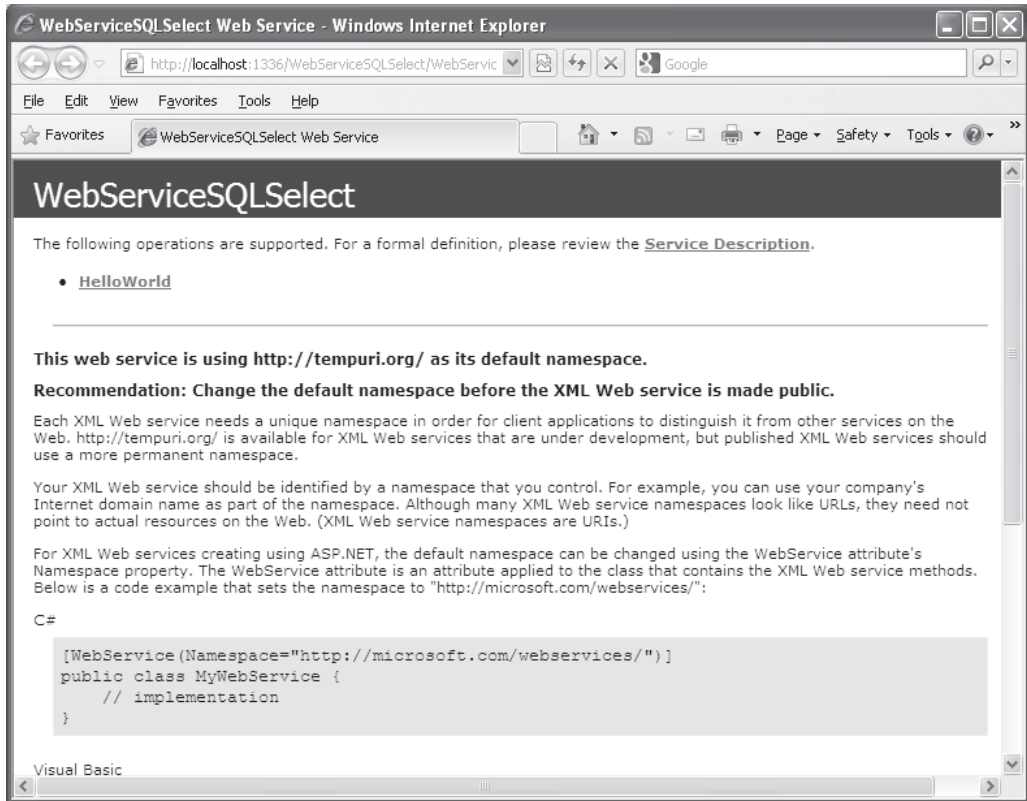


Figure 9.10. The running status of the default Web service project.

service page `WebServiceSQLSelect.asmx` sends a request to the IIS, and 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 returned 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.

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 testing purposes for the default Web service. In a real application, no such Web interface will be provided and displayed.

Click on the Close button that is located on the upper-right corner of the browser to close two browser pages.

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 build our own Web service project by developing the codes to perform the request to our sample database, that is, to the Faculty table, to get the desired faculty information.

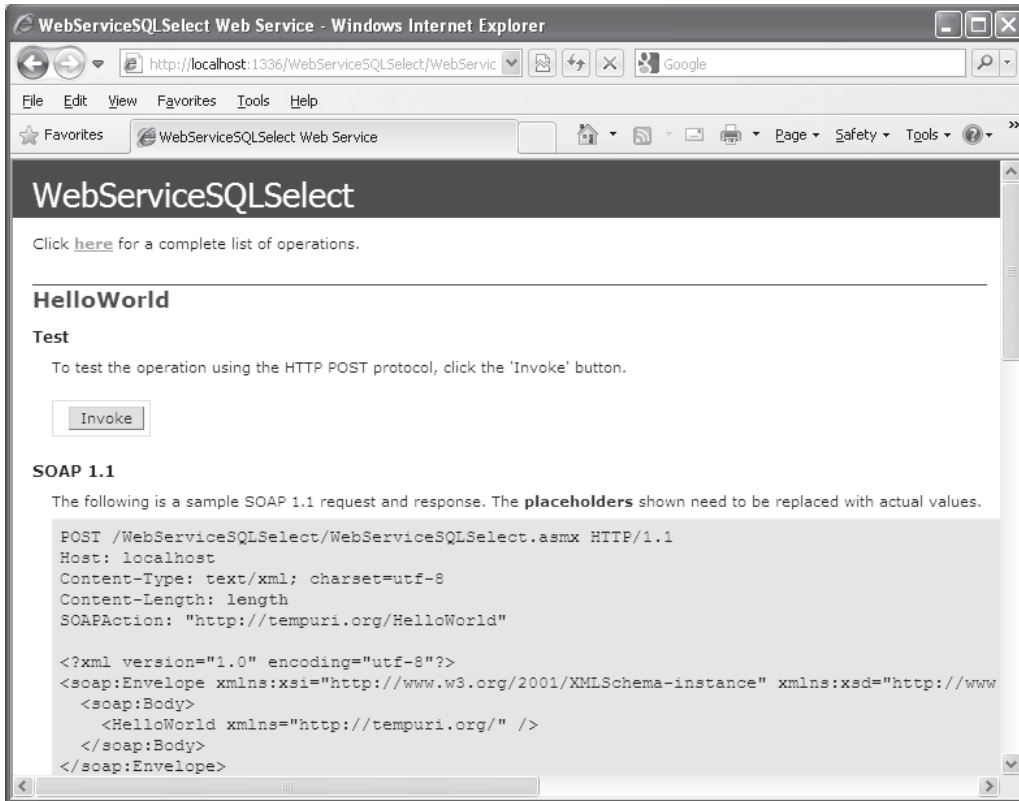


Figure 9.11. The test method page.

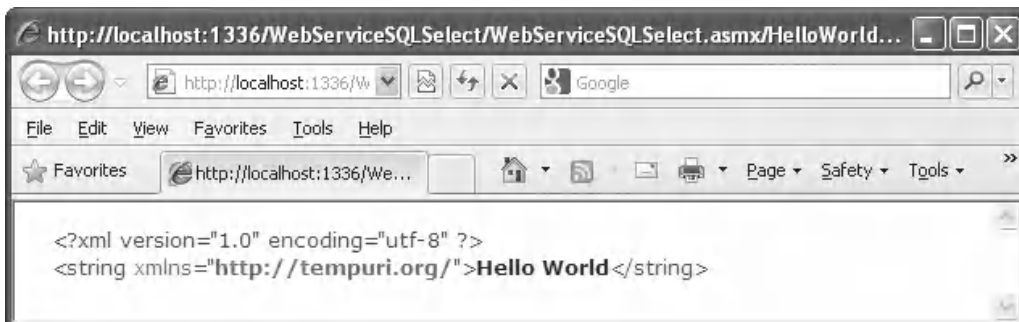


Figure 9.12. The running status of the default method.

We will develop our Web service project in the following sequence:

1. Modify the default namespace to create our own Web service namespace.
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 response 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 IIS.

Let's start from the step 1.

9.3.3 Modify the Default Namespace

We will modify the default Web service namespace to create our own namespace to store our Web service project.

Open the code-behind page `WebServiceSQLSelect.vb` by double clicking-on it from the Solution Explorer window, and perform the modifications that are shown in Figure 9.13 to this page.

Let's have a closer look at this piece of modified codes 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 the Wiley appended with the 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.

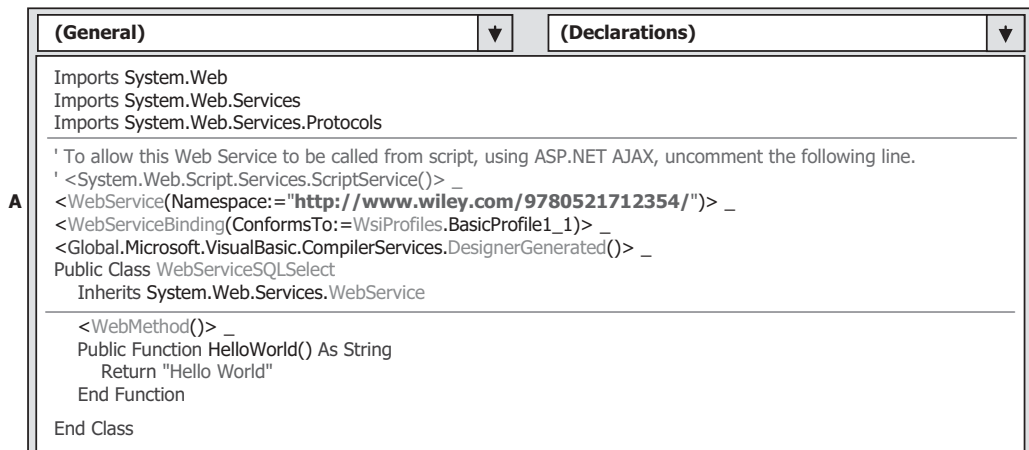


Figure 9.13. The modified code-behind page.

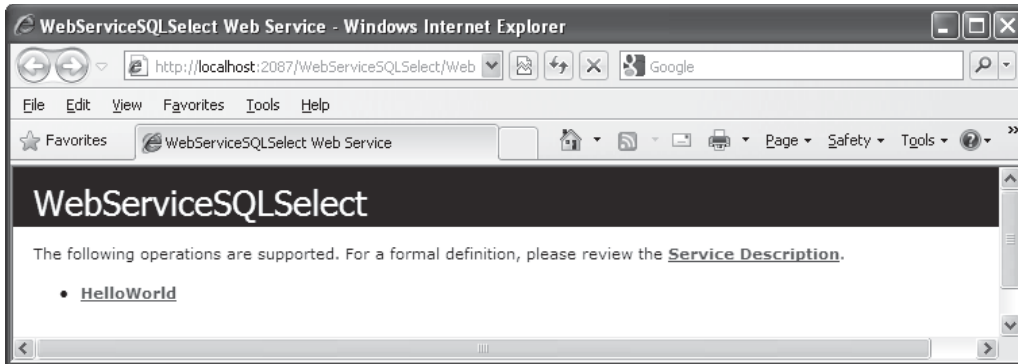


Figure 9.14. The running status of the modified Web service project.

Double-click on our main service file `WebServiceSQLSelect.asmx` from the Solution Explorer window to open it. Now click on the Start Debugging button to run our new Web service project; a default Web interface is displayed with our project name, as shown in Figure 9.14.

If you click on the default method `HelloWorld` and then Invoke button to test that method, you can find that the namespace has been updated to our new specific namespace, `http://www.wiley.com/9780521712354/`.

A point is that you must set the service file `WebServiceSQLSelect.asmx` as the start page before you can run our service project since this file is the entry point of our Web service project.

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 for some reasons 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 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.

To create a new base class in our new project, right-click on our new service project `WebServiceSQLSelect` from the Solution Explorer window. Then select the Add New Item from the pop-up menu. On the opened Add New Item wizard, select the Class item from the Template list, and enter `SQLSelectBase.vb` into the Name box as our new class name. Then click on the Add button to add this new class into our project.

Click Yes to the message box to place this new class into the `App_Code` folder in our new Web service project.

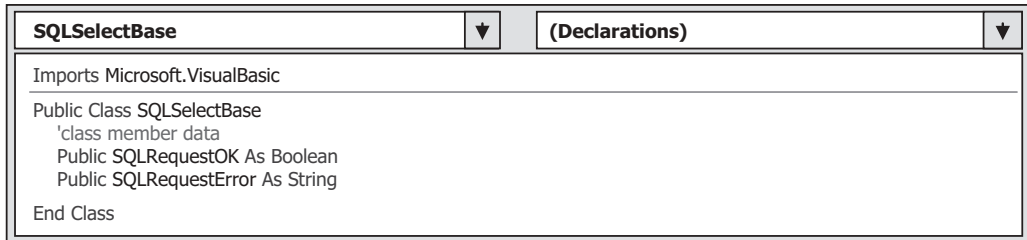


Figure 9.15. The class member data.

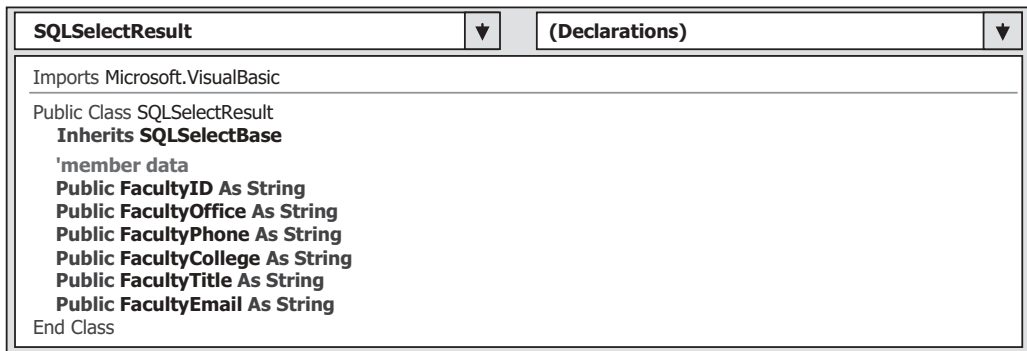


Figure 9.16. The member data for the class SQLSelectResult.

Now double-click on this newly added class and enter the codes that are shown in Figure 9.15 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 the Real Web Service Class

Now we need to create our real Web service class that 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 `WebServiceSQLSelect` from the Solution Explorer window, and select the **Add New Item** from the pop-up menu. On the opened **Add New Item** wizard, select the **Class** item from the **Template** list and then enter the `SQLSelectResult.vb` 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 **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 newly added class and enter the codes that are shown in Figure 9.16 into this class as the member data to this class.

Since this class will be instantiated to an object that will be returned with our desired faculty information to us as the Web method is called, so 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 data should be included and returned:

- Faculty_id
- Faculty office
- Faculty phone
- Faculty college
- Faculty title
- Faculty email

All of these pieces of information, which can be exactly mapped to all columns in the Faculty table in our sample database, are needed to be added into this class as the member data. This does not look like a professional schema, yes, that is true. Another better option is that we do not need to create any class that will be instantiated to an object to hold these pieces of information, instead we can use a DataSet to hold those pieces of information and allow the Web method to return that DataSet as a whole package for those pieces of faculty information. But that better option is relatively complicated compared with our current class. So right now we prefer to start our project with an easier way, and later on we will discuss how to use the DataSet to return our desired information in the following sections.

Let's take a look at these added member data for this class.

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. Six pieces of faculty information are declared here as the member data for this class.

Next, we need to take care of our Web method that will respond 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 process for it, we want to emphasize an important point that is easy to be confused by users, which is the Web service class and those classes we just created in the last sections.

The Web service class **WebServiceSQLSelect.vb** 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** are created by us, and they belong to application classes. These application classes will be instantiated to the associated objects that will be used by the Web methods developed in the system class **WebServiceSQLSelect.vb** to return the requested information as the project runs. Keep this difference in mind and this will help you understand them better as you 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.vb**. This method will use an object

instantiated from the application class `SQLSelectResult` we created in the previous section to hold and return the faculty information we requested.

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 data.
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 is a challenging task. There are two ways to perform this database connection in Web service applications. One way is to directly use the connection string and 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 better since the `Web.config` file provides an attribute `<connectionStrings/>` for this purpose, and ASP.NET 4.0 recommends to store 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 enter the following codes into this configuration file (just above the configuration ending tag `</configuration>`):

```
<connectionStrings>
  <add name = "sql_com" connectionString = "Server = localhost\SQL2008EXPRESS; _
    Integrated Security = SSPI;Database = CSE_DEPT;" />
</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. However, because of the space limitation, here, we used two lines to represent this attribute. But in your real codes, you must place this attribute in a single line in your `Web.config` file; otherwise, a grammar problem will 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. 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 that the database we are using is not a real SQL Server 2008 database, instead, we are using SQL Server 2008 Express, so we have to add the **InstanceName** of our database, which is **SQL2008EXPRESS**, 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 codes in 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.vb** by double-clicking on it from the Solution Explorer window, and enter the codes shown in Figure 9.17 into this page.

All modified codes have been highlighted in boldface, and let's see how this piece of codes works to test our connection string defined in the **web.config** file.

- A. The namespace that contains the SQL Server Data Provider is added into this page using the **Imports** command since we need to use those data components in this page.
- 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**, which works as an argument for this property, must be identical with the name we used for the connection

```

A Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
A Imports System.Data.SqlClient

<WebService(Namespace:= "http://www.cambridge.org/9780521712354/")> _
<WebServiceBinding(ConformsTo:= WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class WebServiceSQLSelect
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function HelloWorld() As String
B Dim cmdString As String=ConfigurationManager.ConnectionStrings("sql_conn").ConnectionString
C Dim sqlConnection As New SqlConnection
sqlConnection.ConnectionString = cmdString
D sqlConnection.Open()
E If sqlConnection.State <> Data.ConnectionState.Open Then
MsgBox("Database Open is failed")
F Else
MsgBox("Database Open is successful")
sqlConnection.Close()
End If
        Return "Hello World"
    End Function
End Class
  
```

Figure 9.17. The modified codes to test the connection string.

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.

Now you can run the project by clicking on the Start Debugging button. Click on 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 `MsgBox()` is used, and it is displayed behind the current Web page when it is activated. You need to move the current page by dragging it down, and then you can find that `MsgBox`. Click on the `OK` button to that `MsgBox`, and the project will be continued, and the running result will be displayed.

Another issue is that this piece of codes is only used for testing purposes, and we will modify this piece of codes and place it into a user-defined function `SQLConn()` later when we develop our real project.

9.3.7.2 Modify the Existing HelloWorld 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 the functions (2–7) listed at the beginning of this section.

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` format.
4. Create a new object based on our child application class `SQLSelectResult` and name this object as `SQLResult`.

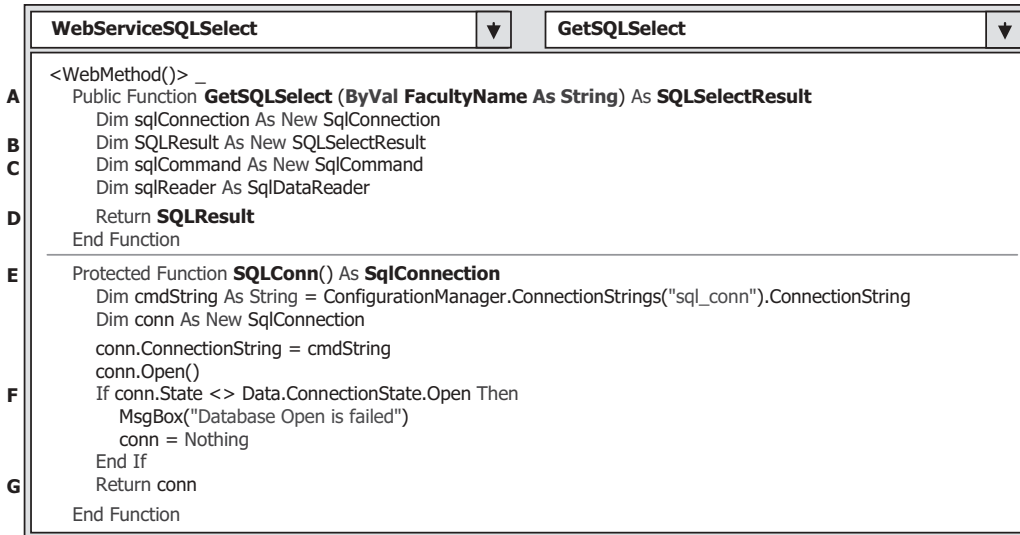


Figure 9.18. The modified Web method—GetSQLSelect().

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 “Hello World” string to the newly created object `SQLResult`.
7. Move the connection testing codes we developed in this section into a user-defined function `SQLConn()`.

Your finished Web method `GetSQLSelect()` is shown in Figure 9.18.

Let’s take a closer look at this piece of modified codes to see how it works.

- A. Modification steps 1, 2, and 3 listed above are preformed at this line. The method’s name and the returned data type are modified 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`, respectively.
- D. Modification step 6 is performed at this line, and the original returned data is updated to the current object `SQLResult`.
- E. Modification step 7 is performed here, and a new user-defined function `SQLConn()` is created with the codes we developed to test the connection string above.
- F. If this connection has failed, a warning message is displayed, and the returned connection object is assigned with `Nothing`. 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 the codes to execute the associated data object's method to perform the data query to the Faculty table based on the input parameter, Faculty Name.

9.3.7.3 *Develop the Codes to Perform the Database Queries*

To perform the database query via our Web service project, we need to perform the following coding developments:

- Add the major codes into our Web method to perform the data query.
- Create a user-defined subroutine `FillFacultyReader()` to handle the data assignments to our returned object.
- Create an error or exception-processing subroutine `ReportError()` to report any errors encountered during the project runs.

Now, let's first concentrate on adding the codes to perform the data query to our sample database `CSE_DEPT`.

Open our code-behind page and add the codes that are shown in Figure 9.19 into our Web method. The codes we developed in the previous sections have been highlighted with the gray color as the background. This sentence has been reworded for clarity. Please check and confirm it is correct.

Let's take a closer look at these newly added codes to see how they work.

- A.** The namespace `System.Data` is added into this page since some basic data components and data types are defined in this namespace, and we need to use those components in this page.
- B.** The query string is declared at the beginning of this method. One point you may have already noted is that a `+` symbol is used here to replace the concatenated operator `&` that is used in our Visual Basic.NET project before. The Web service page allows us to use this one as the concatenating operator.
- C.** 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.
- D.** The user-defined function `SQLConn()`, whose detailed codes are shown in Figure 9.18, is called to perform the database connection. This function will return a connection object if the connection is successful. Otherwise, the function will return a `Nothing` object.
- E.** If a `Nothing` is returned from calling the function `SQLConn()`, which means that the database connection has something wrong, a warning message is displayed, and the user-defined subroutine `ReportError()`, whose codes are shown in Figure 9.21, is executed to report the encountered error.
- F.** The Command object is initialized with the connection object that is obtained from the function `SQLConn()`, command type, and command text. Also, the input parameter `@facultyName` is assigned with the real input parameter `FacultyName`, which 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.
- G.** The `ExecuteReader()` method of the command class is called to invoke the `DataReader` to perform the data query from our Faculty table.

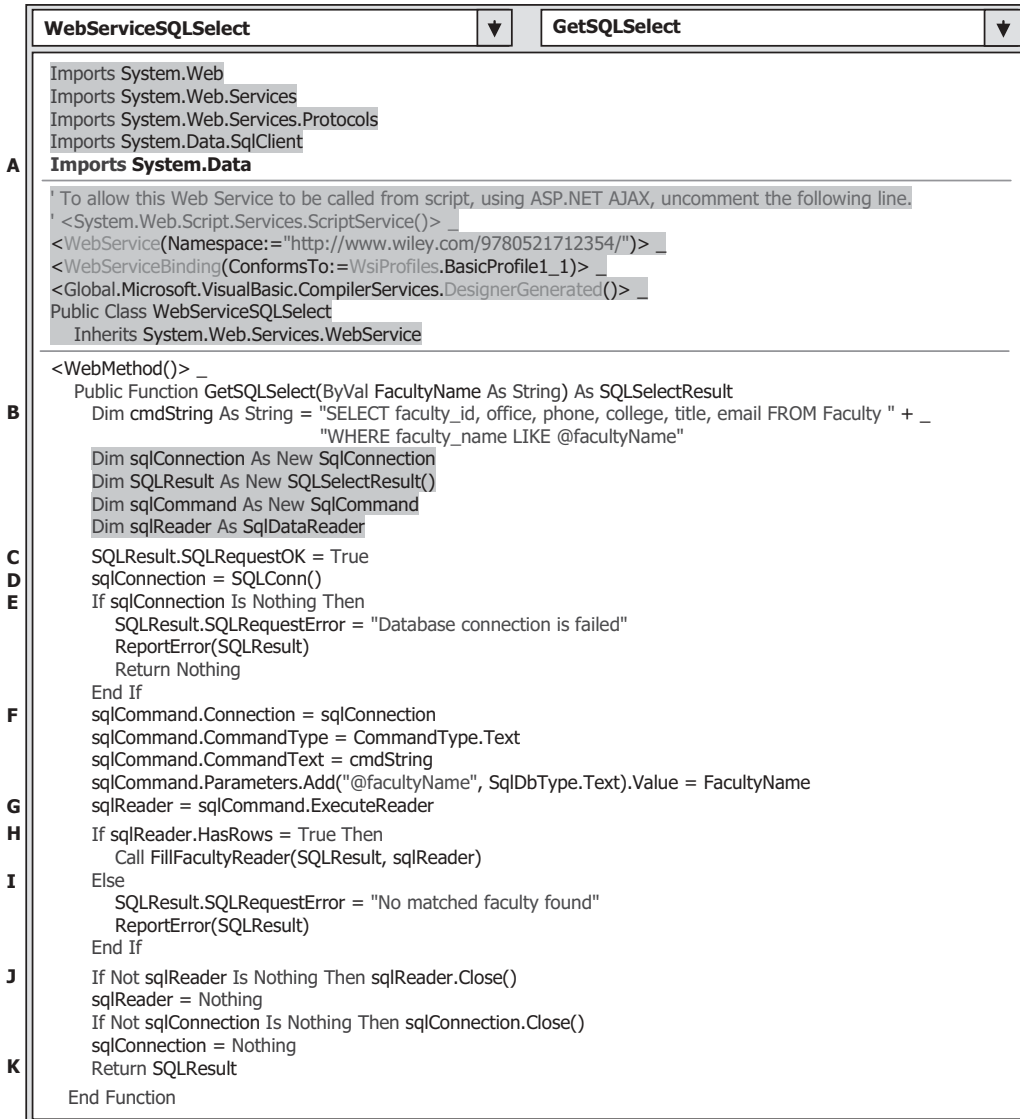


Figure 9.19. The modified codes for the Web method.

- H.** 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 subroutine FillFacultyReader() is called to assign all queried data columns to the associated member data we created in our child class SQLSelectResult. Two arguments, SQLResult, which is our returning object, and sqlReader, which is our DataReader object, are passed into that subroutine. 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 subroutine, 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. But the `DataReader sqlReader` is passed by using a passing-by-value mode, which means that only a copy of that object is passed into the subroutine, and any modification to that object is temporary.

- I. If the `HasRows` property returns `False`, this means that the data query has failed. An error message is assigned to the member data `SQLRequestError` defined in our base class `SQLSelectBase`, and our `ReportError()` subroutine is called to report this error.
- J. A cleaning job is performed to release all data objects used in this method.
- K. The object `SQLResult` is returned as the query result to our Web service.

Next, let's take care of developing the codes for our two user-defined subroutine procedures `FillFacultyReader()` and `ReportError()`.

9.3.7.4 Develop the Codes for Subroutines Used in the Web Method

The codes for the subroutine `FillFacultyReader()` are shown in Figure 9.20. Let's have a closer look at this piece of codes in this subroutine to see how it works.

- A. The `Read()` method of the `DataReader` is executed to read out the queried data row; in our case, only one row that is matched to the input faculty name is read out and fed into the `DataReader` object `sReader`.
- B. A `With . . . End With` block is utilized here to simplify the assignment operations. The object `sResult`, which is our returning object, is attached after the keyword `With`, and all member data of that object can be represented by using the dot (.) operator without needing the prefix of that object. Each data column in the `Faculty` table can be identified by using its name from the `DataReader` 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 `DataReader 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 if this method is used.

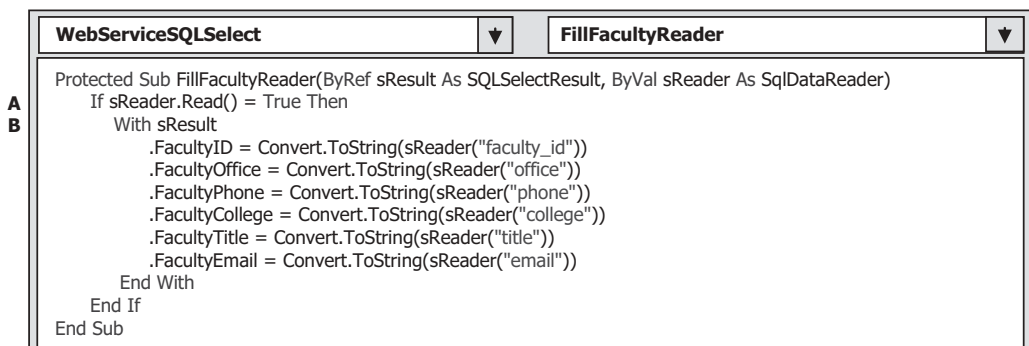


Figure 9.20. The codes for the subroutine `FillFacultyReader()`.

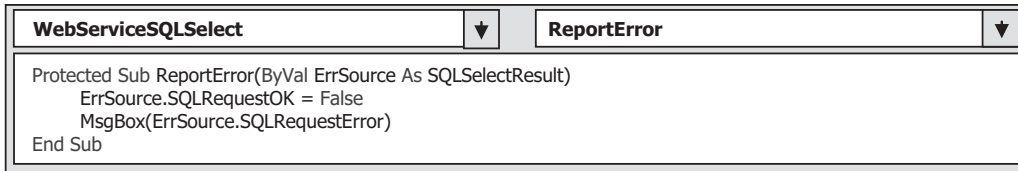


Figure 9.21. The codes for the subroutine ReportError().

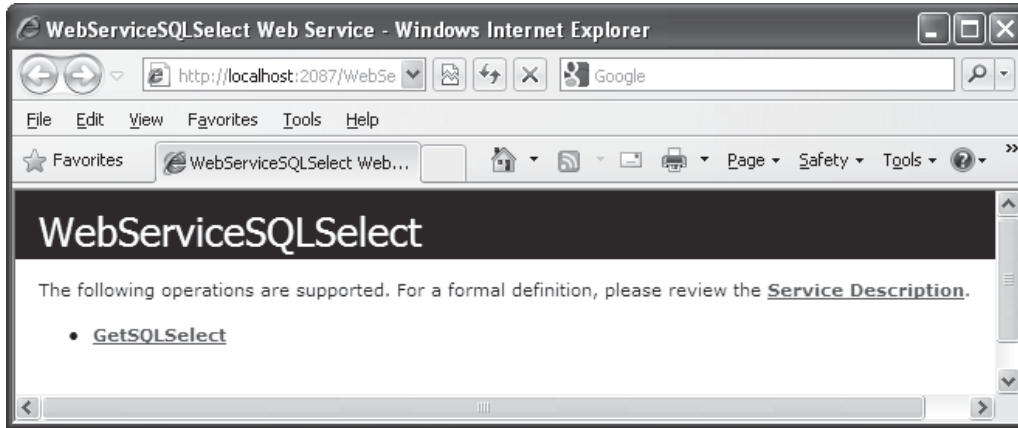


Figure 9.22. The running status of the Web service.

The key point for this subroutine is the passing mode for the first argument. A passing-by-reference mode is used for our returning object, and this is equivalent to returning an object from a function.

The detailed codes for the subroutine ReportError() are shown in Figure 9.21.

The input parameter to this subroutine 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 functionality. Click on the Start Debugging button to run the project, and the built-in Web interface is displayed, which is shown in Figure 9.22.

Click on our Web method **GetSQLSelect** to open the built-in Web interface for our Web method, which is shown in Figure 9.23. Enter the faculty name **Ying Bai** into the **FacultyName** box as an input Value, 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 9.24.

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

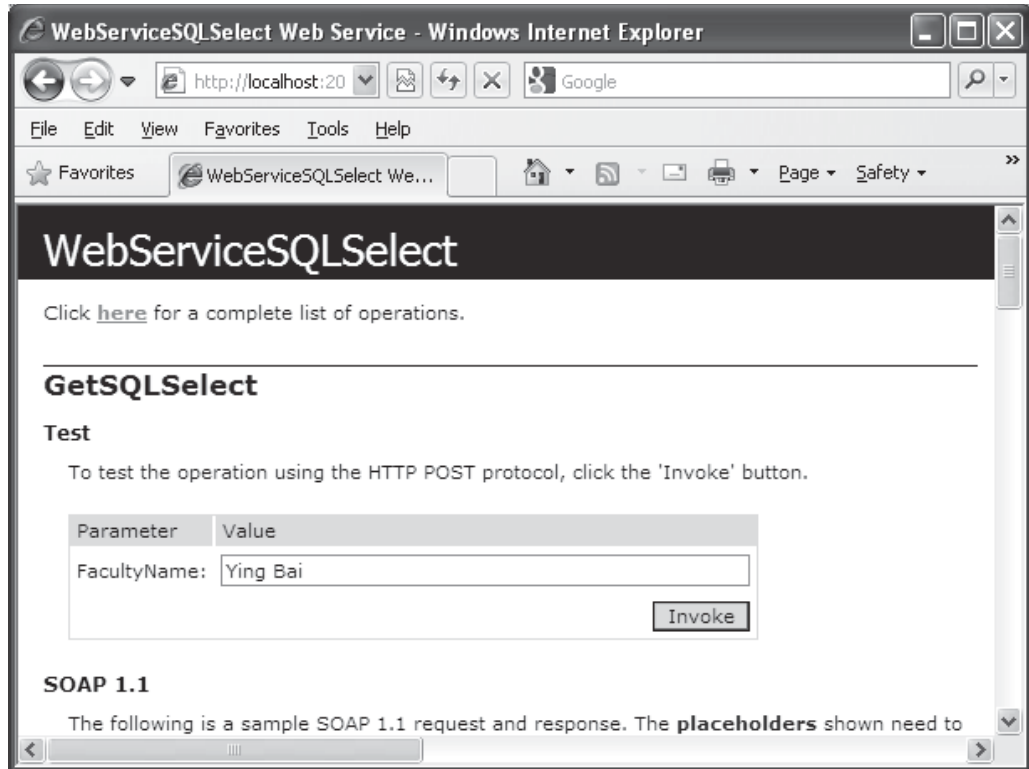


Figure 9.23. The running status of our Web method.

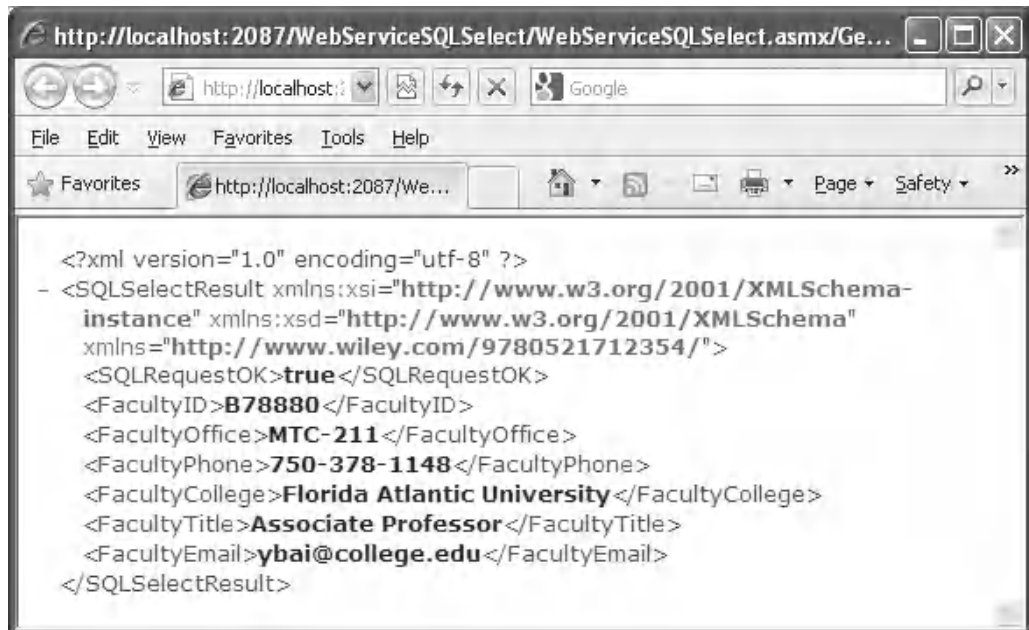


Figure 9.24. The running result of our Web service project.

<FacultyID>. . .</FacultyID>, and the name of this tag is defined in our child class `SQLSelectResult`. Our first Web service method is very successful.

As we mentioned before, a Web service did not provide any user interface, and one needs to develop some user interfaces to consume the Web service if one wants to display those pieces of information obtained from the Web services. Here, a built-in Web interface is provided by Microsoft to help users to display queried information from the Web services. In most real applications, users need to develop user interfaces themselves to perform these data displaying or other data operations.

Click on the **Close** button that is located at the upper-right corner of the page to close our Web service project.

9.3.8 Develop the Stored Procedure 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 process 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 body to strengthen the integrity of the query.

9.3.8.1 Develop the Stored Procedure *WebSelectFacultySP*

Now, let's first develop our stored procedure in the Server Explorer window in Visual Studio.NET environment.

Open the Visual Studio.NET and open the Server Explorer window, and 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 that are shown in Figure 9.25 into this code body as our new stored procedure.

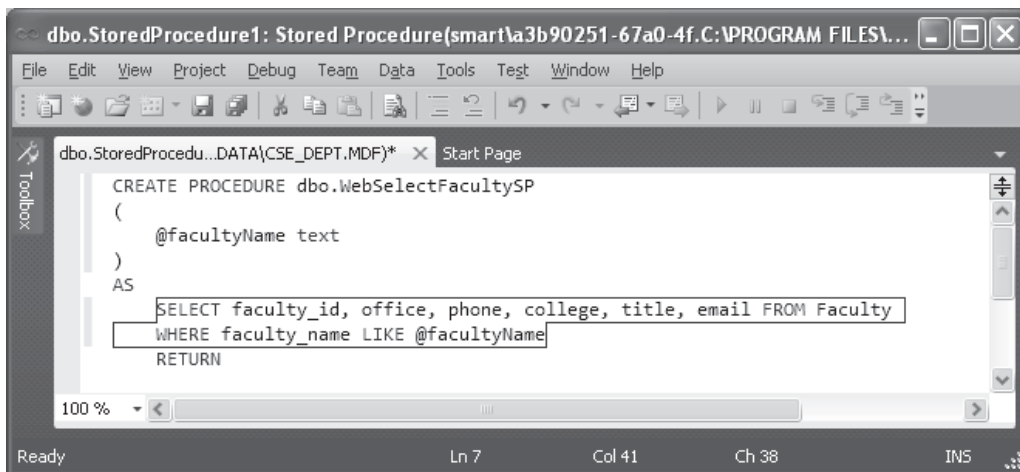


Figure 9.25. The stored procedure `dbo.WebSelectFacultySP`.

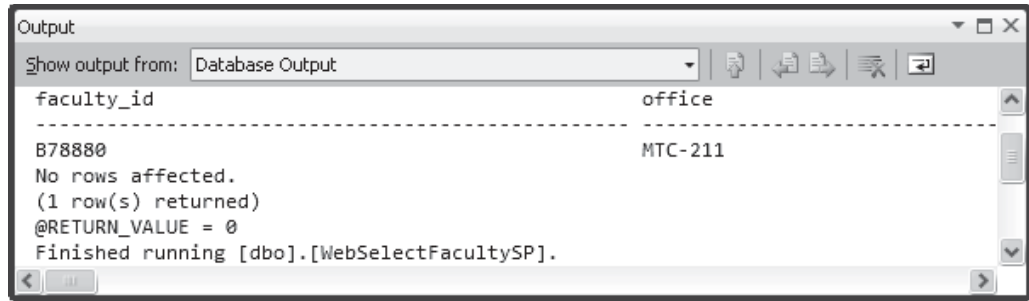


Figure 9.26. The running result of the stored procedure.

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 from the pop-up menu to open the Run Stored Procedure wizard. Enter the faculty name **Ying Bai** to the **Value** box as the input parameter and click on the **OK** button to run this stored procedure. The running result is displayed in the **Output** window that is located at the bottom of this wizard, which is shown in Figure 9.26.

All queried six columns that 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 development 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. Then, copy this whole coding body and paste it to the bottom of our code-behind page (must be inside our Web service class). Perform the modifications shown in Figure 9.27 to this copied Web method to make it as our second Web method **GetSQLSelectSP()**. The modified parts have been highlighted in bold.

- A.** Change the Web method's name by attaching two letters **SP** to the end of the original Web method's name, and the new 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 during the project runs because the project cannot find the desired stored procedure.

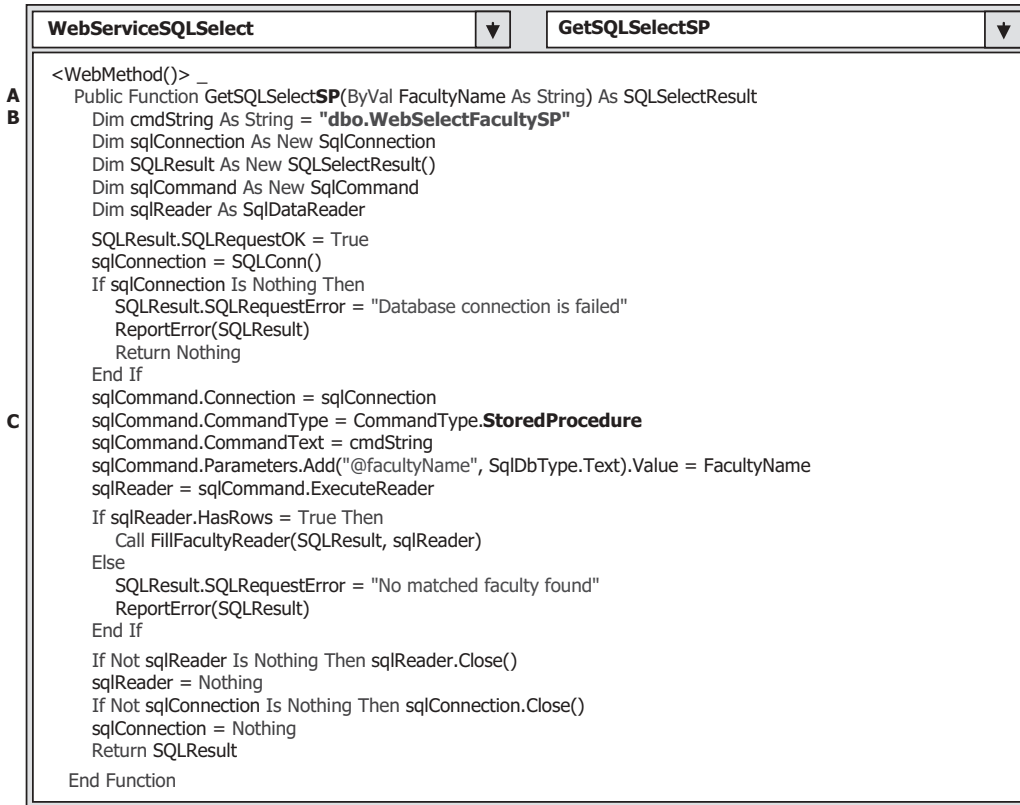


Figure 9.27. The modified Web method—GetSQLSelectSP().

- C. Change the **CommandType** property of the **Command** object from the **CommandType.Text** to the **CommandType.StoredProcedure**. This changing 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.

Now you can run the project to test this new Web method. Click on the second Web method **GetSQLSelectSP** as the project runs, enter a desired faculty member, such as **Ying Bai**, into the **FacultyName** box, and click on the **Invoke** button to run it. The same running result that 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 the stored procedure in the Web service project.

Next, we will discuss how to use a **DataSet** as a returning object to contain all pieces of queried information we need from running a Web service project.

9.3.9 Use DataSet as the Returning Object for the 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, 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 could be greatly reduced when a DataSet is used, especially for a large block of data that are queried via the Web service project.

To distinguish from those Web methods we developed in the previous sections, we can create another new Web method `GetSQLSelectDataSet()` and add it into our Web service project. To do that, open our code-behind page if it is not yet opened, highlight and select the whole coding body of our first Web method `GetSQLSelect()`, including the method header and coding body. Copy and paste it to the bottom of our page (must be inside our Web service class). Perform the modifications shown in Figure 9.28 to this copied Web method to make it as our third Web method. The modified codes have been highlighted in bold.

Let's have a closer look at this modified Web method to see how it works.

- A. The Web method's name is modified by attaching the `DataSet` to the end of the original method name. The data type of the returning object is the `DataSet`, which means that this Web method will return a `DataSet`.



Figure 9.28. The modified Web method—`GetSQLSelectDataSet()`.

- B.** Two new data objects, `FacultyAdapter` and `dsFaculty`, are created. The first object works as a `DataAdapter` and the second works as 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, that is, 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 stored in the local integer variable `intResult` to indicate whether this calling is successful or not.
- E.** If the returned value is zero, which means that no row has been retrieved from the Faculty table in our sample database and no any 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 subroutine `ReportError()` later.
- 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 one since only one record is returned from the Faculty table in our sample database and filled into the `DataSet`.
- G.** Finally, the filled `DataSet` `dsFaculty`, exactly the filled Faculty table in this `DataSet`, is returned to the Web service.

Now, we can run the project to test this returned `DataSet` functionality. Click on the Start Debugging button to run the project. Now, we have three Web methods available to this Web service, which is shown in Figure 9.29.

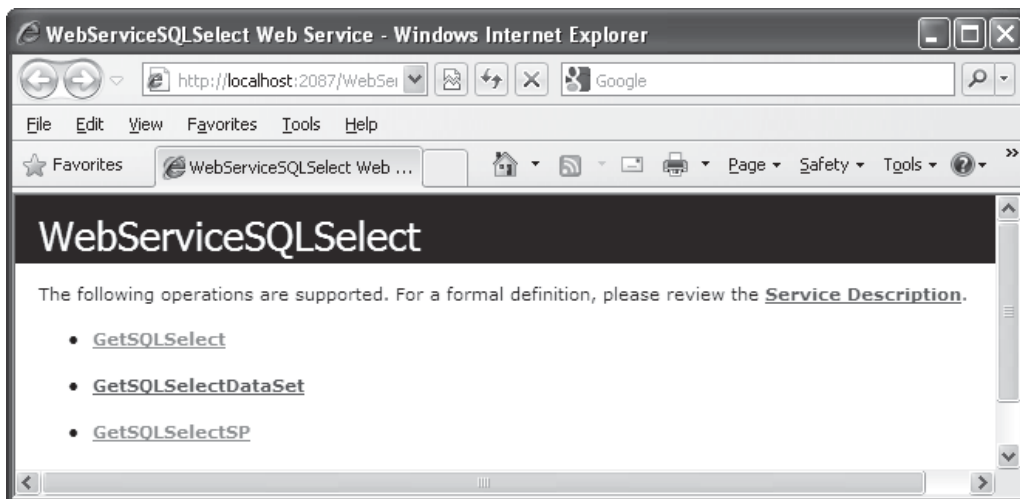


Figure 9.29. Three Web methods in built-in Web interface window.

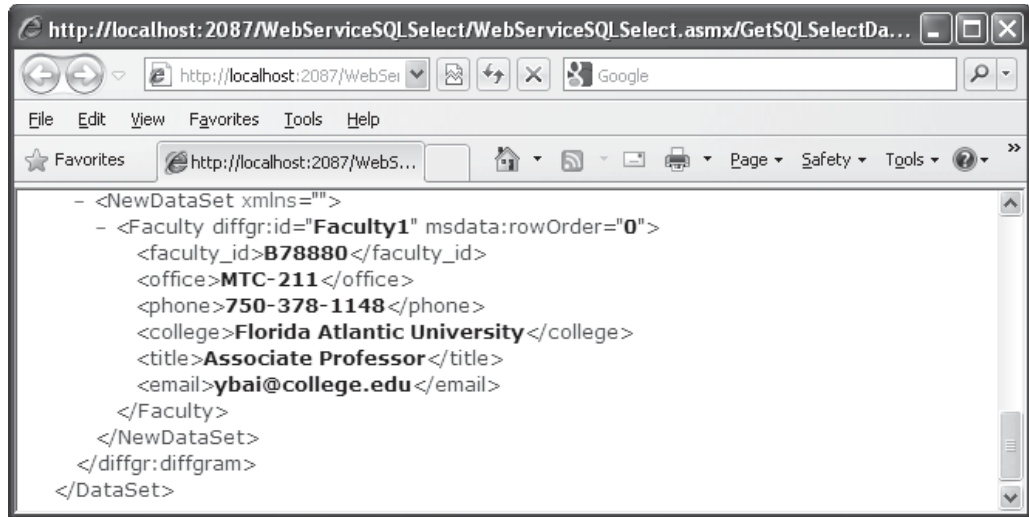


Figure 9.30. The running result of the Web method GetSQLSelectDataSet().

Click on the second Web method `GetSQLSelectDataSet` from the built-in Web interface window and enter the faculty name Ying Bai into the Value box as our desired faculty member. Then click on the Invoke button to call this Web method to perform the data query. The running result is returned and shown in Figure 9.30.

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 information are returned to our Web service project and displayed in the built-in Web interface window, as shown in Figure 9.30.

At this point, we have finished all codes 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 in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Next, we need to develop some professional Windows-based or Web-based applications with beautiful GUIs to use or consume our Web service project. Those Windows-based or Web-based applications can be considered as Web service clients.

9.3.10 Build Windows-Based Web Service Clients to Consume the Web Services

To use or consume 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 2010 will try to find all Web services available to our applications. The following operations will be performed by Visual Studio.NET 2010 during this process:

1. When looking for Web services from the local computers, Visual Studio.NET 2010 will check all files that include a Web service main page with an .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 2010 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 your 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 detailed 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 if 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 2010. 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 sections.

Open Visual Studio.NET 2010 and create a new Visual Basic Windows-based project, and name it as WinClientSQLSelect.

Now let's add a Web reference to our new project.

There are two ways we can use to select the desired Web services and add it as a reference to our client project: one way is to use the **Browser** provided by the Visual Studio.NET 2010 to find the desired Web service, and another way is to copy and paste the desired Web service URL to the URL box located in this Add Web Reference wizard. In order to use the second way, you need first to run the Web service, and then copy its URL and paste it to the URL box in this wizard if you have not deployed that Web service to IIS. If you did deploy that Web service, you can directly type that URL into the **Address** box in this wizard.

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 should use the second way to find our Web service. Perform the following operations to add this Web reference:

1. Open Visual Studio.NET 2010 and our Web service project **WebServiceSQLSelect**, and click on the **Start Debugging** button to run it.
2. Copy the URL from the **Address** bar in our running Web service project.
3. Then open another Visual Studio.NET 2010 and open our client project **WinClientSQLSelect**.
4. Right-click on our client 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** wizard.
5. Click on the **Advanced** button located at the lower-left corner on this wizard to open the **Service Reference Settings** wizard.
6. Click on the **Add Web Reference** button to open the **Add Web Reference** wizard, which is shown in Figure 9.31.
7. Paste that URL we copied from step 2 into the **URL** box in the **Add Web Reference** wizard and click on the **Green Arrow** button to enable the Visual Studio.NET 2010 to begin to search it.
8. When the Web service is found, the name of our Web service is displayed in the right pane, which is shown in Figure 9.31.
9. 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.

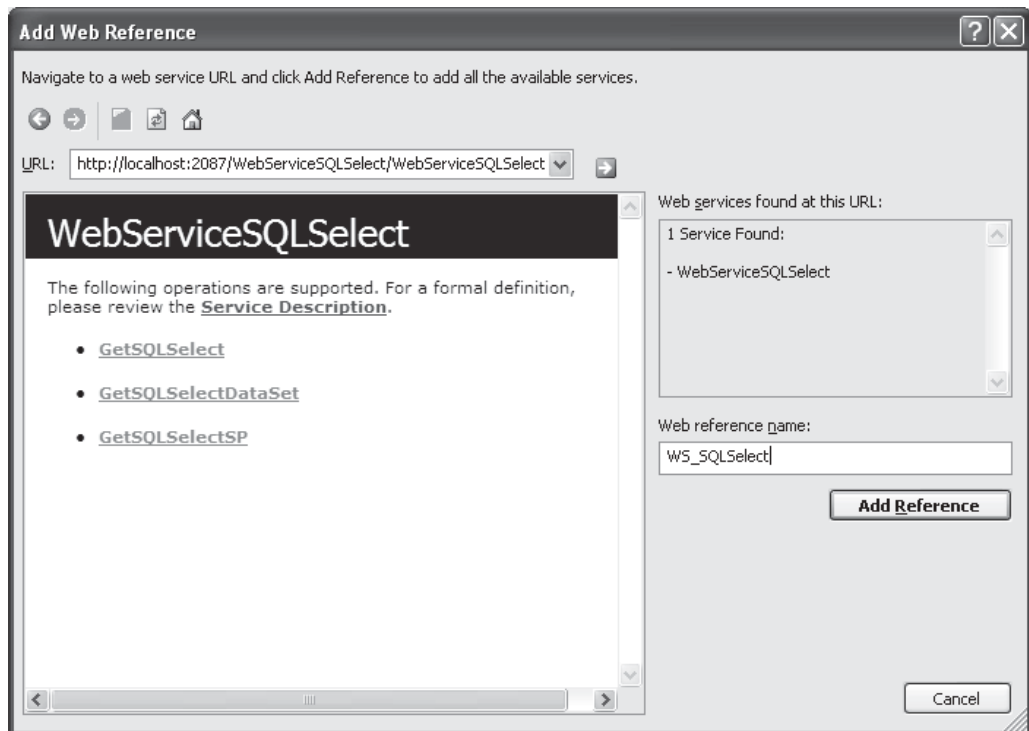


Figure 9.31. The Add Web Reference wizard.

10. 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 client project. This reference is the so-called Web service proxy class.

Next, let's develop the GUI by adding useful controls to interface to our Web service and display the queried information.

9.3.10.2 *Develop the Graphic User Interface for the Windows-Based Client Project*

Perform the following modifications to our new project:

1. Rename the Form File object from the default name **Form1.vb** to our desired name **WinClient Form.vb**.
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 the GUI located in the project **SQLUpdateDeleteRTOObject** we developed in Chapter 7. Open that project from the folder **DBProjects\Chapter 7** that is located at the Wiley ftp site (refer to Figure 1.2 in 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 **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 form window, and paste those controls we copied from the Faculty form in the project **SQLUpdateDeleteRTOObject**.

The purpose of the combo box control **Query Method** is to select one of three different query methods developed in our Web service project to get the desired faculty information:

1. Method 1: uses an object to return our queried information.
2. Method 2: uses a stored procedure to return our queried information.
3. Method 3: uses a **DataSet** to return our queried information.

The Faculty Name combo box control is used to select the desired faculty member as the input parameter for the Web method to pick up our desired faculty information. 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 by the user. Our client project will be connected to our Web service based on the Web reference we provided, and the selected method will be called based on the method chosen from the **Query Method** combo box control to perform the data query to retrieve the desired faculty record from our sample database, and display it in this GUI.

Now let's take care of the coding process for this project to connect to our Web service using the Web reference we developed in the last section.

9.3.10.3 Develop the Code to Consume the Web Service

The coding job can be divided into four parts:

1. The coding development for the `Form_Load` event procedure to initialize the Query Method and the Faculty Name combo box controls. The first initialization is to 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. The coding process for the `Back` button click event procedure to terminate the project.
3. The coding development for the `Select` button click event procedure.
4. The coding process for user-defined subroutine procedures, such as the `ShowFaculty()`, `ProcessObject()`, `FillFacultyObject()`, and `FillFacultyDataSet()`.

The main coding job is performed inside the `Select` button's click event procedure. As we discussed, as this button is clicked by the user, a connection to our Web service should be established using the Web reference we set up in the previous section. So we need first to create an object based on that Web reference or instantiate that Web service to get an instance, and then we can access the different Web methods to perform our data query. This process is called instantiating the proxy class and invokes the Web methods. The protocol to instantiate a proxy class is:

```
Dim newInstanceName As New WebReferenceName.WebServiceName
```

After this new instance is created, a connection between our client project and our Web service can be set up by using this instance. The pseudo codes for this event 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 to the real object `SQLSelectResult` developed in the Web service. We can easily access the related Web method to perform our data query and to 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 Query Method combo box control, different Web methods can be called to perform the related data query.
- E. The returned data that are stored in the real object is assigned to our mapping object, and each piece of information can be retrieved from this object and displayed in our GUI.
- F. If a `DataSet` method is used, the returned `DataSet` object is assigned to our mapping `DataSet`, and the subroutine `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 the Codes for the Form_Load Event Procedure Now let's begin to develop the codes for the `Form_Load` event procedure to complete the initialization jobs listed in step 1 above.

Open the `Form_Load` event procedure by selecting the (FacultyForm Events) item from the Class Name combo box, and `Load` item from the Method Name combo box

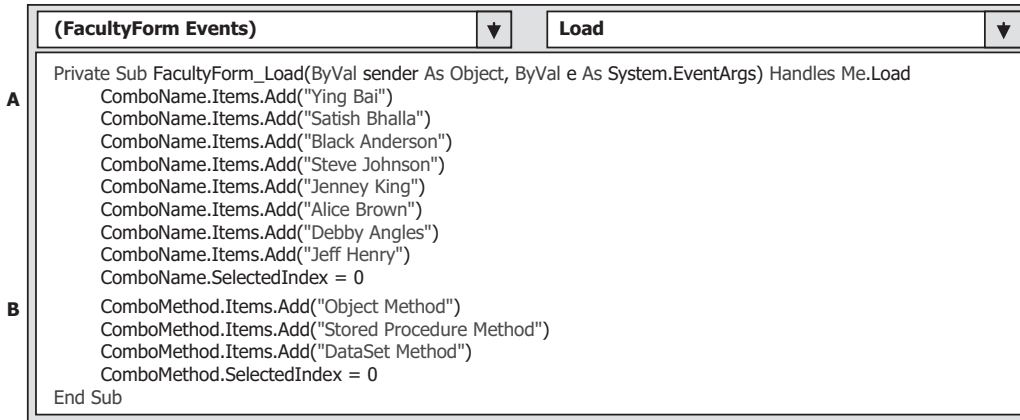


Figure 9.32. The codes for the Form_Load event procedure.

from the code window of the **FacultyForm**. Enter the codes that are shown in Figure 9.32 into this event procedure.

Let's take a closer look at this piece of codes to see how it works.

- A.** Eight default faculty members are added into the Faculty Name combo box control using the **Add()** method. This will allow users to select one desired faculty from this combo box control to perform the data query as the project runs. The default faculty is the first one by setting the **SelectedIndex** property to zero.
- B.** Three Web query methods are also added into the Query Method combo box control 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.

The codes for the **Back** button's click event procedure are very simple. Open this event procedure and enter **Me.Close()** into this event procedure, which means that the project will be terminated as soon as the user clicks on this button as the project runs. The **Close()** method tells Visual Basic.NET to terminate the current project.

Next, let's build the codes for the **Select** button's Click event procedure.

9.3.10.3.2 Develop the Codes for the Select Button Click Event Procedure Open the **Select** button's click event procedure and enter the codes that are shown in Figure 9.33 into this procedure.

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

- A.** Some data objects are created at the beginning of this event procedure, 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 to 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 this mapping object later. Also, a new **DataSet** object that is used to call the Web method that returns a **DataSet**.
- B.** If the user selected the **Object Method** from the Query Method combo box 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



Figure 9.33. The codes for the Select button click event procedure.

query. The selected faculty that is located in the Faculty Name combo box 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 subroutine **ProcessObject()** is executed to pick up all pieces of retrieved information from the returned object and displays 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 referenced 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.** The user-defined subroutine **ProcessObject()** is executed to pick up all pieces of retrieved information from the returned object and displays them in this form.
- H.** If users selected the **DataSet Method**, the Web method **GetSQLSelectDataSet()** is called through the instance of the Web referenced class, and the method returns a **DataSet** that contains our desired faculty record.
- 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.** The subroutine **FillFacultyDataSet()** is called to pick up all pieces of retrieved information from the returned **DataSet**, and displays them in this form window.

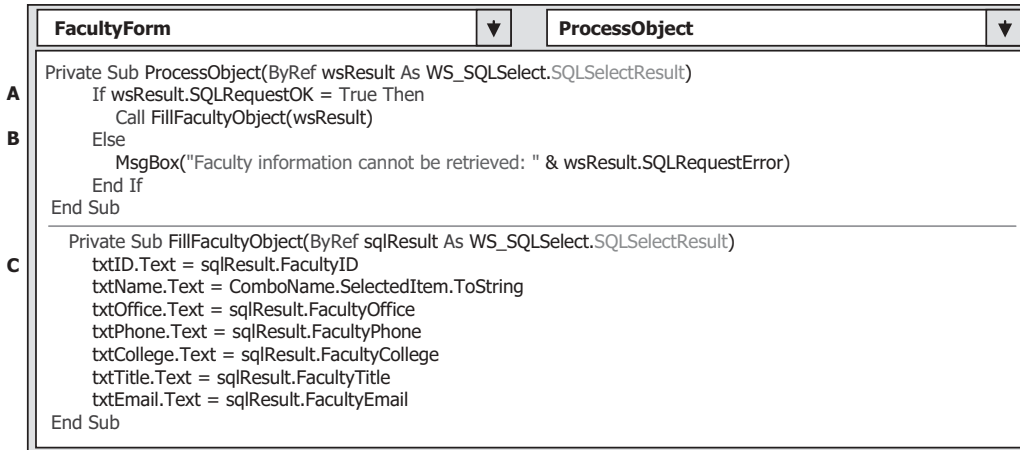


Figure 9.34. The codes for two user-defined subroutines.

9.3.10.3.3 Develop the Codes for User-defined Subroutines The codes for the user-defined subroutines `ProcessObject()` and `FillFacultyObject()` are shown in Figure 9.34. Both subroutines 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 codes is:

- A.** If the member data `SQLRequestOK`, which 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 subroutine `FillFacultyObject()` is called and executed. This subroutine picks up each piece of information from the queried object `wsResult` and displays it in this form window.
- B.** Otherwise, some exceptions occurred, and a warning message is displayed with a message box.
- C.** As the subroutine `FillFacultyObject()` is called, all six pieces of faculty information stored in the returned object are picked up and assigned to the associated textbox in this form to be displayed. The faculty name can be obtained directly from the Faculty Name combo box control from this form window.

The codes for the subroutine `FillFacultyDataSet()` are shown in Figure 9.35. The argument passed into this subroutine is an instance of `DataSet` we created in the `Select` button click event procedure.

Let's take a look at this piece of codes 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 newly created object `FacultyTable`. Because the `DataSet` we created in the `Select` button click event procedure 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.

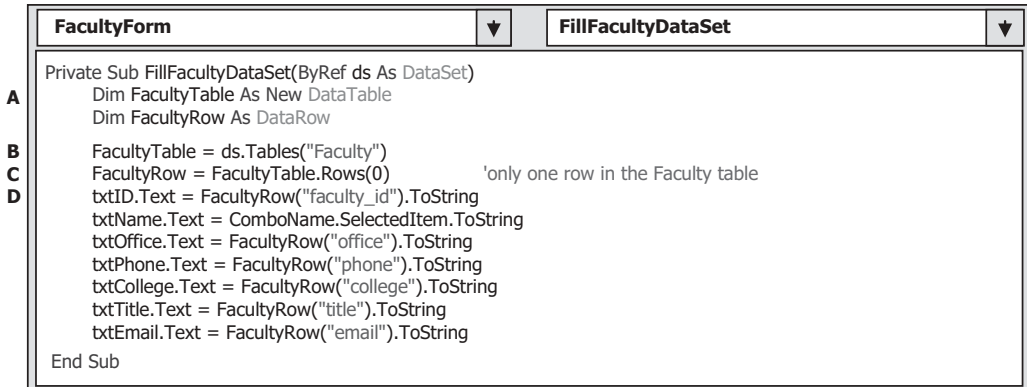


Figure 9.35. The codes for the subroutine FillFacultyDataSet().

- C. Since we only request one record or one row from the Faculty table, the returned Faculty table contains only one row information that is located at the top row with an index of zero. This one row information is assigned to our FacultyRow object we created above.
- D. We can access each column from returned one row data 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 development for this Windows-based Web service client project. We have one more step to go to complete this client project, which is to add another user-defined subroutine ShowFaculty() to display the requested faculty photo in the image box in this form.

9.3.10.3.4 Develop a Subroutine ShowFaculty to Display the Faculty Image All default faculty and student images can be found in the folder Images that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). To display the requested faculty image in the form, following jobs are needed to be performed:

1. Copy all default faculty and student image files from the folder Images that is located at the Wiley ftp site, and paste them into our current client project folder, exactly into the Debug folder that is under our project's bin folder. For our case, one needs to paste all faculty and student image files into 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 subroutine. Open the code window from our client project, and type the codes that are shown in Figure 9.36 into this code window to create our subroutine ShowFaculty().

The codes for this subroutine are straightforward with no trick. A Selection-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

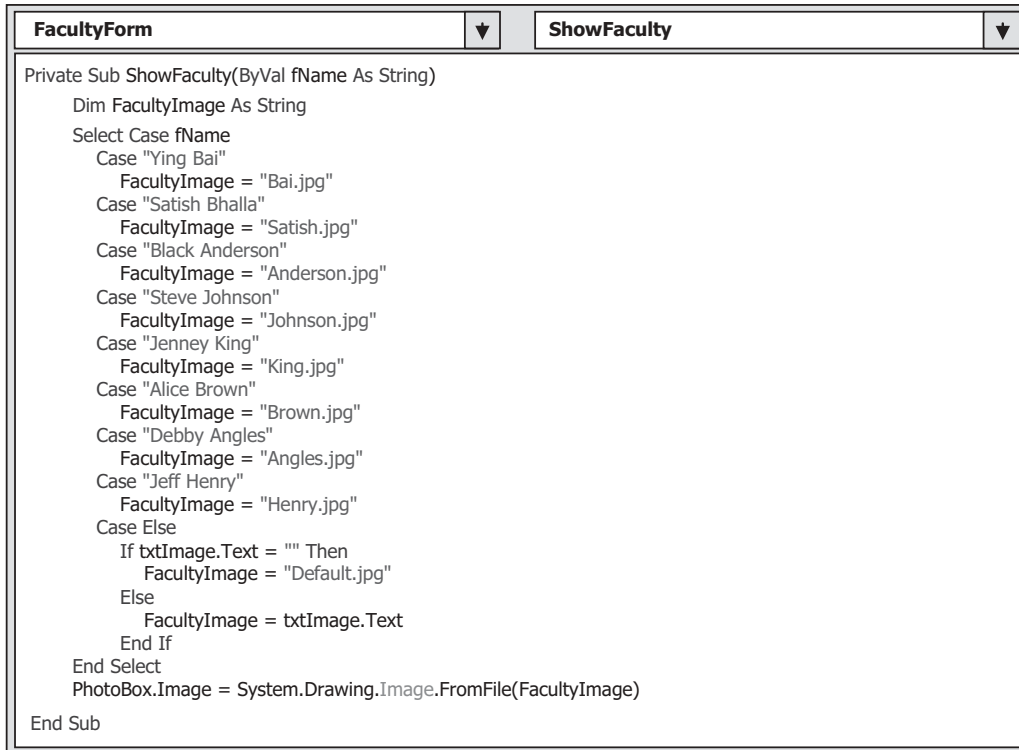


Figure 9.36. The codes for the subroutine ShowFaculty().

method `Image.FromFile()`, and then is displayed in the `PhotoBox` control in this form window. A default faculty image will be used if no matched faculty image can be found.

To call this subroutine to display the selected faculty image, add one more statement that is shown below to the **Select** button's click event procedure. The location to add this statement is the first code line under the data objects declaration part.

```
Call ShowFaculty(ComboName.Text)
```

Now we can start to run this client project to interface to 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, which is that you must run our Web service project `WebServiceSQLSelect` first to allow our Web service available to all clients. Otherwise, you may encounter some running exceptions (such as the Web service or remote computer cannot be found) during your client project runs.

Open Visual Studio.NET and our Web Service project `WebServiceSQLSelect`, and click on the Start Debugging button to run it. Once our Web service project runs, you can close it and access it using our client project without problem at all. One point is that our Web service project must be kept in the running status (even the page has been closed) in order to allow our client project to access and interface to it. An exception will be encountered if you stop our Web service when you try to access it using our client project.

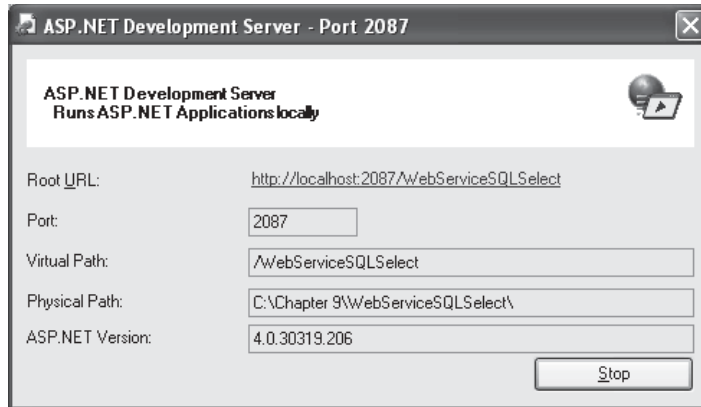


Figure 9.37. The running status of our Web service project WebServiceSQLSelect.



Figure 9.38. The running status of our client project.

Make sure that our Web service has been run once, and it is in the running status, which can be identified by a small Web service running icon in the task bar on the bottom of your screen. You can double click on that icon to open the running status of our Web service project, which is shown in Figure 9.37.

Then start our client project by clicking on the Start Debugging button from the project WinClientSQLSelect. The running status is shown in Figure 9.38.

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 record. The returned faculty data is displayed in the associated textboxes with the faculty photo, which is shown in Figure 9.38.

You can try to select other two Web methods, either the Stored Procedure or the DataSet method, and other faculty members to perform this data query. The running

result confirmed 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.

A complete Windows-based Web service client project **WinClientSQLSelect** can be found in the folder **DBProjects\Chapter 9** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). You need to load both our client and our Web service projects from that site and install them in your computer if you want to run and test this client project. Also, you must run our Web service project one time 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 consume our Web service by retrieving the desired faculty information.

9.3.11 Build Web-Based Web Service Clients to Consume the Web Service

Developing a Web-based client application to consume 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 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 this Web service is consumed 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 as our new Web-based Web service client project **WebClientSQLSelect**. In fact, 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 application project and only copy and modify the Faculty page.

The developing process in this section can be divided into the following parts:

1. Create a new ASP.NET website project **WebClientSQLSelect** and add an existing website 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 page window of the **Faculty.aspx** to meet our data query requirements.
3. Modify the codes in the related event procedures of the **Fcaulty.aspx.vb** 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** event procedure.
 - B. Modify the codes in the **Select** button's click event procedure.
 - C. Add three user-defined subroutines, **ProcessObject()**, **FillFacultyObject()**, and **FillFacultyDataSet()**. These three subroutines are basically identical with those we developed in the last Windows-based Web service client project **WinClientSQLSelect**. One can copy and paste them into our new project. The only modification is for the subroutine **ProcessObject()**.
 - D. Modify the codes in the **Back** button's click event procedure.

Now let's start with the first part listed above.

9.3.11.1 Create a New Web Site 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 the OK button to create this new Website project.

Perform the following operations to add an existing Web page Faculty.aspx into our new website project:

1. Remove the Default.aspx from the Solution Explorer window since we do not need this page in this application.
2. Right-click on our new project WebClientSQLSelect from the Solution Explorer window, and select the item Add Existing Item from the pop-up menu to open the Add Existing Item wizard.
3. Browse to our Web project SQLWebSelect that can be found in the folder DBProjects\Chapter 8 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Select the Web page Faculty.aspx from the list and click on the Add button to add this Web page into our new Website project.

One issue we need to emphasize is that we had better add all faculty image files into our new project before we can continue to the next step. In this way, the selected faculty image can be displayed as that faculty's information is queried. This step is highly recommended, since we need those faculty image files later when we display each of them for each selected faculty as we perform the data query. You can find all faculty and student image files in the folder Images that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Perform the following operations to add all faculty and student image files into our project:

1. Right-click on our new project icon WebClientSQLSelect from the Solution Explorer window, and select the Add Existing Item from the pop-up menu.
2. Browse to the Images folder, select all image files, and then click the Add button to add all of image files into our current Website project.

Now let's handle to add a Web reference to our project to access the Web service we built in the previous section.

9.3.11.2 Add a Web Service Reference and Modify the Web Form Window

Perform the following operations to add this Web reference:

1. Open Visual Studio.NET 2010 and our Web service project WebServiceSQLSelect, and click on the Start Debugging button to run it.
2. Copy the URL from the Address bar in our running Web service project.
3. Then open another Visual Studio.NET 2010 and open our Web client project WebClientSQLSelect.
4. Right-click on our client project WebClientSQLSelect from the Solution Explorer window, and select the item Add Web Reference from the pop-up menu to open the Add Web Reference wizard, which is shown in Figure 9.39.

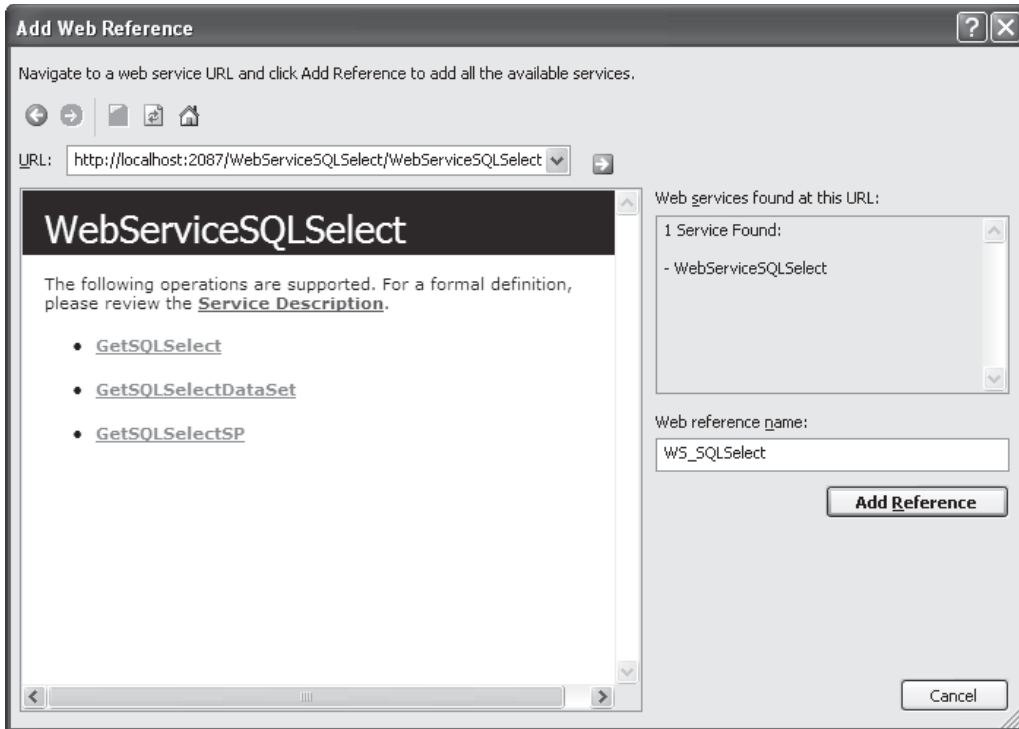


Figure 9.39. Add a Web reference.

5. Paste that URL we copied from step 2 into the URL box in the Add Web Reference wizard and click on the **Green Arrow** button to enable the Visual Studio.NET 2010 to begin to search it.
6. When the Web service is found, the name of our Web service is displayed in the right pane, which is shown in Figure 9.39.
7. 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.
8. Click on the **Close** button from our Web service built-in Web interface window to close our Web service page.

Immediately, you can find that the following three files are created in the Solution Explorer window under the folder of the newly added Web reference:

- WebServiceSQLSelect.disco
- WebServiceSQLSelect.discomap
- WebServiceSQLSelect.wsdl

This reference is the so-called Web service proxy class.

Next, let's begin the code modification process to build the codes for this project.

9.3.11.3 Modify the Codes for the Related Event Procedures

The first modification is to add a combo box control Query Method to the Faculty.aspx page to enable users to select one of three query methods to perform the related data query operation.

9.3.11.3.1 Add a Combo Box Control Query Method to the Faculty Page Open the Faculty.aspx page and add one more combo box control and an associated label Query Method just under the Faculty Name combo box control in this page. Name this control as **ComboMethod**. Your modified Faculty.aspx page should match the one that is shown in Figure 9.40.

The second modification is to change the codes in the Page_Load event procedure and some global variables.

9.3.11.3.2 Modify the Codes in the Page_Load Event Procedure Perform the following changes to complete this modification:

1. Remove the second Imports command Imports System.Data.SqlClient from the top of this page since we do not need it in this application.
2. Remove the form level variable FacultyTextBox(6) that is a textbox array.
3. Remove the If block inside the Page_Load event procedure and the associated global connection object that is stored in the Application state Application("sqlConnection").
4. Add the codes to display three Web methods in the Query Method combo box control.

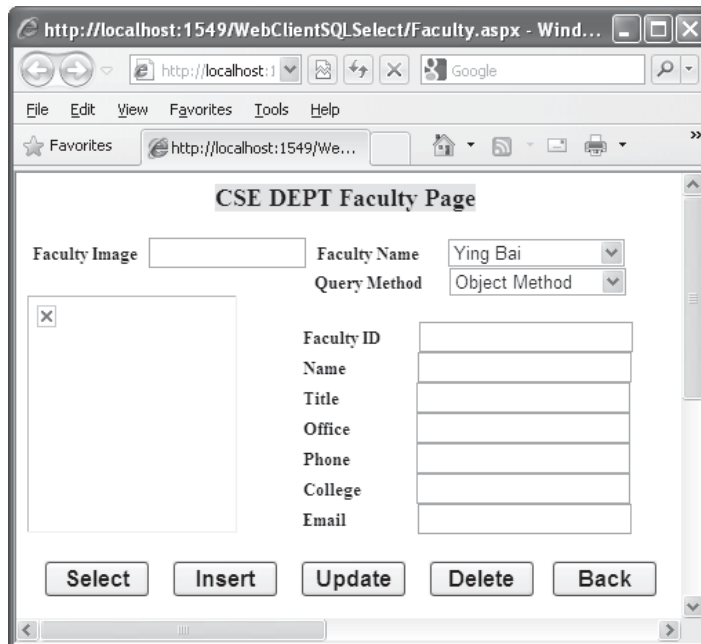


Figure 9.40. The modified Faculty.aspx page.

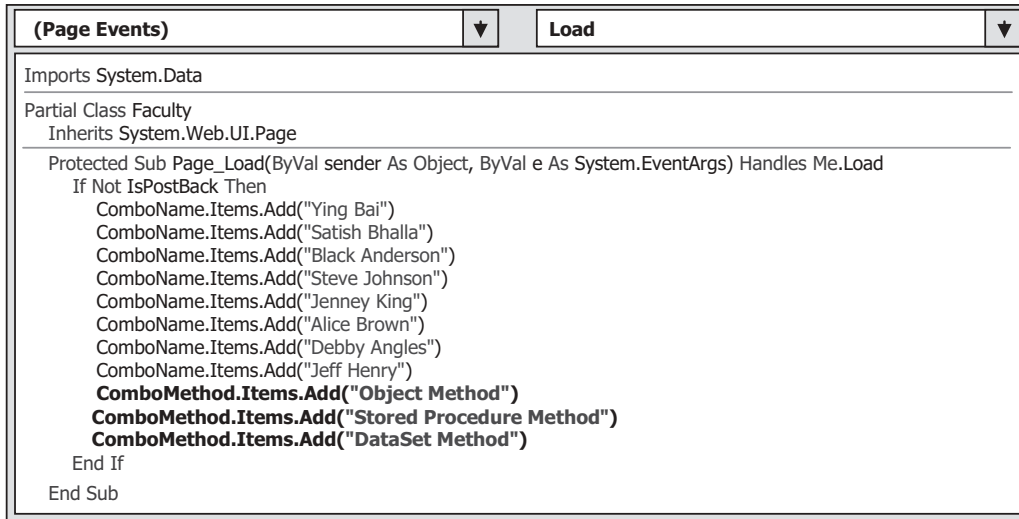


Figure 9.41. The modified Page_Load event procedure.

Your finished codes for the Page_Load event procedure should match the one that is shown in Figure 9.41. The newly added codes have been highlighted in bold.

The next modification is to change the codes inside the **Select** button's click event procedure.

9.3.11.3.3 Modify the Codes in the Select Button Event Procedure Replace all codes in this event procedure with the modified codes shown in Figure 9.42. This replacement includes:

- A.** Create the following 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 the DataSet class
- B.** Create a local string variable **errMsg**, and it is used to store the possible error message.
- C.** Call the subroutine **ShowFaculty()** to display the selected faculty image.
- 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 subroutine **ProcessObject()** is executed to assign the retrieved faculty information to the associated textbox in our Web page to display each of 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 referenced 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. The subroutine



Figure 9.42. The modified codes for the Select button's click event procedure.

`ProcessObject()` is executed to pick up all pieces of retrieved information from the returned object and displays them in this page.

- G.** If users selected the **DataSet Method**, the Web method `GetSQLSelectDataSet()` is called through the instance of the Web referenced 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 subroutine `FillFacultyDataSet()` is called to pick up all pieces of retrieved information from the returned `DataSet` and displays them in this page.

All of these modification steps are shown in Figure 9.42.

9.3.11.3.4 Add Three User-Defined Subroutine Procedures We need to add three user-defined subroutines: `ProcessObject()`, `FillFacultyObject()`, and `FillFacultyDataSet()`, into this project. The codes for these three subroutines 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 a little modification.

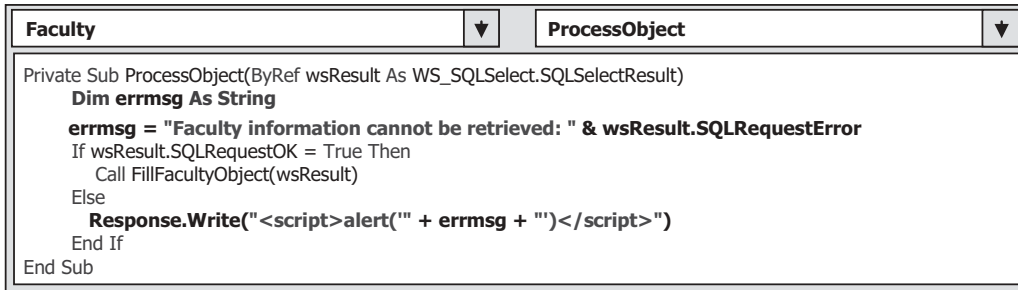


Figure 9.43. The modified codes for the subroutine ProcessObject().

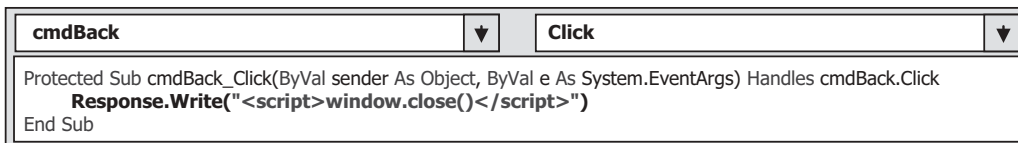


Figure 9.44. The modified codes for the Back button event procedure.

Open our Windows-based Web service client project WinClientSQLSelect, copy these three subroutines from that project, and paste them into the code page of our current Faculty page. The only modification is for the `MsgBox()` method in the subroutine `ProcessObject()`.

As you know, 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 `MsgBox()` to display an error message. Create a local string variable `errmsg` for this subroutine to hold the possible error message. Your modified codes for this subroutine should match those that are shown in Figure 9.43. The modified parts have been highlighted in bold.

There is no any modification needed for the other two subroutines `FillFacultyObject()` and `FillFacultyDataSet()`.

9.3.11.3.5 Modify the Codes for the Back Button Event Procedure The modification to the Back button's click event procedure 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 event procedure should match the one that is shown in Figure 9.44. The modified parts have been highlighted in bold.

Before we can run our project, we need to remove two unused user-defined subroutine procedures, `FillFacultyReader()` and `MapFacultyTable()`, from this project since both of them were built in the previous project, and we will not use them in this project.

Now, it is the time for us to run our Web-based Web service client project to test the functions 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 our `Faculty.aspx` page from the Solution Explorer window and select the item `Set As Start Page` from the pop-up menu.

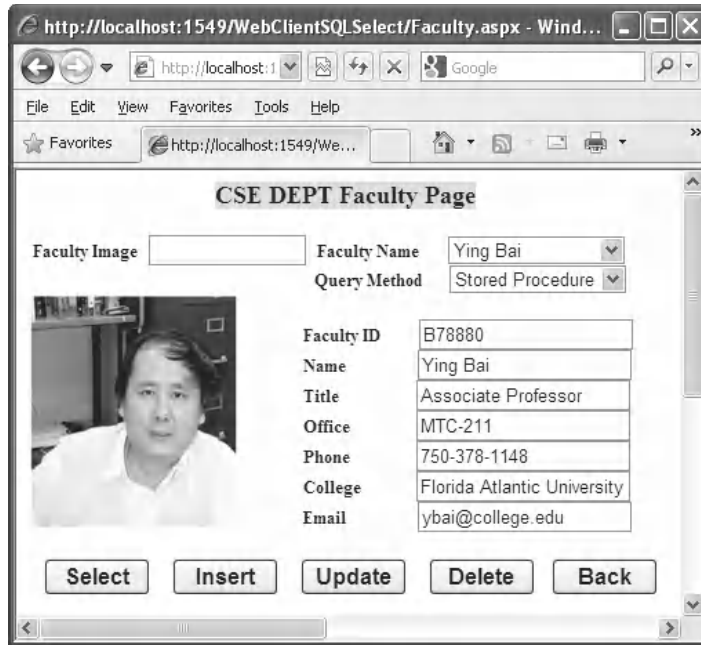


Figure 9.45. The running status of our Web-based client project.

2. Make sure that our Web service `WebServiceSQLSelect` has been run at least one time and that Web service status is running. This 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 9.45.

Keep the default Web method in the Query Method combo box control selected and the faculty name in the Faculty Name combo box control unchanged. 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 9.45.

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 in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

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 IIS 5 or higher virtual directory to allow users to access and use it via a production server. We may discuss this topic in the early section

when we finished developing our Web service project. The reason we delay this discussion until this section is that we do not have to perform this Web service deployment if we run our Web service and access it using a client project in our local computer (development server). However, we must deploy our Web service to IIS if we want to run it in a formal Web server (production server).

Basically, we have two ways to do this deployment. One way is to simply 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 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. Now 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 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 double-clicking on the **Administrative Tools** icon from the Control Panel. On the opened wizard, double-click on the icon **Computer Management**, then expand the item **Services and Applications** from the opened wizard and continue to expand the item **Internet Information Services**. Three items are listed under this icon: **We Sites**, **Default Web Site**, and **Default SMTP Virtual Server**.
3. Right-click on the second item **Default Web Site** and select the item **New/Virtual Directory** from the pop-up menu to open the Creation Wizard. Click on the **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** button to continue.
5. In the next wizard, click on the **Browse** button to find folder we created at step 1, which is `WSSQLSelect`. Then click on the **OK** button and the **Next** button to go to the next step.
6. Keep all default setting unchanged in the next wizard and click on the **Next** button 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.aspx` page in our Web service project. In order to enable the Web server to find our starting page, we need to modify the default page for this virtual directory. Follow the steps below to finish this modification:

1. Right click on our newly 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 list box by selecting them and clicking on the **Remove** button.
3. Click on the **Add** button and enter our starting page, `WebServiceSQLSelect.aspx`, into the **Default Document Name** box as our default starting page. Then click on the **OK** button.
4. Click on the **Apply** and then the **OK** button to close this window.

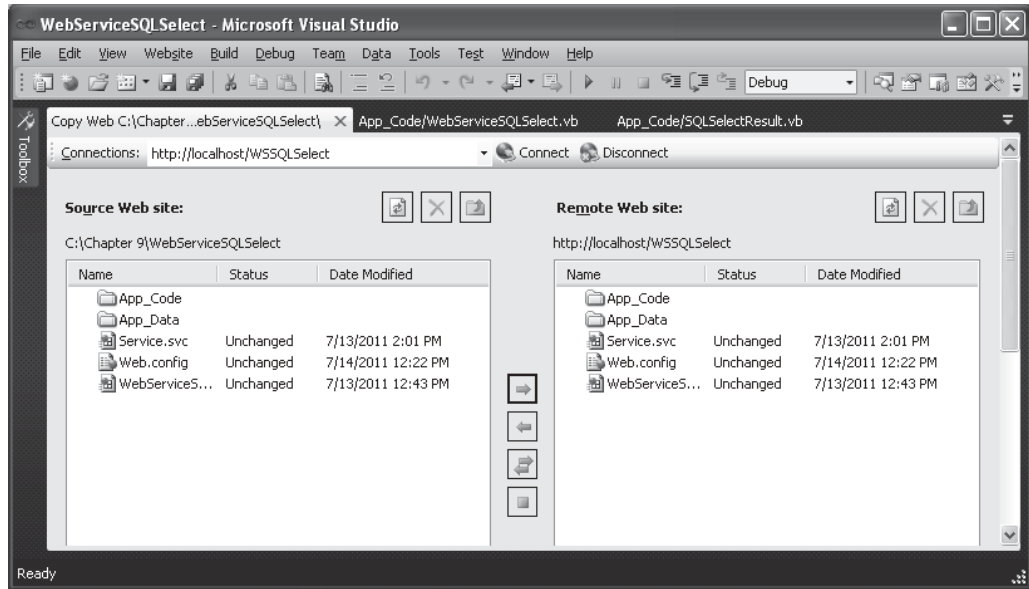


Figure 9.46. Copy Web service files to the virtual directory.

After our virtual directory is set up, next, 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 this method is relatively simple.

9.3.12.1 Copy Web Service Files to the 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 **Website|Copy Web Site** menu item to open the Copy Web Site wizard that is shown in Figure 9.46.
3. Click on the **Connect** button that is located at the right of the **Connections** text box.
4. On the opened wizard, 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 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 9.46.
6. Now go to the **File|Open Web Site** menu item to open the Open Web Site wizard. Click on the **Local IIS** icon from the left pane and select our virtual directory **WSSQLSelect**, and then click the **Open** button. Click **Yes** on the message box to allow our site to be configured for use ASP.NET 4.0 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 1–7 listed in Section 9.3.12 to create this virtual directory if you have not done it.

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 wizard. Enter the virtual directory we 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.

When 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 9.47.

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

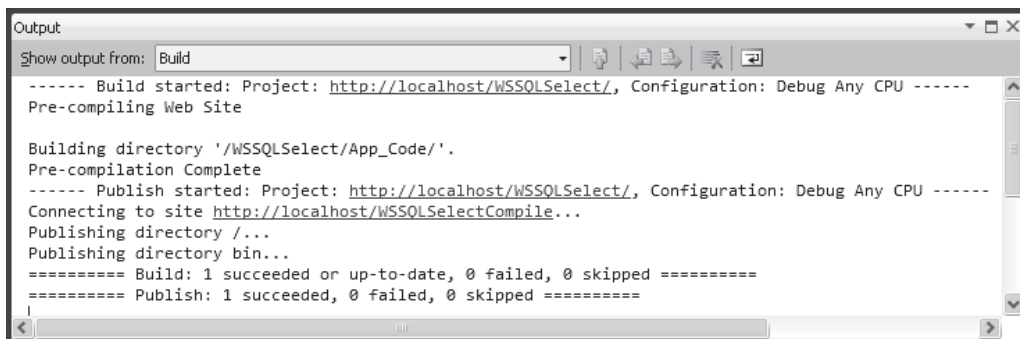


Figure 9.47. The processing result of the Web service publishing.

following sections, we will expand these discussions to perform the data insertion, updating, and deleting actions against the database through the Web services.

9.4 BUILD ASP.NET WEB SERVICE PROJECT TO INSERT DATA INTO SQL SERVER DATABASE

In this section, we will 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 record 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 to make it as our new Web service project `WebServiceSQLInsert`.

You can find the Web service application project `WebServiceSQLSelect` in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). You can copy that project and save it to a local folder at your computer.

9.4.1 Modify an Existing Web Service Project

First, let's create a new folder, such as `Chapter 9`, in our root directory using the Windows Explorer, and then copy the `WebServiceSQLSelect` project and paste it into our newly created folder `C:\Chapter 9`. Rename it to `WebServiceSQLInsert` and perform the following modifications to this project in the Windows Explorer:

1. Change our main Web service page from `WebServiceSQLSelect.aspx` to `WebServiceSQLInsert.aspx`.
2. Change the name of our base class from `SQLSelectBase`, which is located in the folder `App_Code`, to `SQLInsertBase`.
3. Change the name of our child class from `SQLSelectResult`, which is located in the folder `App_Code`, to `SQLInsertResult`.
4. Change the name of the code-behind page from `WebServiceSQLSelect.vb` to `WebServiceSQLInsert.vb`.
5. Open Visual Studio.NET and our new Web project `WebServiceSQLInsert`. Then open our entry page `WebServiceSQLInsert.aspx` by double-clicking on it, and change the compiler directive from

```
CodeBehind = "~/App_Code/WebServiceSQLSelect.vb"
```

to

```
CodeBehind = "~/App_Code/WebServiceSQLInsert.vb"
```

Also, change the class name from

```
Class = "WebServiceSQLSelect" to Class = "WebServiceSQLInsert"
```

6. Remove the child class `SQLInsertResult.vb` from this project since the data insertion has no any data to be returned.

7. Open the base class `SQLInsertBase.vb` and perform the following modifications:

Change the class name from `SQLSelectBase` to `SQLInsertBase`.

- A. Change two member data from `SQLRequestOK` to `SQLInsertOK`, and from `SQLRequestError` to `SQLInsertError`.
- B. Add the following seven member data into this class:
 - a. `Public FacultyID As String`
 - b. `Public CourseID(10) As String`
 - c. `Public Course As String`
 - d. `Public Schedule As String`
 - e. `Public Classroom As String`
 - f. `Public Credit As Integer`
 - g. `Public Enrollment As Integer`

Go to the `File/Save All` menu item to save these modifications.

Next, let's take care of creating the different Web methods to perform the course record insertion actions.

9.4.2 Develop the Web Service Methods

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 newly 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`. The functions of these methods are described 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 an associated `faculty_id`. 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 method combines those 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.

The main code developments and modifications are performed in our code-behind page `WebServiceSQLInsert.vb`. In fact, most modifications will be made on the codes in our four Web methods listed above.

9.4.3 Develop and Modify the Codes for the Code-Behind Page

Open our new project `WebServiceSQLInsert` if it has not been opened. Open the code window of our code-behind page `WebServiceSQLInsert.vb` and change our main Web service class's name from `WebServiceSQLSelect` to `WebServiceSQLInsert`.

The second modification is to remove the user-defined subroutine `FillFacultyReader()` since we do not need to return any data for this data insertion operation.

The third modification is to remove two Web methods, `GetSQLSelectSP()` and `GetSQLSelectDataSet()`, including the entire coding body, since we will not use them in this application.

The last modification to this page is to modify the codes of the user-defined subroutine `ReportError()`. Perform the following modifications to this subroutine:

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 the First Web Method `SetSQLInsertSP`

Perform the modifications shown in Figure 9.48 to the Web method `GetSQLSelect()` to get our new Web method `SetSQLInsertSP()`.

This Web method uses a stored procedure to perform the data insertion. Recall that in Section 6.8.1.1 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 again in this Web method to reduce our coding load. Refer to that section to get more detailed information about how to develop this stored procedure. Seven input parameters are used for this stored procedure, `@FacultyName`, `@CourseID`, `@Course`, `@Schedule`, `@Classroom`, `@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 the codes for this Web method to see how they work.

- A. Our Web service class name is changed to `WebServiceSQLInsert` to distinguish it from the original one.
- B. The Web method 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 a new course record to be inserted into the `Course` table. 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 6.8.1.1 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`. There is no any data supposed to be returned for the data insertion action. However, in order to enable our client project to get a clear feedback from the execution

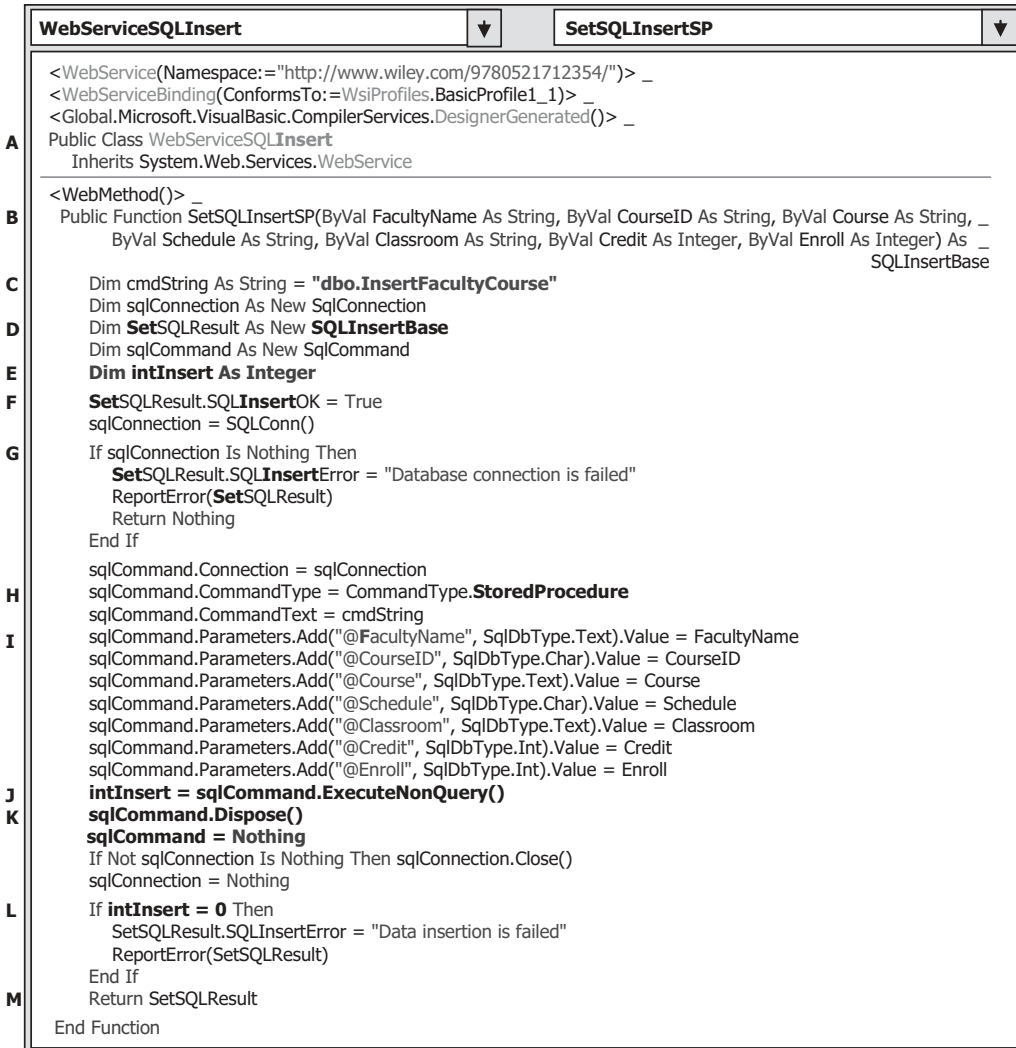


Figure 9.48. The modification to the first Web method.

of this Web service, we prefer to return an object that contains the information indicating whether this Web service is successful or not.

- E.** A local integer variable **intInsert** is declared, and this variable is used to hold the returned value from calling of the **ExecuteNonQuery()** method of the **Command** class, and that method will call 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 contained **Nothing**, 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. The user-defined subroutine `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 a new course record to be inserted into the `Course` table. One important point to be noted is that these input parameters' names must be identical with those names defined in the stored procedure `dbo.InsertFacultyCourse` developed in Section 6.8.1.1 in Chapter 6. Refer to that section to get a detailed description of those parameters' names defined in that stored procedure.
- J. The `ExecuteNonQuery()` method is executed to call 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 subroutine procedure `ReportError()`.
- 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 insert a new course record to our sample database via this Web service. Click on the Start Debugging button to run the project. The built-in Web interface is shown in Figure 9.49.

Click on the Web method `SetSQLInsertSP` to open another built-in Web interface to display the input parameters window, which is shown in Figure 9.50.

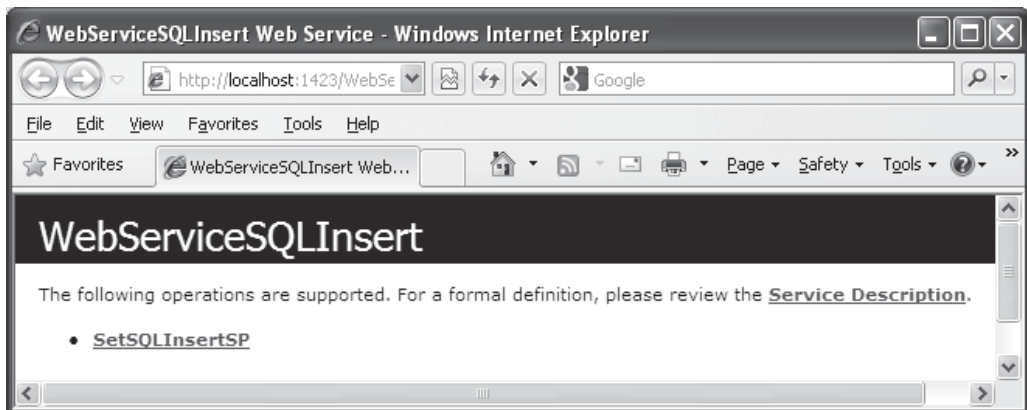


Figure 9.49. The running status of the built-in Web interface.

WebServiceSQLInsert Web Service - Windows Internet Explorer

http://localhost: Google

File Edit View Favorites Tools Help

★ Favorites WebServiceSQLInsert We...

WebServiceSQLInsert

Click [here](#) for a complete list of operations.

SetSQLInsertSP

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
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

Invoke

SOAP 1.1

Figure 9.50. The input parameter interface.

Enter the following parameters as a new course record into 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

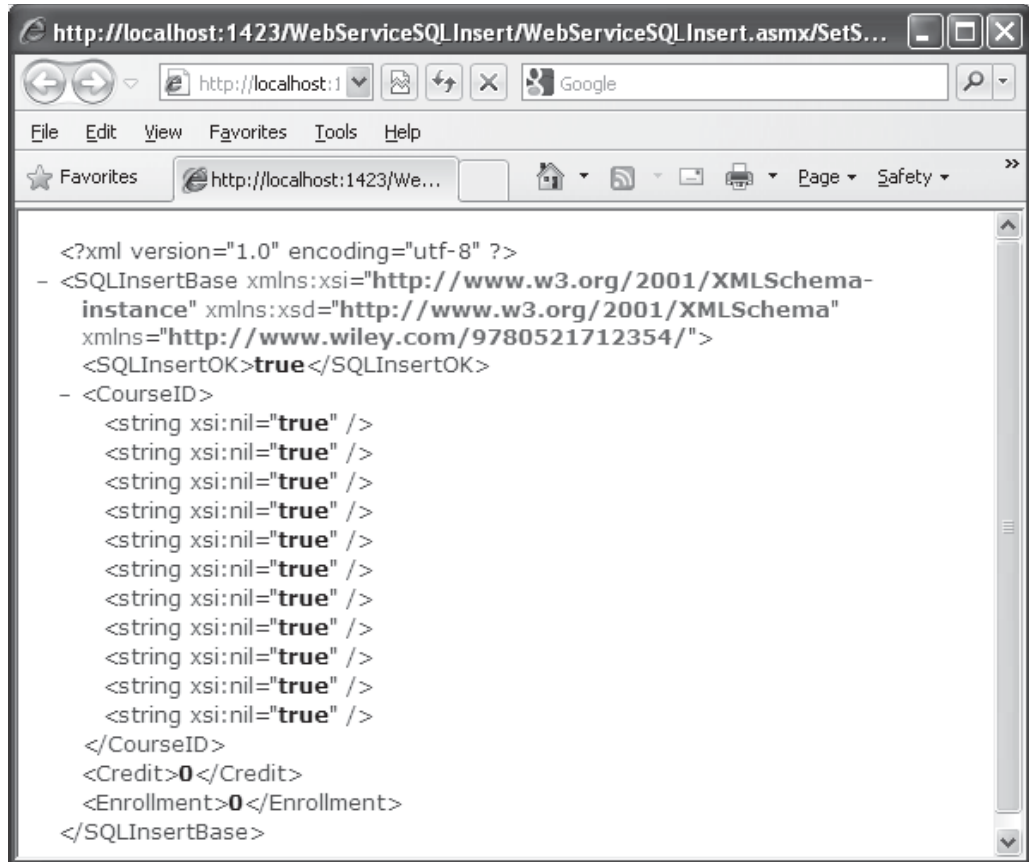


Figure 9.51. The running result of the first Web method.

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 9.51.

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 that is located at the upper-right corner of this Web interface to terminate our Web service project. Then open the `Course` table in our sample database `CSE_DEPT.mdf` using the SQL Server Management Studio to check this newly 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 have not assigned any data to them, and the default value is zero for any Integer variable. Also you can find that the `<CourseID>` tag, which is a string array `CourseID(10)` (0–10 elements) defined in our base class `SQLInsertBase.vb`, returned 11 empty or true strings. Because we only inserted a new course record into the `Course` table in our sample database, and this insertion needs to return nothing according our definition to this base class.

9.4.3.2 Develop the Second Web Method GetSQLInsert

The function of this Web method is to retrieve all `course_id`, which includes the original and the newly 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 list box control in the client project.

Recall that in Section 5.19.6 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 perform a query from the Faculty table to obtain it. In this situation, a join query is the 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.vb` and enter the codes that are shown in Figure 9.52 into this page to create our new Web method `GetSQLInsert()`.

WebServiceSQLInsert	GetSQLInsert
<pre> <WebMethod()> _ A Public Function GetSQLInsert(ByVal FacultyName As String) As SQLInsertBase B Dim cmdString As String = "SELECT Course.course_id FROM Course JOIN Faculty " + _ "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.faculty_name LIKE @fname)" C Dim sqlConnection As New SqlConnection Dim GetSQLResult As New SQLInsertBase Dim sqlCommand As New SqlCommand Dim sqlReader As SqlDataReader D GetSQLResult.SQLInsertOK = True E sqlConnection = SQLConn() If sqlConnection Is Nothing Then GetSQLResult.SQLInsertError = "Database connection is failed" ReportError(GetSQLResult) Return Nothing End If F sqlCommand.Connection = sqlConnection sqlCommand.CommandType = CommandType.Text sqlCommand.CommandText = cmdString G sqlCommand.Parameters.Add("@fname", SqlDbType.Text).Value = FacultyName H sqlReader = sqlCommand.ExecuteReader I If sqlReader.HasRows = True Then Call FillCourseReader(GetSQLResult, sqlReader) J Else GetSQLResult.SQLInsertError = "No matched course found" ReportError(GetSQLResult) End If K If Not sqlReader Is Nothing Then sqlReader.Close() sqlReader = Nothing If Not sqlConnection Is Nothing Then sqlConnection.Close() sqlConnection = Nothing sqlCommand.Dispose() L Return GetSQLResult End Function </pre>	

Figure 9.52. The codes for our second Web method `GetSQLInsert()`.

Let's have 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 an entire course record 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. But the up-to-date standard is recommended. Refer to Section 5.19.6 in Chapter 5 to get more detailed descriptions for this topic. The nominal name of the input dynamic parameter to this query is `@fname`.
- 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 with the queried course information.
- D.** Initially, we set the running status of our Web method to `ok`.
- E.** The user-defined function `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 subroutine `ReportError()` is executed to report any exception that occurred during this connection. The Web method is exited 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 `@fname` using the `Add()` 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 subroutine `FillCourseReader()`, whose detailed codes will be discussed below, 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 subroutine `ReportError()`.
- 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 subroutine `FillCourseReader()` are shown in Figure 9.53.

The function of this piece of codes is straightforward, 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`.

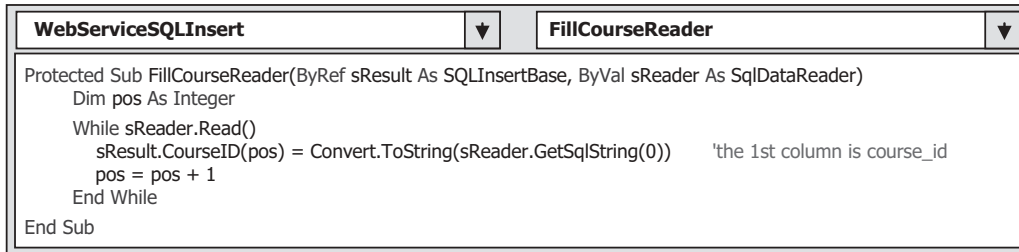


Figure 9.53. The codes for the subroutine FillCourseReader().

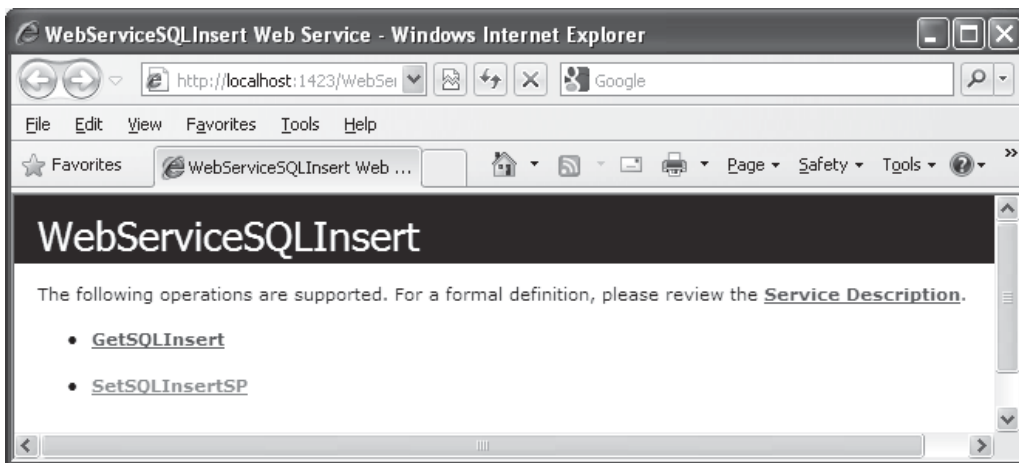


Figure 9.54. The running status of our Web service project.

Now, let's test this Web method by running this project. Click on the Start Debugging button to run our project, and the built-in Web interface is displayed, which is shown in Figure 9.54.

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 9.55.

Click on the Invoke button to execute this Web method. The running result of this method is shown in Figure 9.56.

It can be seen that all courses (exactly all course_id), including our newly inserted course CSE-556, taught by the selected faculty Ying Bai, are listed in an XML format.

Our second Web method is successful. Click on the Close button that is located at the upper-right corner of this page to terminate our Web service project. Then go to FileSave All to save all methods we have developed.

Next, let's take care of building our third Web method SQLInsertDataSet() to insert data into the Course table using the DataSet method.

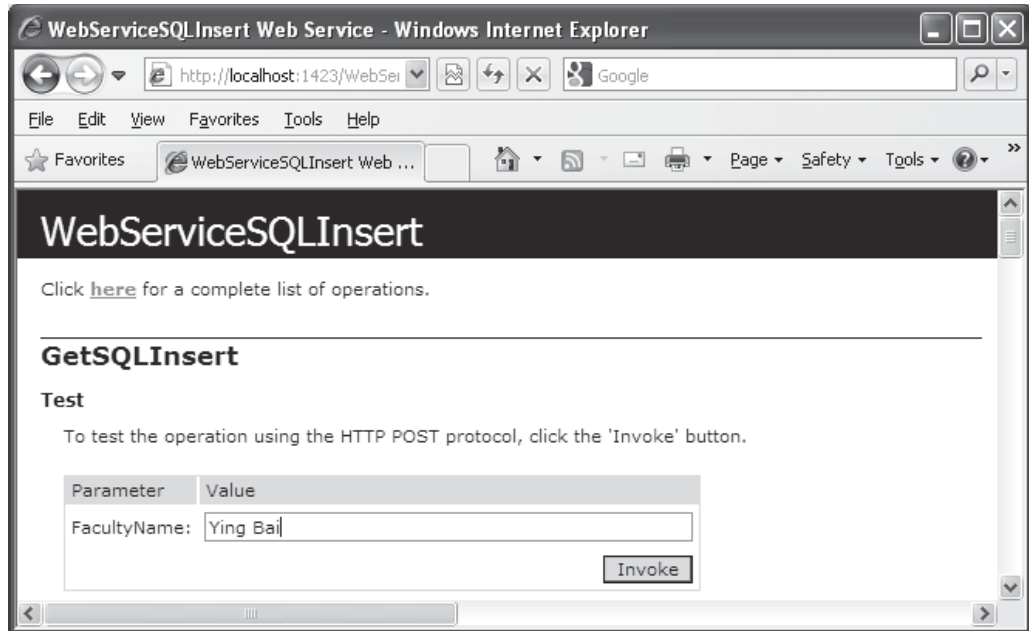


Figure 9.55. The input parameter wizard for the Web method GetSQLInsert().

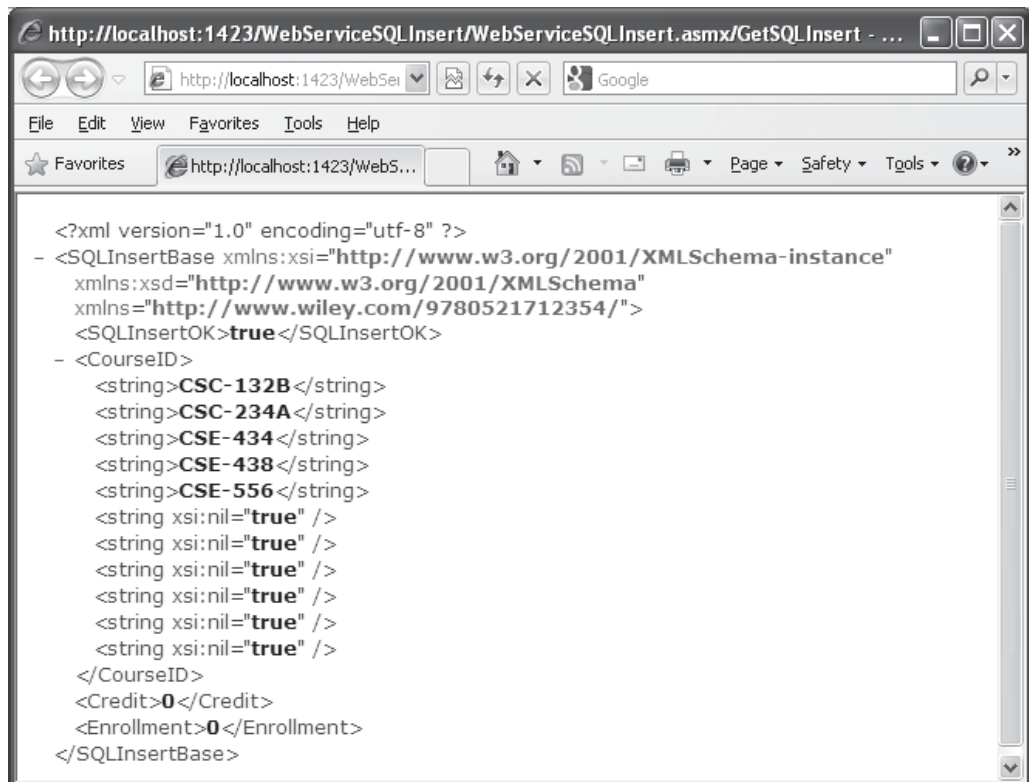


Figure 9.56. The running result of our Web method GetSQLInsert().

9.4.3.3 Develop the Third Web Method *SQLInsertDataSet*

The function of this Web method is similar to that of the first Web method, which is to insert a new course record 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. Furthermore, the returned DataSet contains the updated Course table that includes the newly inserted course record. The advantages of using a DataSet as the returned object are:

1. Unlike Web methods 1 and 2, which are a pair of methods—the first one is used to insert data into the database and the second one is used to retrieve the new inserted data from the database to confirm the data insertion—Web method 3 contains both data insertion and retrieving 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, so 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 during the project runs, we still need to create and use an instance of our base class to handle those error-processing issues.

Create a new Web method **SQLInsertDataSet()** and enter the codes that are shown in Figure 9.57 into this method.

Let's have 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 a newly inserted record, and the returned data type is DataSet.
- 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 newly 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. 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 query 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. The 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 subroutine procedure **SQLConn()** is called to perform the database connection. A warning message will be displayed and reported using the subroutine **ReportError()** if this connection encountered any error.
- F. The Command object is first initialized to perform the first query—get **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 **@fname** is assigned with the actual input parameter **FacultyName**.

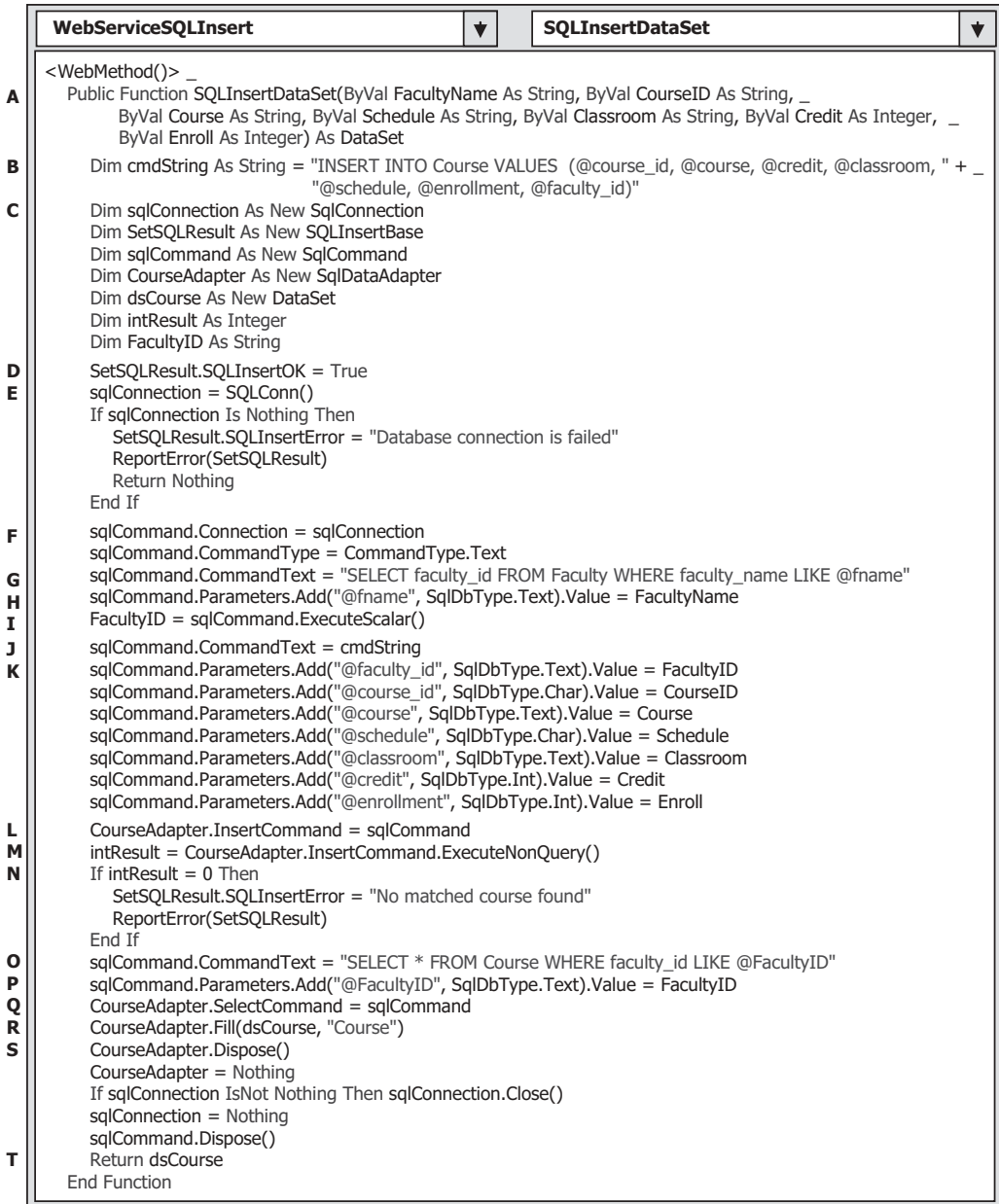


Figure 9.57. The codes for the Web method SQLInsertDataSet().

- I. The ExecuteScalar() method of the Command object is called to perform the first query to pick up the `faculty_id` and assign it 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 of this method in the normal case, but it can be automatically converted to a String type by Visual Basic.NET if it is assigned to a variable with the String type.

- J.** The second query string is assigned to the `CommandText` property to make it ready to perform the second query—insert 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 be noted is the data types of the last two parameters. Both `credit` and `enrollment` are integers, so the data type `SqlDbType.Int` is used for both of them.
- L.** The initialized `Command` object is assigned to the `InsertCommand` property of the `DataAdapter`.
- 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 subroutine `ReportError()` is called to report this error.
- O.** The third query string, which is used to retrieve all courses, including the newly inserted course, from the database based on the input `faculty_id`, is assigned to the `CommandText` property of the `Command` object.
- P.** The dynamic parameter `FacultyID` 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 newly 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 with the first Web method, it looks like that more codes are involved in this method. Yes, it is true. However, this method has two functionalities: 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 process, the third Web method has less code compared with the first one.

Now let's 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 third Web method `SQLInsertDataSet` from the built-in Web interface to start it. The parameters wizard is displayed, which is shown in Figure 9.58. Enter the following parameters into each associated `Value` box as the data of a new course:

- `FacultyName`: Ying Bai
- `CourseID`: CSE-665
- `Course`: Advanced Robotics
- `Schedule`: T-H: 1:00–2:25 PM
- `Classroom`: TC-309

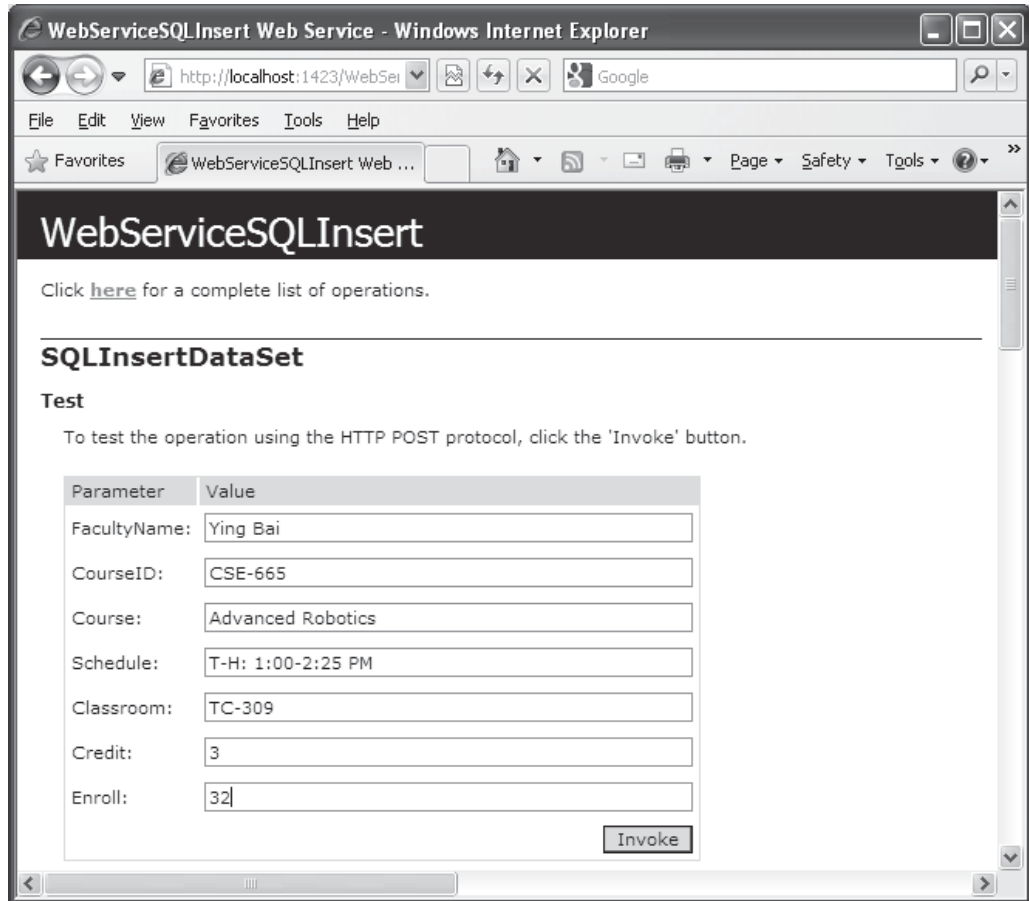


Figure 9.58. The finished input parameter wizard.

- Credit: 3
- Enroll: 32

Your finished parameter wizard should match the one that is shown in Figure 9.58.

Click on the **Invoke** button to run this Web method to perform this new course insertion. The running result is shown in Figure 9.59.

All six courses, including the sixth course **CSE-665**, which is the newly inserted course, are displayed in the XML format or tags in this running result interface.

A point to be noted is that you can only insert this new course record into the database one time, 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 that is located at the upper-right corner of this Web interface to terminate our service. A complete Web service project **WebServiceSQLInsert**

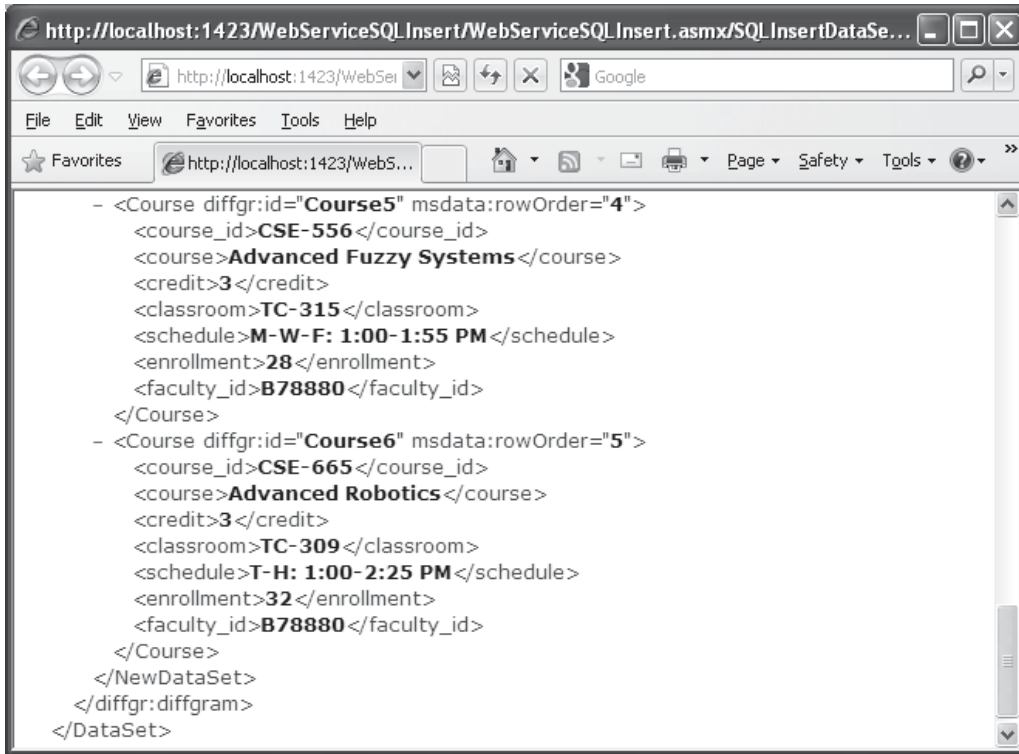


Figure 9.59. The running result of the third Web method.

can be found in the folder DBProjects\Chapter 9 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Next, let's develop our fourth Web method.

9.4.3.4 Develop the 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 consumed by a client project when users want to get detailed course information, such as the course name, schedule, classroom, credit, enrollment, and `faculty_id` when a `course_id` is selected from a list box 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 with the normal query, so we prefer to use a stored procedure to perform this query.

Now let's first create our stored procedure `WebSelectCourseSP`.

9.4.3.4.1 Create the Stored Procedure `WebSelectCourseSP` Open Visual Studio. NET 2010 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

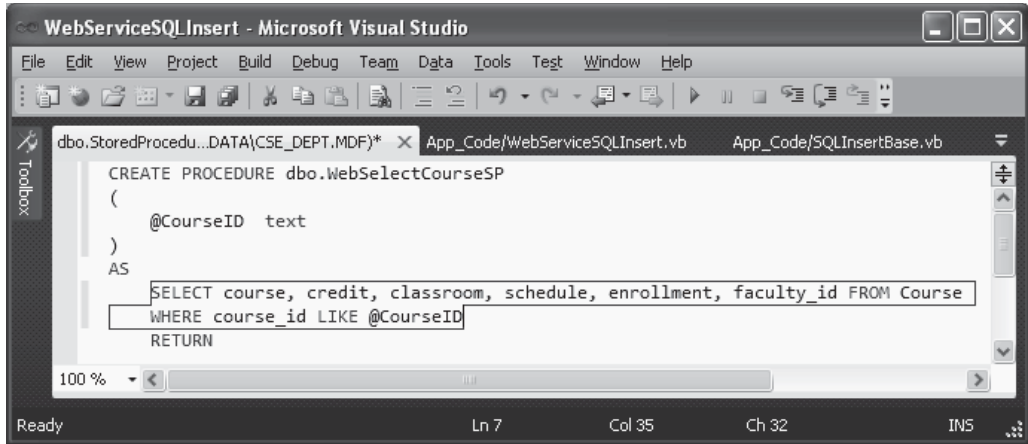


Figure 9.60. The codes for the stored procedure WebSelectCourseSP().

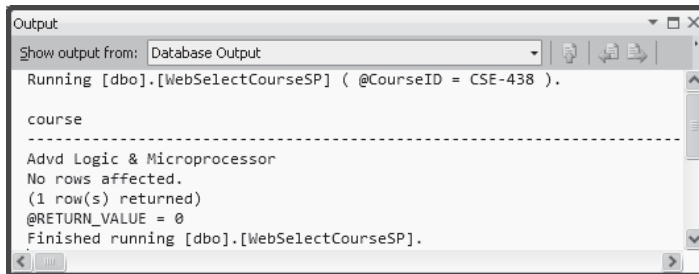


Figure 9.61. The running result of the stored procedure WebSelectCourseSP().

stored procedure, right-click on this folder and select the item Add New Stored Procedure to open the Add New Stored Procedure wizard.

Enter the codes that are shown in Figure 9.60 into this wizard to create our new stored procedure WebSelectCourseSP().

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 newly created stored procedure. Then select the item Execute from the pop-up menu to open the Run Stored Procedure wizard. Enter CSE-438 into the Value box in this wizard 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 9.61.

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 Output window 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.

9.4.3.4.2 Develop the 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.

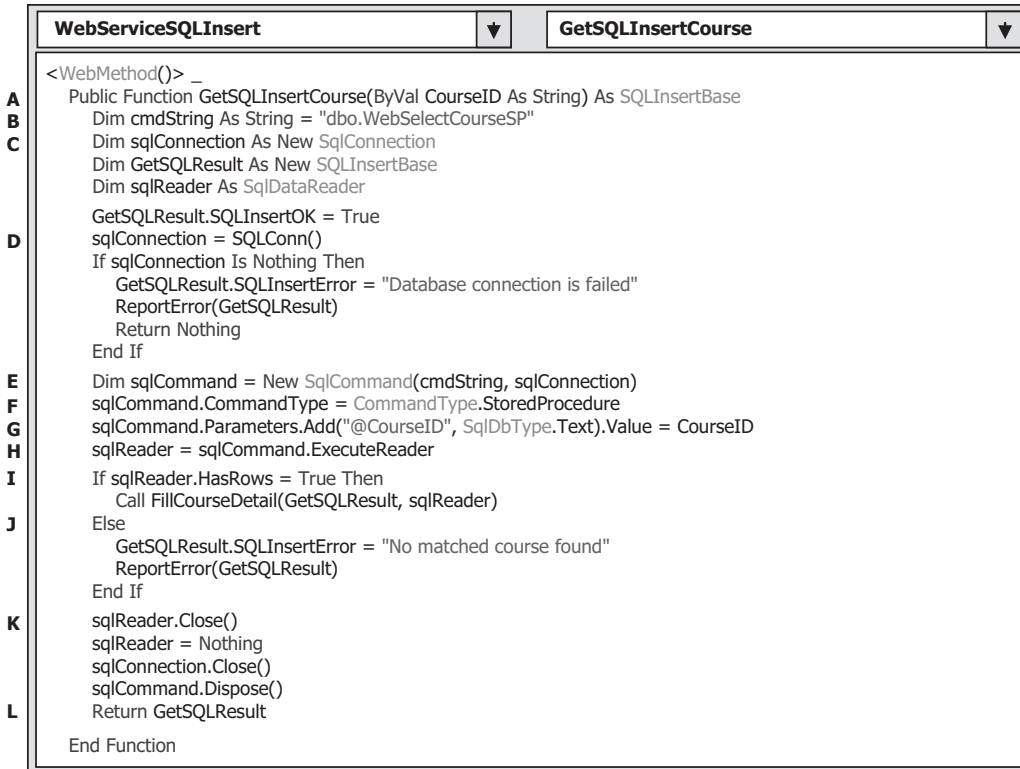


Figure 9.62. The codes for the Web method GetSQLInsertCourse().

Open the code-behind page **WebServiceSQLInsert.vb** and add the codes that are shown in Figure 9.62 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 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 during 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 subroutine **SQLConn()** is called to perform the database connection. A warning message is displayed and reported using the subroutine **ReportError()** 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 parameter by the user as the project runs. One point to be noted is that the nominal name of this dynamic parameter must be identical with the name of 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 subroutine **FillCourseDetail()**, whose detailed codes are shown in Figure 9.63, is executed to assign 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 subroutine **ReportError()**.
- 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**, which contains the queried course detailed information, is returned to the calling procedure.

The detailed codes for the subroutine **FillCourseDetail()** are shown in Figure 9.63. Let's have a closer look at this piece of codes to see how it works.

- A.** Two arguments are passed into this subroutine: the first one is our returned object that contains all member data, and the second one is the DataReader that 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 subroutine. In this way, all modified member data that contain the course information in this

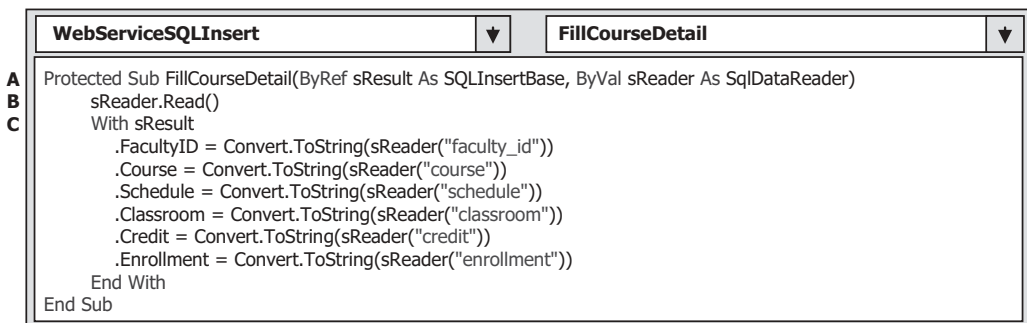


Figure 9.63. The codes for the subroutine **FillCourseDetail()**.

returned object can be returned to the calling procedure or our Web method—`GetSQLInsertCourse()`. From this point of view, this subroutine works just as a function, and our object can be returned as this subroutine is completed.

- B.** The `Read()` method of the `DataReader` is executed to read course records from the `DataReader`.
- C.** A `With . . . End With` block is executed to assign each column of queried course record to the associated member data in our base class. A `Convert.ToString()` class method is used to convert all data to strings before this assignment.

Now let's run our project to test this Web method. 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. Then click on the `Invoke` button to run this Web method. The running result is shown in Figure 9.64.

Six pieces of course information are displayed in XML tags except the `course_id`. We defined this member data as a string array with a dimension of 11. Keep in mind that the index of an array starts from 0, not 1. Therefore, the size of our array `CourseID(10)`

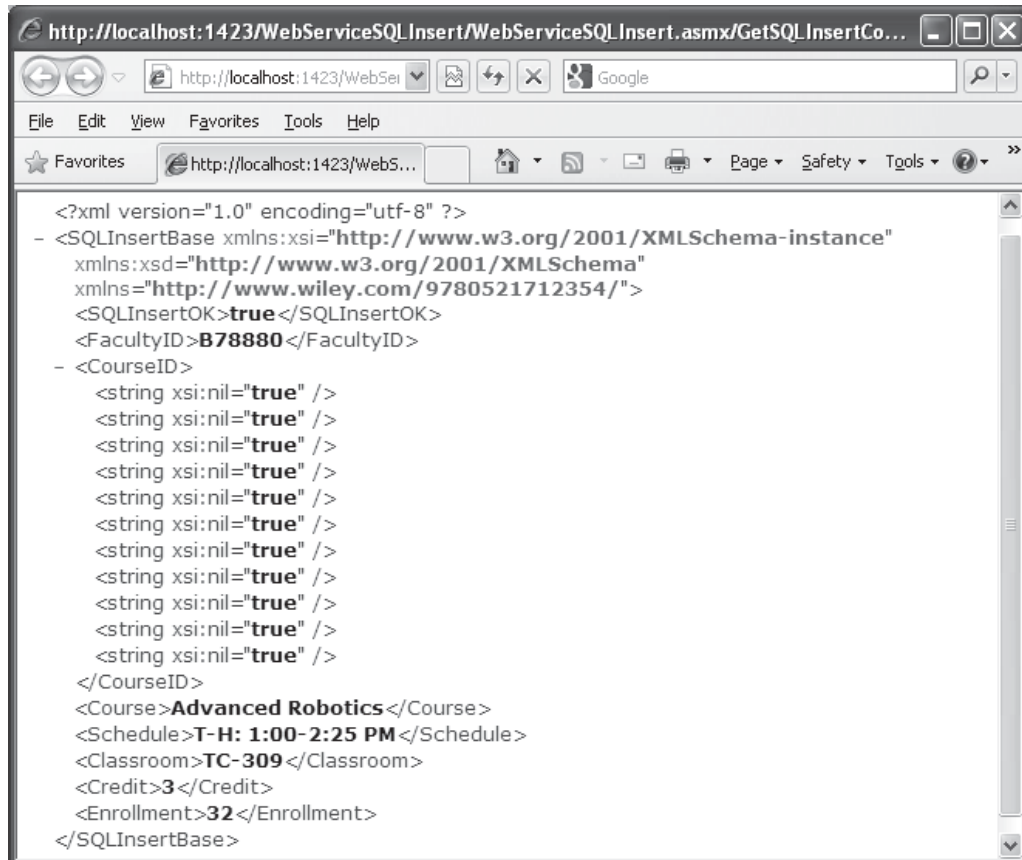


Figure 9.64. The running result of our Web method `GetSQLInsertCourse()`.

is 11. This member data is used for our second Web method, `GetSQLInsert()`, which returns an array contains all `course_id`. Since we did not use it in this method, 11 elements of this `CourseID` array are set to `true` and displayed in this resulting file.

Click on the **Close** button that is located at the upper-right corner of this Web interface to terminate our service. A complete Web service project `WebServiceSQLInsert` that contains all of those four Web methods can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

At this point, we have finished all code developing jobs in our Web service project in the server side. Next, we want to develop some professional Windows-based and Web-based application projects with beautiful GUIs to use or to consume the Web service application we developed in this Web service project `WebServiceSQLInsert`. Those Windows-based and Web-based applications can be considered as Web service clients.

9.4.4 Build Windows-Based Web Service Clients to Consume the Web Services

To use or consume 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.

9.4.4.1 Create a Windows-Based Consume Project and 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 2010 will try to find all Web services available to our applications.

To add a Web reference to our client project, we need first to create a client project. Open Visual Studio.NET 2010 and create a new Windows-based project, and name this project as `WinClientSQLInsert`.

As we mentioned in Section 9.3.10.1, there are two ways we can use to select the desired Web service and add it as a reference to our client project: (1) use the Browser provided by the Visual Studio.NET 2010 to find the desired Web service, or (2) copy and paste the desired Web service URL to the URL box located in the Add Web Reference wizard. 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 wizard 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 wizard.

Because we developed our Web service using the File System on our local computer, and, also, we have not deployed our Web service to IIS, we can use the second way to find our Web service. Perform the following operations to add this Web reference:

1. Open Visual Studio.NET 2010 and our Web service project `WebServiceSQLInsert`, and click on the Start Debugging button to run it.
2. Copy the URL from the Address bar in our running Web service project.

3. Then open another Visual Studio.NET 2010 and our Windows-based client project WinClientSQLInsert.
4. Right-click on our client project WinClientSQLInsert from the Solution Explorer window, and select the item **Add Service Reference** from the pop-up menu to open the Add Service Reference wizard.
5. Click on the **Advanced** button located at the lower-left corner on this wizard to open the Service Reference Settings wizard.
6. Click on the **Add Web Reference** button to open the Add Web Reference wizard, which is shown in Figure 9.65.
7. Paste that URL we copied from step 2 into the URL box in the Add Web Reference wizard and click on the **Green Arrow** button to enable the Visual Studio.NET 2010 to begin to search it.
8. When the Web service is found, the name of our Web service is displayed in the right pane, which is shown in Figure 9.65.
9. Alternately, you can change the name for this Web reference from `localhost` to any meaningful name, such as `WS_SQLInsert` in our case. Click on the **Add Reference** button to add this Web service as a reference to our new client project.
10. Click on the **Close** button from our Web service built-in Web interface window to terminate our Web service project.

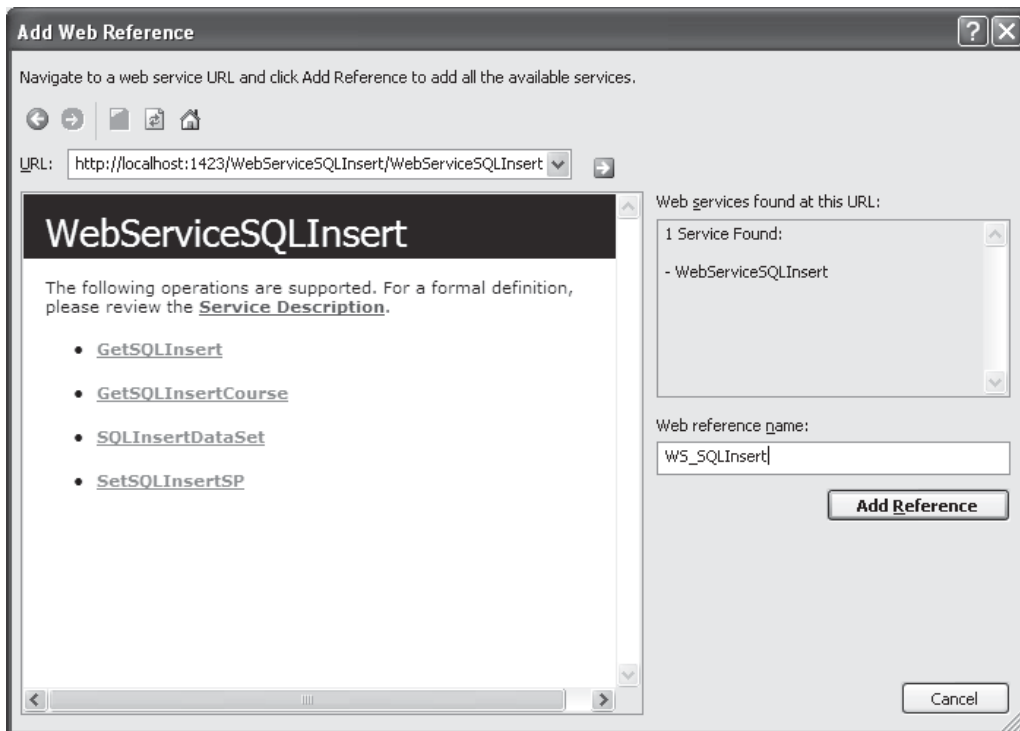


Figure 9.65. The finished Add Web Reference wizard.

Immediately, you can find that the Web service `WS_SQLInsert`, which is under the folder **Web References**, has been added into the Solution Explorer window in our client project. This reference is the so-called Web service proxy class.

Next, let's develop the GUI by adding useful controls to interface to our Web service and to display the queried course information.

9.4.4.2 *Develop the Graphic User Interface for the Client Project*

Perform the following modifications to our new client project:

1. Rename the Form File object from the default name `Form1.vb` to our desired name `WinClient Form.vb`.
2. Rename the Window Form object from the default name `Form1` to our desired name `CourseForm` by modifying the `Name` property of the form window.
3. Rename the form title from the default title `Form1` to `CSE_DEPT Course Form` by modifying the `Text` property of the form.
4. Change the `StartPosition` property of the form window to `CenterScreen`.

To save time and space, we can use the `Course Form` located in the project `SQLUpdateDeleteRTOObject` we developed in Chapter 7 as our GUI. You can find this project in the folder `DBProjects\Chapter 7` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Perform the following operations to add this `Course Form` into our new client project:

1. Open the project `SQLUpdateDeleteRTOObject` and the `Course form` window from the Wiley ftp site.
2. Select all controls from that form by going to the item `Edit|Select All`, and go to `Edit|Copy` menu item to copy all controls selected from this form window.
3. Return to our new Windows-based Web service client project `WinClientSQLInsert`, open our form window `CourseForm`, and paste those controls we copied from step 2 into this form.

Your finished GUI is shown in Figure 9.66.

The purpose of the `Query Method` combo box control is to select two different methods developed in our Web service project to get our desired course information:

1. **Stored Procedure Method** that uses a stored procedure to insert a new course record into the database.
2. **DataSet Method** that uses three queries to insert a new course record into the database and return a `DataSet` that contains the detailed course information.

The `Faculty Name` combo box control is used to select a desired faculty member as the input parameter to the Web methods to insert and pick up the desired course record.

In this application, only the **Insert**, **Select**, and **Back** buttons are used. The **Insert** button is used to trigger a data insertion action, the **Select** button is to trigger a data validation action to confirm that data insertion, and the **Back** button is used to terminate our project.

Figure 9.66. The finished graphic user interface.

The detailed functions of this project are:

1. **Insert Data Using the Stored Procedure Method:** When the project runs, as this method and a faculty name have been selected, and a new course record that is stored in six text-boxes have been entered. Then, when the **Insert** button is clicked by the user, our client project will be connected to our Web service via the Web reference we provided, and call the selected Web method `SetSQLInsertSP()` to run the stored procedure to insert that new course record into our sample database.
2. **Insert Data Using the DataSet Method:** If this method is selected, the Web method `SQLInsertDataSet()` developed in our Web service will be called to execute two queries to perform this new course insertion. Also, all courses, which include the newly inserted course, taught by the selected faculty that works as an input to this method, will be retrieved and stored into a `DataSet` by another query, and that `DataSet` will be returned to our client project.
3. **Validate Data Insertion Using the Stored Procedure Method:** To confirm this data insertion, the **Select** button, that is the **Select** button's click event procedure we will develop below, is used to validate that data insertion. If the **Stored Procedure Method** is selected, the Web method `GetSQLInsert()` is called to perform a joined-table query to retrieve all `course_id`, which include the newly inserted `course_id`, from the database, and stored them into an instance of our base class `SQLInsertBase` in our Web service. This instance will be returned to our client project, and all `course_id` stored in that instance will be taken out and displayed in the list box control `CourseList` in our client form window.
4. **Validate Data Insertion Using the DataSet Method:** If this method is selected and the **Select** button is clicked, the **Select** button's click event procedure we will develop below is executed to pick up all `course_id` from a `DataSet` that is returned in step 2. Also all `course_id` will be displayed in the list box control `CourseList` in our client form window.

5. **Get Detailed Course Information for a Specific Course:** When this method is selected and a `course_id` in the list box control `CourseList` is clicked, the Web method `GetSQLInsertCourse()` in our Web service will be called to run a stored procedure to retrieve all six pieces of information related to that selected `course_id` and store them into an instance of our base class `SQLInsertBase` in our Web service. This instance will be returned to our client project, and all six pieces of course information stored in that instance will be taken out and displayed in six textbox controls in our client form window.

Now let's take care of the coding development for this project to connect to our Web service using the Web reference we developed in the last section to call the associated Web methods to perform the different data actions.

9.4.4.3 *Develop the Code to Consume the Web Service*

The coding development can be divided into the following four parts:

1. Initialize and terminate the client project.
2. Insert a new course record into the database using both methods.
3. Validate the data insertion using both methods.
4. Get the detailed information for a specific course using both methods.

Now let's start our coding process based on these four steps.

9.4.4.3.1 Develop the Codes to Initialize and Terminate the Client Project This coding process includes the development codes for the `Form_Load` event procedure, the `Back` button's click event procedure, and some other initializations, such as the `Imports` commands and form-level variables.

Open Visual Studio.NET 2010 and our client project `WinClientSQLInsert` if it has not been opened. Then open the Code Window of this client project by clicking on the `View Code` button from the Solution Explorer window and enter the codes that are shown in Figure 9.67 into this Code Window.

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

- A. Two namespaces related to all data components and SQL Server Data Providers are imported since we need to use them later.
- B. Three form-level variables are created here. The first one is a Boolean variable `dsFlag`, and it is used to set a flag to indicate whether the `SQLInsertDataSet` Web method has been executed or not. Because this Web method performs both data insertion and data retrieving, it must be called once from the `Insert` button's click event procedure before you can perform the data retrieving from the `Select` button's click event procedure. The second is a `DataSet` object, since we need to use this `DataSet` in multiple event procedures and multiple processes in this project, such as the data insertion and the data validation processes later. The third one is an instance of the base class `SQLInsertBase` developed in our Web service project, and this instance is used to receive the returned instance from calling the first Web method `SetSQLInsertSP()` when performing a data insertion.
- C. In the `Form_Load` event procedure, eight default faculty members are added into the Faculty Name combo box control using the `Add()` method. These faculty members will be displayed and selected by the user as the input parameter to call different Web methods

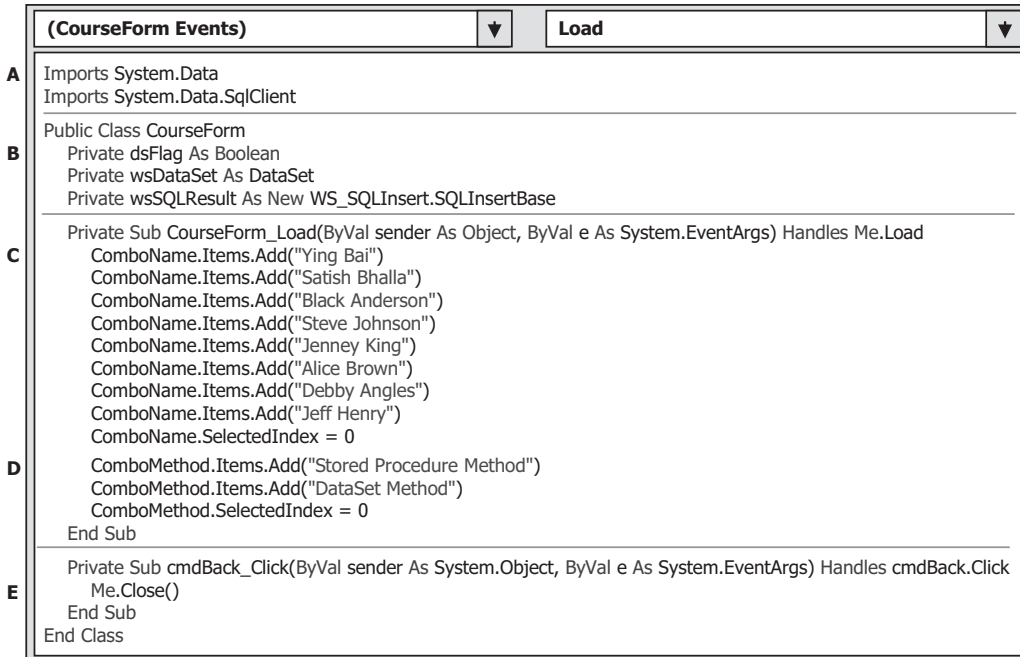


Figure 9.67. The codes of the Form_Load and Back button event procedures.

to perform either a data insertion or data validation operation as the project runs. The first faculty member is selected as the default one by setting the SelectedIndex property to zero.

- D.** Two Web methods, **Stored Procedure Method** and **DataSet Method**, are added into the Query Method combo box control, and these methods can be selected by the user to call the associated Web method to perform the desired data operation as the project runs. Similarly, the first method, the **Stored Procedure Method**, is selected as the default one.
- E.** The codes for the **Back** button's click event procedure are very simple. The **Close()** method is called to terminate our client project.

The first coding job is done, and let's continue to perform the next coding process.

9.4.4.3.2 Develop the Codes to Insert a New Course Record into the Database

This coding development can be divided into two parts based on two methods: the **Stored Procedure Method** and the **DataSet Method**. Because of the similarity between the codes in these two methods, we combine them together.

To insert a new course record into the database via our Web service, the following three jobs should have been completed before the **Insert** button can be clicked:

1. The Web method has been selected from the Query Method combo box control.
2. The faculty name has been selected from the Faculty Name combo box control.
3. Six textboxes have been filled with six pieces of information related to a new course to be inserted.

Besides those conditions, one more important requirement for this data insertion is that any new course record can only be inserted into the database once. In other words, no duplicated record can be inserted into the database. This duplication can be identified by checking the content of the textbox **Course ID**, or the column `course_id` in the **Course** table in the database. As you know, the `course_id` is the primary key in the **Course** table, and each record is identified by using this primary key. As long as the `course_id` is different, no duplication could occur. Based on this analysis, in order to avoid the duplicated insertion from occurring, the **Insert** button should be disabled after a new course record is inserted into the database, and this button should be kept disabled until a different or a new `course_id` is entered into the **Course ID** textbox, which means that a new record is ready to be inserted into the database.

Keep this in mind, and now let's start to develop the codes for the **Insert** button's click event procedure.

Double-click on the **Insert** button from the Design View of our client project to open the **Insert** button's click event procedure. Then enter the codes that are shown in Figure 9.68 into this event procedure.

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

- A.** An instance of the Web reference to our Web service or our proxy class is created here since we need it to access our Web methods to perform different data actions later. This instance works as a bridge between our client project and Web methods developed in our Web service project.
- B.** If users selected the **Stored Procedure Method** to perform the data insertion, a **Try . . . Catch** block is used to call the Web method `SetSQLInsertSP()` with seven pieces of new course information as arguments to insert a new course record into the database.

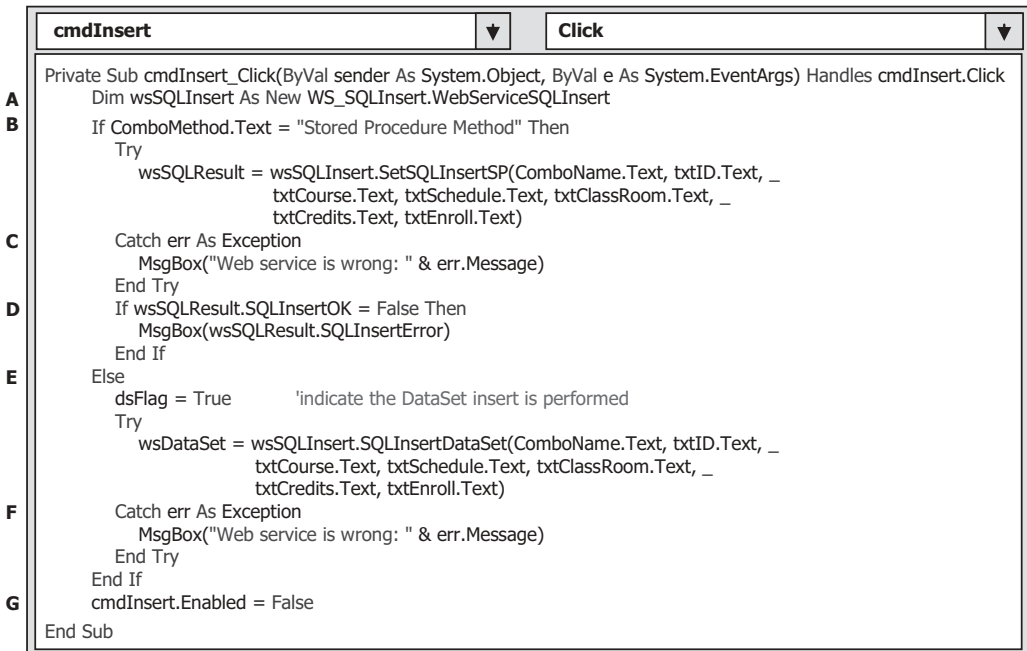


Figure 9.68. The codes for the **Insert** button Click event procedure.

The calling result is returned and assigned to our form-level variable `wsSQLResult` that will be checked later.

- C. If any error is encountered, the error message is displayed.
- D. Besides those errors-checking performed by the **Catch** statement, we also need to check the member data defined in our base class to make sure that the running status of our Web method is fine. One of member data `SQLInsertOK` is used to store this running status. If this status is **False**, which means that something is wrong during the execution of this Web method, the error message is displayed using another member data `SQLInsertError` that stored the error source.
- E. If users selected the **DataSet Method**, first, the Boolean variable `dsFlag` is set to **True** to indicate that the Web method `SQLInsertDataSet()` has been executed once. This flag should be reset to **False** if users want to retrieve the course information from the database by clicking on the **Select** button, but they have not called this Web method to first insert a new course record. If this happened, a message is displayed to direct the users to first execute this Web method to insert a new record into the database. Another **Try . . . Catch** block is used to call the Web method `SQLInsertDataSet()` with seven pieces of new course information as arguments to insert a new course record into the database. In addition to performing the new course insertion, this Web method also performed a data query to retrieve all courses, including the newly inserted course, from the database and assign them to the `DataSet` that is returned to our client project.
- F. If any system error is detected by the **Catch** statement, the error message is displayed.
- G. Finally, the **Insert** button is disabled to avoid multiple insertions of the same record into the database.

Another coding development is for the **Course ID** textbox, that is, to the `TextChanged` event procedure of the **Course ID** textbox. As we mentioned, the **Insert** button should be disabled after one new course record has been inserted into the database to avoid the multi-insertion of the same data. However, this button should be enabled when a new different course record is ready to be inserted into the database. As soon as the content of the **Course ID** textbox changed, which means that a new record is ready, the **Insert** button should be enabled. To do this coding, double-click on the textbox **Course ID** from the Design View of our client project window to open its `TextChanged` event procedure. Enter the following codes into this event procedure:

```
cmdInsert.Enabled = True
```

At this point, we have finished all coding developments for the data insertion process. Before we can continue to develop the rest of our project, we prefer to first run the client project to test this data insertion functionality.

The prerequisite to run our client project is to make sure that our Web service is in the running status in this local computer. To check and confirm that, open our Web service project `WebServiceSQLInsert` and click on the **Start Debugging** button to run it. Then you can close our Web service page by clicking on the **Close** button (our Web service is still in the running status even the page is closed).

Now you should find that a small white icon has been added into the status bar on the bottom of the screen. This small white icon means that our Web service is in the running status, and any client can access and use it now. The reason we closed our Web service page is that we do not need to keep our Web service page in an opening status, instead, we need it in the background running status. After our Web service project runs

one time, it will be in the running status, that is. it is in the background running status even the Web page is closed.

Now run our Windows-based client project **WinClientSQLInsert** by clicking on the **Start Debugging** button. As the **CourseForm** window is displayed, perform the following two insertions by using two Web methods with the following operations and parameters:

1. Insert the first new course record that is shown in Table 9.2 using the **Stored Procedure Method**. Click on the **Insert** button to finish this data insertion.
2. Insert the second new course record that is shown in Table 9.3 using the **DataSet Method**. Click on the **Insert** button to finish this data insertion.

Table 9.2. The first course record to be inserted

Controls	Input Parameters
Method:	Stored Procedure Method
Faculty Name:	Ying Bai
Course ID:	CSE-665
Course Name:	Advanced Fuzzy Systems
Schedule:	T-H: 1:00-2:25 PM
Classroom:	TC-315
Credits:	3
Enrollment:	26

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

Now click on the **Back** button to terminate our client project. To confirm these two data insertions, open the Microsoft SQL Server Management Studio or Studio Express. Then open our sample database **CSE_DEPT.mdf** and our **Course** table. You can find that these two records have been added into our **Course** table in the last two rows. It is highly recommended to delete these two new records from our **Course** table after this checking since we will perform the same data insertions when we confirm these data insertions programmably in the following section.

9.4.4.3.3 Develop the Codes to Perform the Inserted Data Validation To confirm or validate the data insertion, we can open our database and data table to check it. However, a professional way to do this confirmation is to use codes to perform this validation. In this section, we discuss how to perform this validation by developing the codes in the **Select** button's click event procedure in our client project.

As we mentioned in the previous sections, as this **Select** button is clicked after a new course insertion, all **course_id**, including the newly inserted **course_id**, will be retrieved from the database and displayed in a list box control in this **CourseForm** window. This data validation is also divided into two parts according to the method adopted by the user: either the **Stored Procedure Method** or the **DataSet Method**. Different processes will be performed based on these two methods. Because of the codes similarity between these two methods, we combine these codes together and put them into this **Select** button's click event procedure.

Now double-click on the **Select** button from the Design View of our client project **WinClientSQLInsert** to open this event procedure and enter the codes that are shown in Figure 9.69 into this event procedure.

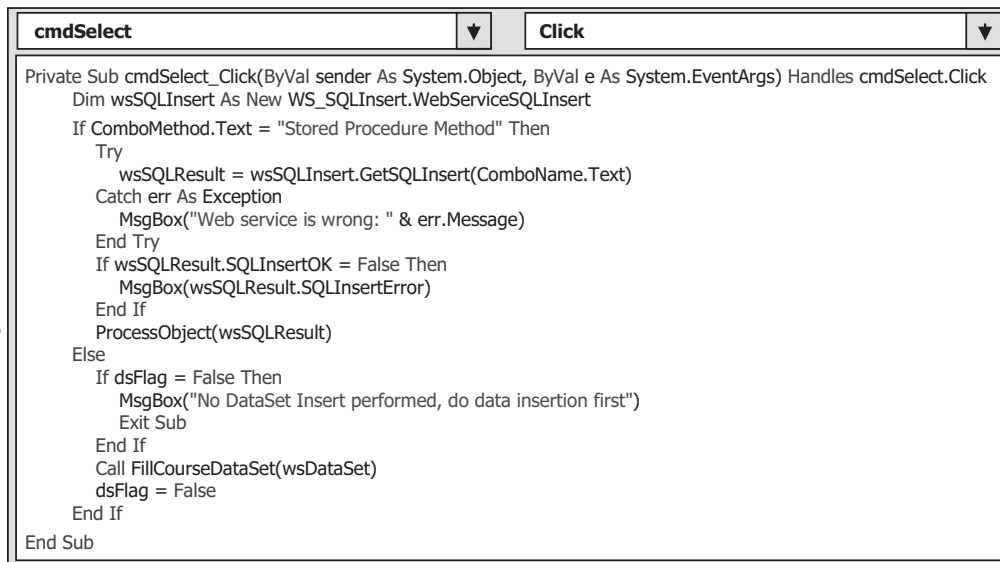


Figure 9.69. The codes for the **Select** button **Click** event procedure.

Let's take a closer look at this piece of codes to see how it works.

- A. An instance of our Web service reference or proxy class is created, and this instance works as a bridge to connect our client project with the Web methods developed in our Web service together.
- B. If the **Stored Procedure Method** has been selected by the user, a **Try . . . Catch** block is used to call our Web method `GetSQLInsert()` with the selected faculty name as the input to retrieve all `course_id` from the database. This method returns an instance of our base class defined in the Web service, and this instance, which contains all `course_id` retrieved from the database, is assigned to our form-level variable `wsSQLResult` to be processed later. An error message is displayed if any error were encountered during the execution of this Web method.
- C. In addition to the error checking performed by the system in the **Catch** statement, we also need to perform our error-checking process by inspecting the status of the member data `SQLInsertOK`. The error source will be displayed if any error occurred.
- D. If this Web method works fine, a user-defined subroutine `ProcessObject()`, whose detailed codes are shown in Figure 9.70, is called to extract all course columns from that returned instance `wsSQLResult`.
- E. If the user selected the **DataSet Method**, first we need to check whether the Web method `SQLInsertDataSet()` has been executed or not by checking the status of the form-level variable `dsFlag`. Because when users use this method to retrieve the course information from the database, this method must have been executed once from the **Insert** button's click event procedure. The reason for that is because this method performs both data insertion and data retrieving. An error may be encountered if you use this method to retrieve the course information from the **Select** button's click event procedure without first performing the data insertion from the **Insert** button's click event procedure since nothing has been inserted. Therefore, nothing can be obtained from the returned `DataSet`. If this `dsFlag` is `False`, which means that nothing has been inserted, an information message is displayed to ask you to first perform the data insertion.

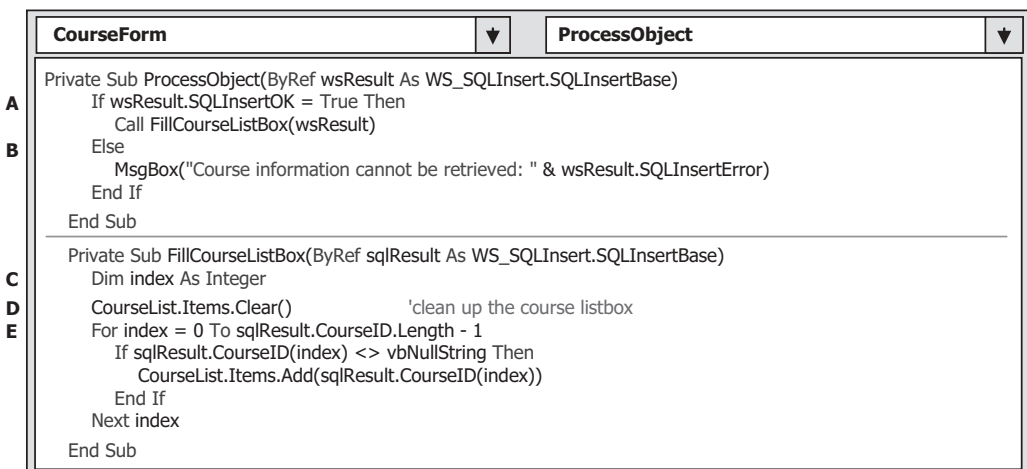


Figure 9.70. The codes for the subroutines `ProcessObject()` and `FillCourseListBox`.

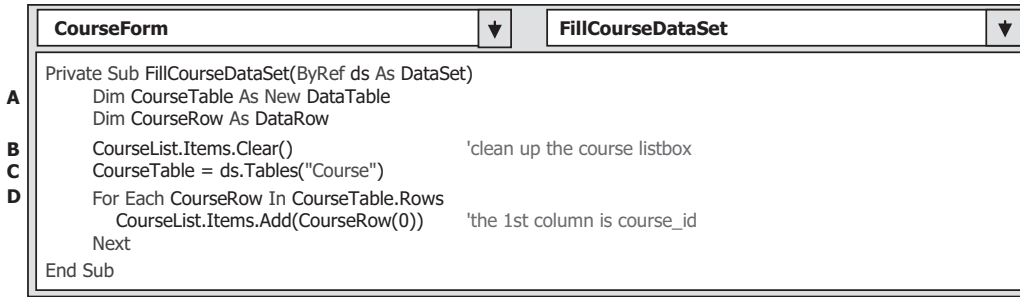


Figure 9.71. The codes for the subroutine FillCourseDataSet().

F. If the Web method `SQLInsertDataSet()` has been executed, a user-defined subroutine `FillCourseDataSet()`, whose detailed codes are shown in Figure 9.71, is called to fill the list box control with all retrieved `course_id`.

G. Finally, the `dsFlag` is reset to `False`.

The detailed codes for the subroutines `ProcessObject()` and `FillCourseListBox()` are shown in Figure 9.70.

Let's have a look at the codes in these two subroutines to see how they work.

- A.** First, we need to check the member data `SQLInsertOK` to make sure that the Web method is executed successfully. If it is, the subroutine `FillCourseListBox()` is called to fill all `course_id` contained in the returned instance to the list box control in our client form.
- B.** A warning message is displayed if any error was encountered during the execution of that Web method.
- C.** In the subroutine `FillCourseListBox()`, first, a local integer variable `index` is created, and it works as a loop number for a `For` loop to continuously pick up all `course_id` from the returned instance and add them into the list box control.
- D.** The course list box control is cleaned up first before any `course_id` can be added into it. This process is very important in displaying all `course_id`, 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.
- E.** A `For` loop is used to continuously pick up the `course_id` from the `CourseID()` array defined in our base class `SQLInsertBase`. One point to be noted is the upper bound and the length of this array. The length or the size of this array is 11, but the upper bound of this array is 10, 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, the remaining `course_id` is added into the list box control by using the `Add()` method.

The codes for the subroutine `FillCourseDataSet()` is shown in Figure 9.71.

Let's have a look at the codes in this subroutine to see how they work.

- A.** Two objects, a `DataTable` and a `DataRow`, are declared at the beginning of this subroutine since we need to use them to perform the data extraction from the returned instance and the data addition to the list box control.
- B.** The list box control is first cleaned up to avoid a messy displaying of multiple `course_id`.

- C. The `CourseTable` object is initialized by adding a new data table named “Course” and is assigned to the `DataSet` object `ds`.
- D. A `For Each . . . In` 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 noted is that the first column has an index value of 0, not 1, since the index starts from 0.

At this point, we finished all coding process for the **Select** button’s click event procedure. In other words, all codes related to the data validation are done.

Now let’s run our client project to perform the data validation after the data insertion process. Before we can start to run the project, make sure that the following two conditions are met:

1. Our Web service is in the running status, and this can be checked by locating a small white icon on the status bar on the bottom of the screen. If you cannot find this icon, open our Web service project `WebServiceSQLInsert` and click on the **Start Debugging** button to run it. After the Web service starts to run, you can close the Web page if you like, but it is still in the running status.
2. Two new course records, which we inserted before by testing the **Insert** button’s click event procedure, have been deleted from the `Course` table in our sample database since we want to insert the same course records in the following test.

Now click on the **Start Debugging** button to run our client project. Enter the same input parameters as shown in Table 9.2 in Section 9.4.4.3.2, and click on the **Insert** button to finish this data insertion using the **Stored Procedure Method**. Next, enter the same input parameters as shown in Table 9.3 in Section 9.4.4.3.2, and click on the **Insert** button to finish this data insertion using the **DataSet Method**.

To check or validate these data insertions, make sure that the selected method in the **Query Method** combo box is still the **DataSet Method** and the **Faculty Name** is **Ying Bai**. Then click on the **Select** button to retrieve all `course_id` from the database. It can be found that all six courses taught by the selected faculty are listed in the list box control with the `course_id` as the identifier for each course.

To test the **Stored Procedure Method**, make sure that the **Stored Procedure Method** is selected from the **Query Method** combo box. Now we can select another faculty from the **Faculty Name** combo box control, and click on the **Select** button to pick up all `course_id` taught by the selected faculty. Next, reselect the default **Faculty Name** **Ying Bai**, and then click on the **Select** button to try to retrieve all `course_id` taught by the selected faculty. You can find that the same results as we obtained using the **DataSet Method** are displayed in the list box control.

The running result or the data validation is shown in Figure 9.72. It can be found that our newly inserted two courses **CSE-665** and **CSE-526** have been added and displayed in the list box control, and our data insertion is successful. Click on the **Back** button to terminate our project.

Next, let’s concentrate on the coding development to display the detailed course information for a selected `course_id` from the list box control.

9.4.4.3.4 Develop the Codes to Get the Details for a Specific Course The function of this piece of codes is that the detailed course information, such as the course name, schedule, classroom, credit, and enrollment, will be displayed in the associated textbox

The screenshot shows a web application window titled "CSE DEPT Course Form". It contains two main sections: "Faculty Name _Query Method" and "Course List / Course Information".

Faculty Name _Query Method:

- Faculty Name:** A dropdown menu showing "Ying Bai".
- Query Method:** A dropdown menu showing "Stored Procedure Method".

Course List: A list box containing the following course IDs: CSC-132B, CSC-234A, CSE-434, CSE-438, CSE-526, and CSE-665.

Course Information: A form with the following fields:

- Course ID:** CSE-526
- Course:** Embedded Microcontrollers
- Schedule:** M-W-F: 9:00-9:55 AM
- Classroom:** TC-308
- Credits:** 3
- Enrollment:** 32

At the bottom of the form, there are five buttons: **Select**, **Insert**, **Update**, **Delete**, and **Back**.

Figure 9.72. The running result of the data validation.

control as the user clicked and selected one `course_id` from the list box control. The main coding job is performed inside the `SelectedIndexChanged` event procedure of the list box control `CourseList`. Because when user clicks or selects a `course_id` from the list box control, a `SelectedIndexChanged` event is issued, and this event is passed to the associated `SelectedIndexChanged` event procedure.

To pick up the detailed course information for the selected `course_id`, the Web method `GetSQLInsertCourse()` in our Web service project `WebServiceSQLInsert` is called. This method returns an instance of the base class `SQLInsertBase` to our client project. The detailed course information is stored in that returned instance.

Double-click on the list box control `CourseList` from the Design View of our client project window to open the `SelectedIndexChanged` event procedure of the list box control, and enter the codes that are shown in Figure 9.73 into this event procedure.

Let's take a closer look at this piece of codes 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 the Web service project.
- B.** A `Try . . . Catch` block is used to call the Web method `GetSQLInsertCourse()` with the selected `course_id` from the list box control as the argument to perform this course information retrieving. The selected `course_id` is stored in the `Text` property of the `CourseList` control.
- C.** An exception message is displayed if any error was encountered during the execution of this Web method and caught by the system method `Catch`.
- D.** 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 occurred

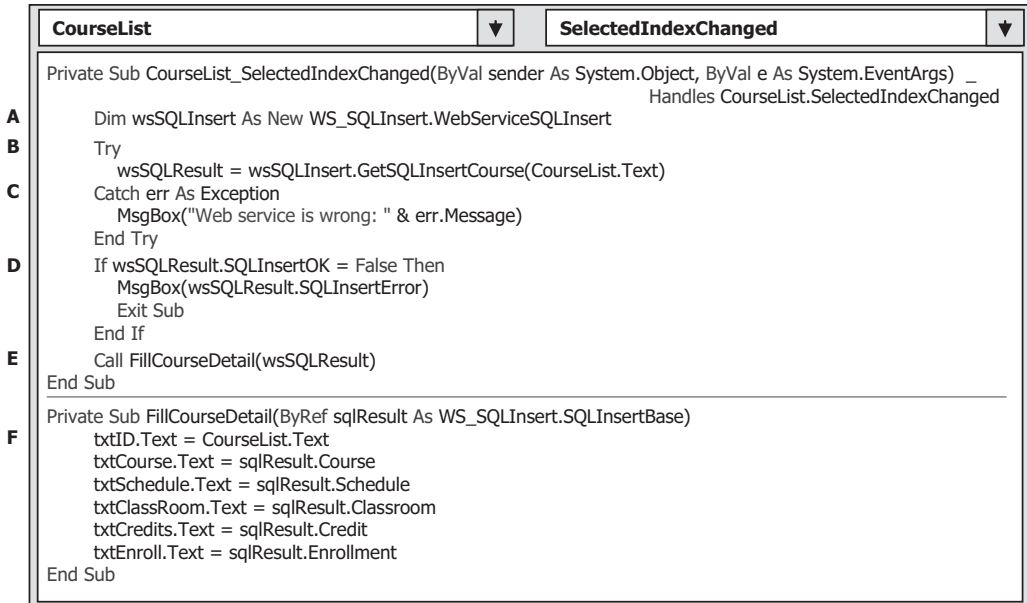


Figure 9.73. The codes for the SelectedIndexChanged event procedure.

during the running of this Web method, an error message is displayed, and the subroutine is exited.

- E.** If everything is fine, a user-defined subroutine `FillCourseDetail()` is executed to extract the detailed course information from the returned instance and assign it to each associated textbox control in our client window form.
- F.** The codes for the subroutine `FillCourseDetail()` are simple. The `course_id` can be directly obtained from the list box control, and all other pieces of information can be extracted from the returned instance and assigned to the associated textbox.

When performing this function to get the detailed course information from the database, no difference exists between the **Stored Procedure Method** and the **DataSet Method**. Both methods use the same process.

At this point, we have finished all coding jobs for our Windows-based client project. Now we can run the client project to test all functions of this project, as well as the functions of our Web service project. Before we can do this, make sure that the following jobs have been performed:

1. Our Web service is in the running status, and this can be checked by locating a small white icon on the status bar on the bottom of the screen. If you cannot find this icon, open our Web service project `WebServiceSQLInsert` and click on the Start Debugging button to run it. After the Web service starts to run, you can close the Web page if you like but it is still in the running status.
2. Two new course records, which we inserted before by testing the `Insert` button's click event procedure, have been deleted from the `Course` table in our sample database since we want to insert the same course records in this test.

The screenshot shows a Windows application window titled "CSE DEPT Course Form". It contains several input fields and a list box. At the top, there are two dropdown menus: "Faculty Name" with "Ying Bai" selected and "Query Method" with "Stored Procedure Method" selected. Below these is a "Course List" box containing the following items: CSC-132B, CSC-234A, CSE-434, CSE-438 (which is highlighted), CSE-526, and CSE-665. To the right of the list box is a "Course Information" section with several textboxes: "Course ID" (CSE-438), "Course" (Advd Logic & Microprocessor), "Schedule" (M-W-F: 11:00-11:55 AM), "Classroom" (TC-213), "Credits" (3), and "Enrollment" (35). At the bottom of the form are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 9.74. The running status of getting the detailed course information.

Now click on the Start Debugging button to run our client project. Insert two new courses by entering parameters listed in Tables 9.2 and 9.3 in Section 9.4.4.3.2 and clicking on the **Insert** button. Then perform the data validation by clicking on the **Select** button. To get the detailed course information for the selected `course_id` from the list box control, click one `course_id`, and immediately the detailed information about the selected `course_id` is displayed in those associated textboxes, which is shown in Figure 9.74.

Click on the **Back** button to terminate our client project.

A complete Windows-based Web service client project `WinClientSQLInsert` can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

9.4.5 Build Web-Based Web Service Clients to Consume the Web Services

As we did in Section 9.3.11, it can be found that there is no significant difference between developing a Web-based client application and developing a Windows-based client project to consume 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 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 consumed 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 as our new Web-based Web service client project `WebClientSQLInsert`. In fact, we can copy and rename that entire project

as our new Web-based client project, but we prefer to create a new ASP.NET website 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.
2. Add a Web service reference to our new project and modify the Web form page **Course.aspx** to meet our data insertion requirements.
3. Modify the codes in the related event procedures of the **Course.aspx.vb** 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** event procedure.
 - B. Develop the codes for the **Insert** button's click event procedure.
 - C. Develop the codes for the **TextChanged** event procedure of the Course ID textbox.
 - D. Modify the codes in the **Select** button's click event procedure. Also, add four user-defined subroutines: **ProcessObject()**, **FillCourseListBox()**, **FillCourseDataSet()**, and **FillCourseDetail()**. These four subroutines 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 procedure.
 - F. Modify the codes in the **Back** button's click event procedure.

Now let's start with the first step listed above.

9.4.5.1 Create a New Web Site 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 Web site project. Enter **C:\Chapter 9\WebClientSQLInsert** into the **Name** box that is next to the **Web Location** box, and click on the **OK** button to create this new project.

On the opened new project window, right click on our new project **WebClientSQLInsert** from the **Solution Explorer** window, and select the item **Add Existing Item** from the pop-up menu to open the **Add Existing Item** wizard. Browse to our Web project **SQLWebInsert** that can be found in the folder **DBProjects\Chapter 8** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1), and double click on this project folder to open all existing items for this Website project.

Select the item **Course.aspx** from the list and click on the **Add** button to add this item into our new Website project.

9.4.5.2 Add a Web Service Reference and Modify the Web Form Window

Perform the following operations to add this Web reference:

1. Open Visual Studio.NET 2010 and our Web service project **WebServiceSQLInsert**, and click on the **Start Debugging** button to run it.
2. Copy the URL from the **Address** bar in our running Web service project.
3. Then open another Visual Studio.NET 2010 and open our Web client project **WebClientSQLInsert**.

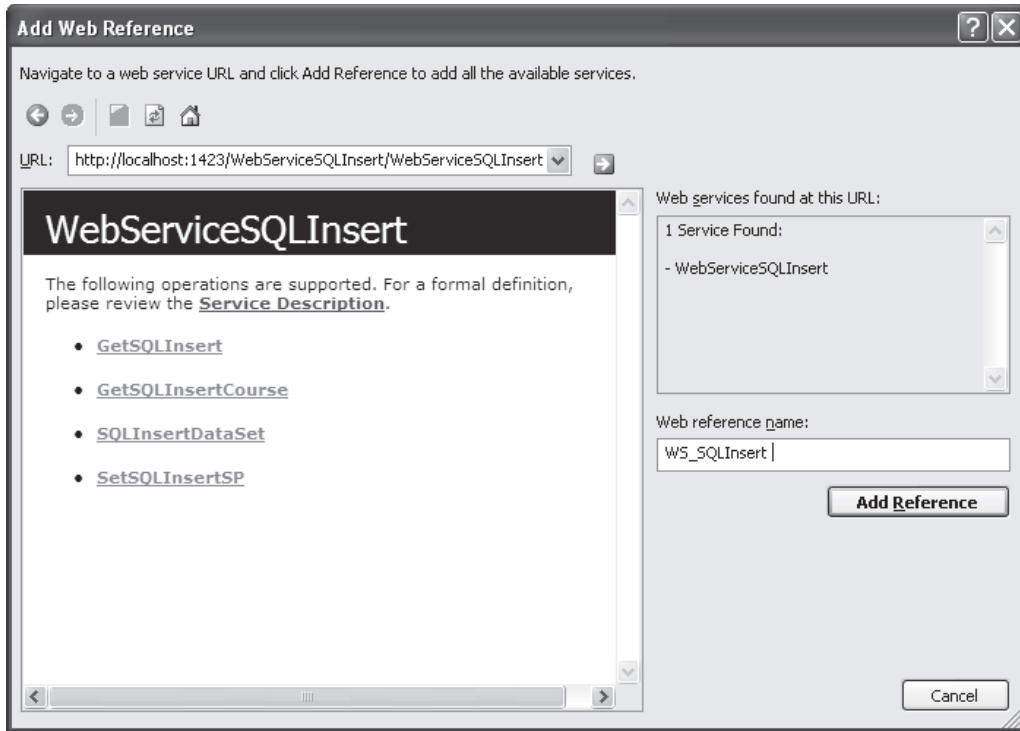


Figure 9.75. The finished Add Web Reference wizard.

4. Right-click on our client project **WebClientSQLInsert** from the Solution Explorer window, and select the item **Add Web Reference** from the pop-up menu to open the Add Web Reference wizard, which is shown in Figure 9.75.
5. Paste the URL we copied from step 2 into the URL box in the Add Web Reference wizard, and click on the **Green Arrow** button to enable the Visual Studio.NET 2010 to begin to search it.
6. When the Web service is found, the name of our Web service is displayed in the right pane, which is shown in Figure 9.75.
7. Alternately, you can change the name for this Web reference from **localhost** to any meaningful name such as **WS_SQLInsert** in our case. Click on the **Add Reference** button to add this Web service as a reference to our new client project.
8. Click on the **Close** button from our Web service built-in Web interface window to close our Web service page.

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 of the newly added Web reference:

- **WebServiceSQLInsert.disco**
- **WebServiceSQLInsert.discomap**
- **WebServiceSQLInsert.wsdl**

The modifications to the Web page of the `Course.aspx` include three steps:

1. Set the `AutoPostBack` property of the Course ID textbox to `True`. **This is very important** since when the content of this textbox is changed during the project runs, a `TextChanged` event occurs. However, this event only occurs in the client side, not the server side. Our Web-based client project is running is a Web server or server-side, so this event cannot be responded by the server. Therefore, the command inside this event procedure 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 to send 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 combo box control. Name this `DropDownList` as `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 allow users to select one of them to perform the associated data insertion as the project runs.
3. Change the ID property of the Credit textbox from `txtCredit` to `txtCredits`.

Your modified `Course.aspx` Web form window is shown in Figure 9.76. Go to the `File|Save All` menu item to save these modifications.

9.4.5.3 Modify the Codes for the Related Event Procedures

The first modification is to change the codes in the `Page_Load` event procedure and some global variables.

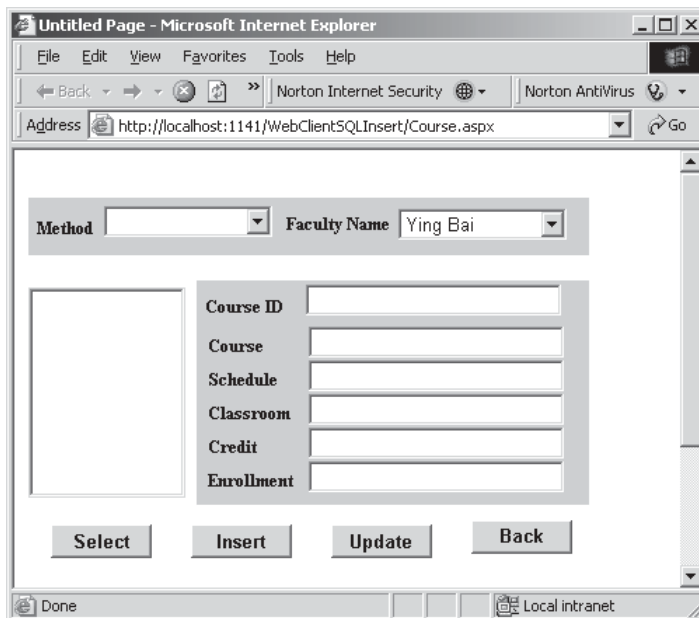


Figure 9.76. The modified Course page window.

9.4.5.3.1 Modify the Codes in the Page_Load Event Procedure Perform the following changes to complete this modification:

1. Remove the second Imports command Imports System.Data.SqlClient from the top of this page since we do not need it in this application.
2. Remove the form level variables CourseTextBox(5) since we do not need it in this application.
3. Add the following three form level variables into the Form's General Declaration section:

```
Private dsFlag As Boolean
Private wsDataSet As New DataSet
Private wsSQLResult As New WS_SQLInsert.SQLInsertBase
```

4. Remove the If block in the Page_Load event procedure and the associated global connection object Application("sqlConnection").
5. Add the codes to display two Web methods in the Method combo box control ComboMethod.

Your finished codes for the Page_Load event procedure should match the one that is shown in Figure 9.77. The newly added codes have been highlighted in bold.

The next step is to develop the codes for the Insert button's click event procedure.

9.4.5.3.2 Develop Codes for the Insert Button Event Procedure The function of this piece of codes 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 piece of codes is basically identical with those in the same event procedure of the Windows-based client project we developed in the last section. Therefore, we can copy those codes from that event procedure and paste them into our current procedure with a few modifications.

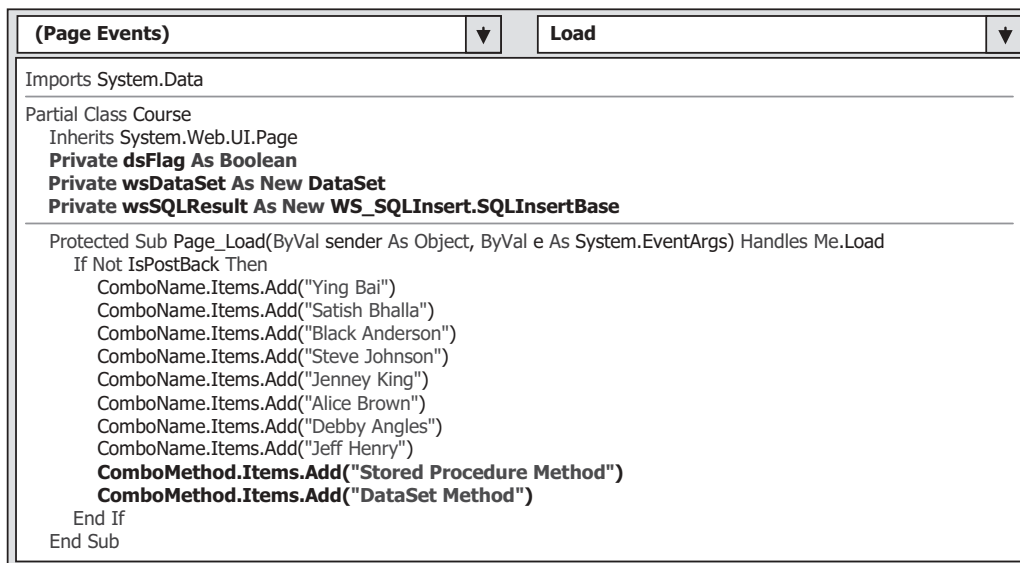


Figure 9.77. The modified Page_Load event procedure.

Open the Windows-based client project `WinClientSQLInsert` in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1), and browse to the `Insert` button's click event procedure. Copy all codes from that event procedure and paste them into the `Insert` button's click event procedure in our current Web-based client project `WebClientSQLInsert`.

The only modification to this event procedure 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 `MsgBox()` should be replaced by the `Write()` method of the `Response` object of the server class since `MsgBox()` can only be used in the client side.

Your finished codes for the `Insert` button's click event procedure should match the one that is shown in Figure 9.78. The modification parts have been highlighted in bold.

Let's have a quick review for this piece of codes to see how it works.

- A. If the user selected the **Stored Procedure Method** to perform this data insertion, the Web method `SetSQLInsertSP()` 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.
- B. 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.

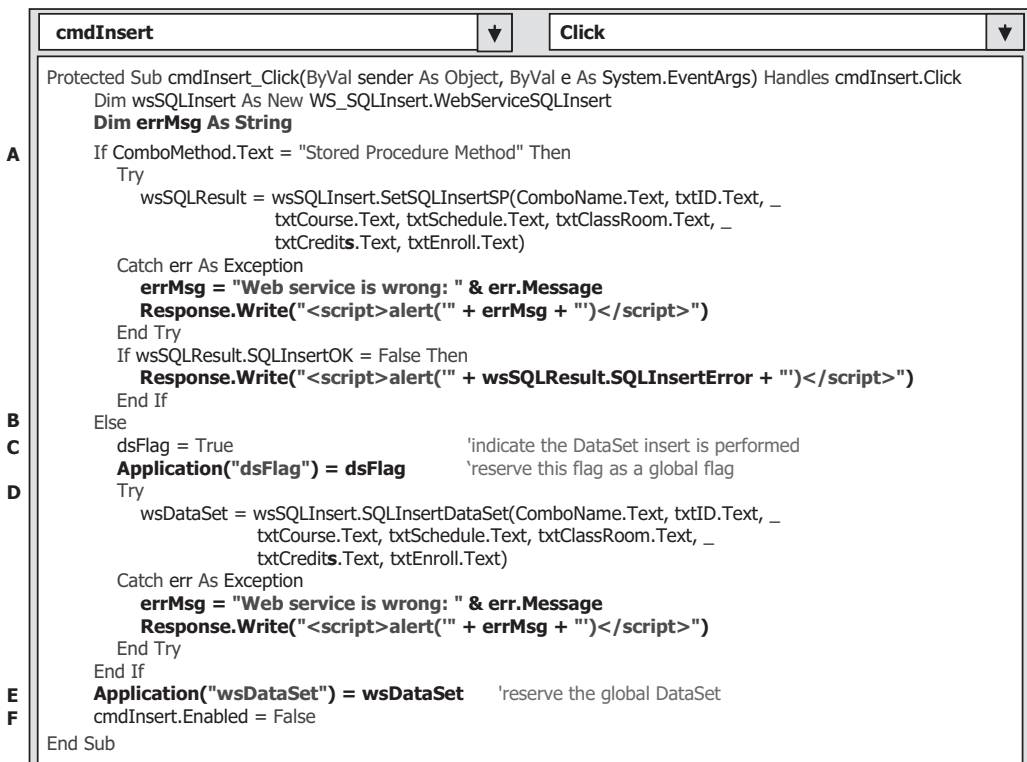


Figure 9.78. The codes for the `Insert` button event procedure.

- C. This flag is set up and stored in a global variable using the Application state. The reason we need to make this setup is that the Web method `SQLInsertDataSet()` has two functions: insert data into the database and retrieve data from the database. In order to perform the data retrieving using this method, first we must insert data using this method. Otherwise, no data can be retrieved if no data has been inserted into the database 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 variables' 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.
- D. 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.
- E. The returned `DataSet` object `wsDataSet` that contains all `course_id` is a form-level variable. Because of the same reason as we discussed in step C, we need to use an Application state to store this `DataSet` since we need to pick up all `course_id` from it when we perform the validation process later by clicking on the **Select** button. Otherwise, the content of this `DataSet` will be refreshed each time when a refreshed Course page is sent back by the server.
- F. Finally, the **Insert** button is disabled to avoid multi-insertion of the same data into the database.

9.4.5.3.3 Develop Codes for the CourseID TextChanged Event Procedure The codes for this event procedure are very simple. Open this event procedure by double clicking on the Course ID textbox from the Web page window and enter the following code into this event procedure:

```
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 multi-insertion of the same record into the database. But 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 when a new course record is ready to be inserted, the **Insert** button should be enabled for this situation.

Another important point is that making sure that the `AutoPostBack` property of this Course ID textbox is set to `True` to allow the server to send back a `TextChanged` event to the client when its content is changed.

9.4.5.3.4 Modify the Codes in the Select Button's Click Event Procedure The codes in this event procedure are similar to those codes we developed in the same event procedure in our Windows-based client project `WinClientSQLInsert`. So we can copy those codes and paste them into our current **Select** button's click event procedure with a few modifications. Open the **Select** button's click event procedure from our

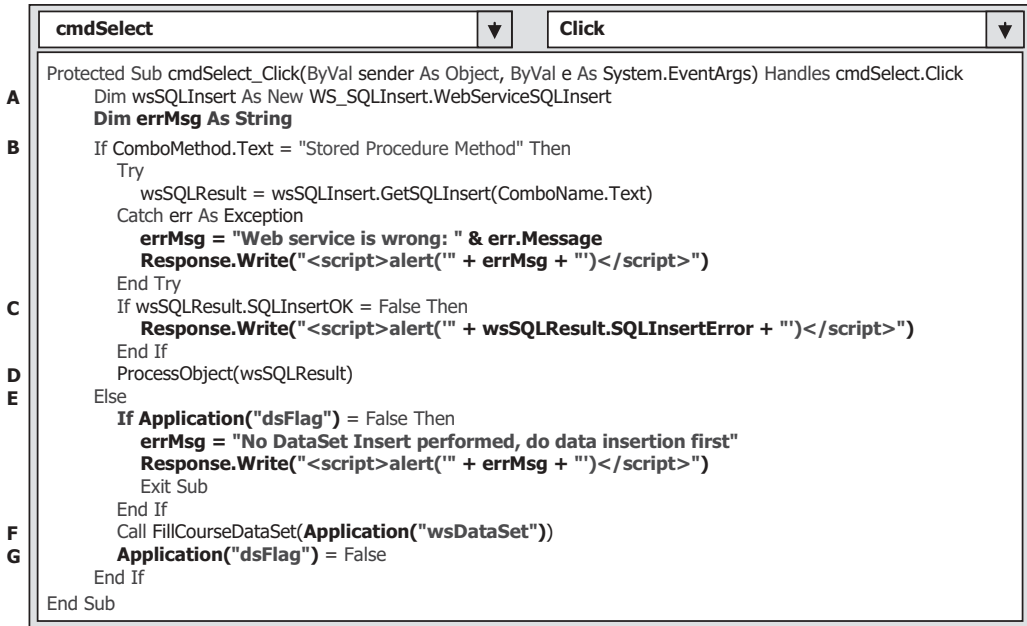


Figure 9.79. The codes for the Select button event procedure.

Windows-based client project WinClientSQLInsert, copy those codes, and paste them into our **Select** button's click event procedure. The only modification to this piece of copied codes is to change the Windows-based message box function `MsgBox()` to the Web-based message box function. Your finished codes for this event procedure are shown in Figure 9.79. The modified parts have been highlighted in bold.

Let's take a quick review for this piece of codes 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 the associated Web methods built in the Web service together. 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_id` taught by the selected faculty based on the input faculty name. If any error occurred during the execution of this Web method, the error source is reported and displayed with an `alert()` script 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 any error is detected, the user-defined subroutine `ProcessObject()`, whose detailed codes are shown in Figure 9.80, is called to extract all retrieved `course_id` from the returned instance and add them into the list box control in our client page window.
- E.** If 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 exe-

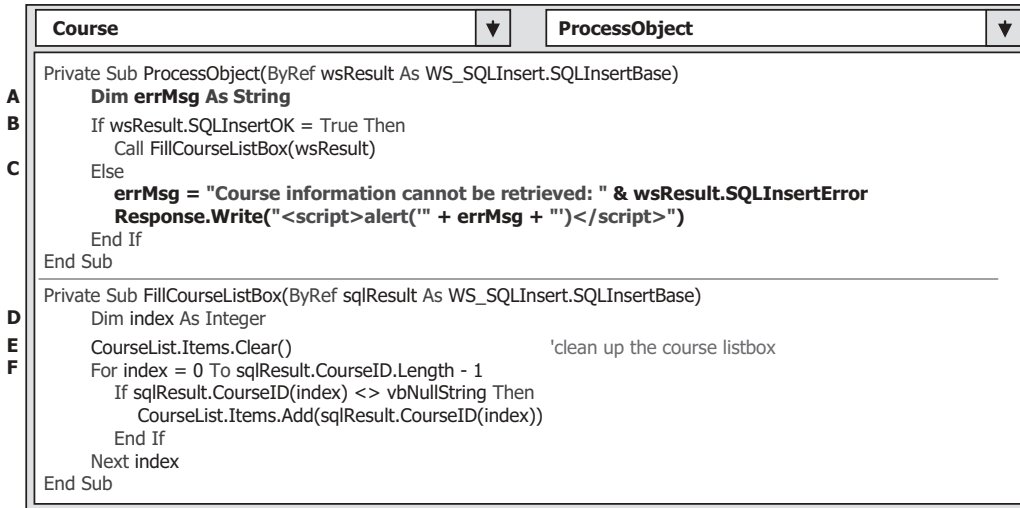


Figure 9.80. The codes for subroutines ProcessObject() and FillCourseListBox().

cuted once since our current data query needs to extract all `course_id` 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, therefore, we do not have any returned DataSet available. A warning message is displayed, and the procedure is exited if that situation occurred.

- F.** If the `dsFlag` is `True`, which means that the Web method `SQLInsertDataSet()` has been executed, and a returned DataSet that contains all `course_id` is available. A user-defined subroutine `FillCourseDataSet()` is executed to extract all `course_id` from that returned DataSet and add them into the list box 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 subroutine calling.
- G.** Finally, the `dsFlag` stored in an Application state is reset to `False`.

The detailed codes for the subroutines `ProcessObject()` and `FillCourseListBox()` are shown in Figure 9.80.

Let's have a closer look at this piece of codes 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.
- B.** First, we need to check the member data `SQLInsertOK` to make sure that the Web method is executed successfully. If it is, a user-defined subroutine procedure `FillCourseListBox()` is called to fill all `course_id` contained in the returned instance to the list box 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 subroutine `FillCourseListBox()`, first, a local integer variable `index` is created, and it works as a loop number for a `For` loop to continuously pick up all `course_id` from the returned instance and add them into the list box control.

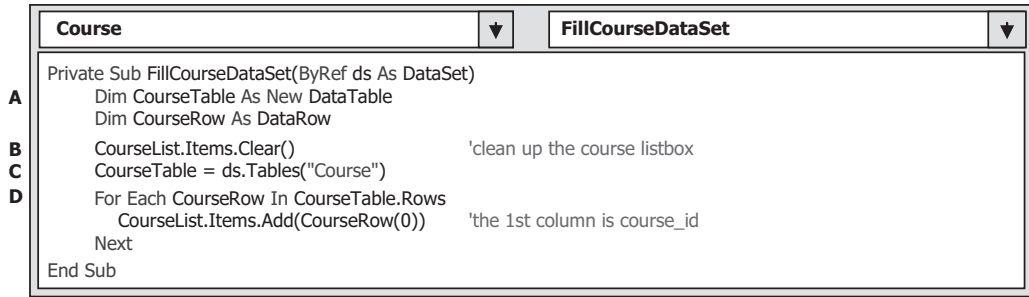


Figure 9.81. The codes for the subroutine FillCourseDataSet().

- E.** The course list box control is cleaned up first before any `course_id` can be added into it. This process is very important in displaying all `course_id`, 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`. One point to be noted is the upper bound and the length of this array. The length or the size of this array is 11, but the upper bound of this array is 10, 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, a valid `course_id` is added into the list box control by using the `Add()` method.

The codes for the subroutine `FillCourseDataSet()` are shown in Figure 9.81. This piece of codes is identical with that in the same subroutine we developed in our Windows-based client project `WinClientSQLInsert`. You can copy it from that Windows-based project and paste it into our current project.

Let's have a look at the codes in this subroutine to see how they work.

- A.** Two objects, a `DataTable` and a `DataRow`, are declared at the beginning of this subroutine since we need to use them to perform the data extraction from the returned instance and data addition to the list box control.
- B.** The list box control is first cleaned up to avoid messy displaying of multiple `course_id`.
- C.** The `CourseTable` object is initialized by adding a new data table named "Course" and is assigned to the `DataSet` object `ds`.
- D.** A `For Each . . . In` 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 noted 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 procedure and add the fourth subroutine `FillCourseDetail()`. Before we can continue to do these jobs, first we need to delete the following procedures and subroutines from our current project:

- `FillCourseReader()`
- `FillCourseReaderTextBox()`
- `MapCourseTable()`

Now let's modify the codes in the `SelectedIndexChanged` event procedure and add the fourth user-defined subroutine procedure `FillCourseDetail()`.

9.4.5.3.5 Modify the Codes in the SelectedIndexChanged Event Procedure The function of this piece of codes 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 and selected one `course_id` from the list box control. In fact, the main coding job is performed inside the `SelectedIndexChanged` event procedure of the list box control. Because when the user clicks or selects a `course_id` from the list box control, a `SelectedIndexChanged` event is issued, and this event is passed to the associated `SelectedIndexChanged` event procedure.

To pick up the detailed course information for the selected `course_id`, the Web method `GetSQLInsertCourse()` 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 procedure are identical with those we did for the same event procedure in our Windows-based client project `WinClientSQLInsert`. So we can copy those codes from that event procedure and paste them into our current project with a few modifications.

Double-click on the list box control `CourseList` from our client page window to open the `SelectedIndexChanged` event procedure of the list box control. Copy and paste those codes into our current Web-based project. The only modification is to change the Windows-based `MsgBox()` method to the Web-based script message method `alert()`. Your finished `SelectedIndexChanged` event procedure should match the one that is shown in Figure 9.82. The modified parts have been highlighted in bold.

Let's take a closer look at this piece of codes 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

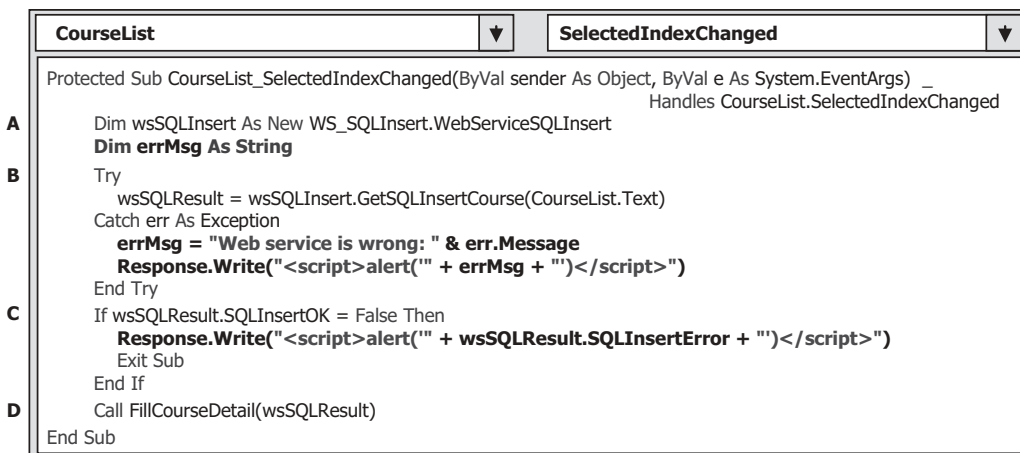


Figure 9.82. The modified codes for the `SelectedIndexChanged` event procedure.

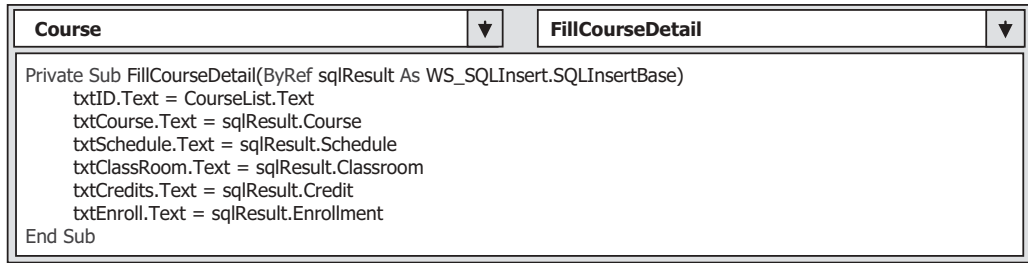


Figure 9.83. The codes for the subroutine FillCourseDetail().

in the 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 list box control as the argument to perform this course information retrieving. The selected `course_id` is stored in the `Text` property of the `CourseList` control. An exception message is displayed if any error was encountered during the execution of this Web method and caught by the system method `Catch`.
- 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`, this means that an application error occurred during the running of this Web method. A related error message is displayed and the subroutine is exited.
- D.** If everything is fine, the user-defined subroutine `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 subroutine `FillCourseDetail()` is shown in Figure 9.83.

This piece of codes is identical with that we developed in the same subroutine in our Windows-based client project `WinClientSQLInsert`. You can copy it from that project and paste it in this project.

The function of this piece of codes is straightforward without tricks. Each piece of course information is extracted from the returned instance and assigned to the associated textbox control in our client page window.

9.4.5.3.6 Modify the Codes in the Back Button's Click Event Procedure The final modification is to change the codes for the `Back` button's click event procedure. When this button is clicked by the user, our client project should be terminated. Open this event procedure and replace the original codes with the following codes in this event procedure 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 coding jobs for this Web-based client project. Before we can run this project to test the data insertion and validation functionalities, make sure that the following tasks 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 select the item **Set As Start Page** from the pop-up menu.
- Our Web service **WebServiceSQLInsert** is in the running status, and this can be checked by locating a small white icon on the status bar on 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 its Web page if you like but it is still in the running status.
- Two new course records, **CSE-665** and **CSE-526**, which we inserted before by testing the **Insert** button's click event procedure, should have been deleted from the Course table in 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** combo box control. Then select the default faculty **Ying Bai** from the **Faculty Name** combo box control, enter the first new course record (shown in Table 9.2 in Section 9.4.4.3.2) into the associated textboxes, and then click on the **Insert** button. Perform the similar operation to insert the second new course record (shown in Table 9.3 in Section 9.4.4.3.2) with the **DataSet Method** selected. Your running Web page is shown in Figure 9.84.

To validate these data insertions, click on the **Select** button for **DataSet Method** and then the **Stored Procedure Method**. The running result is shown in Figure 9.85.

You can find that our two newly inserted courses **CSE-665** and **CSE-526** have been added into and retrieved from our database and displayed in the list box control.

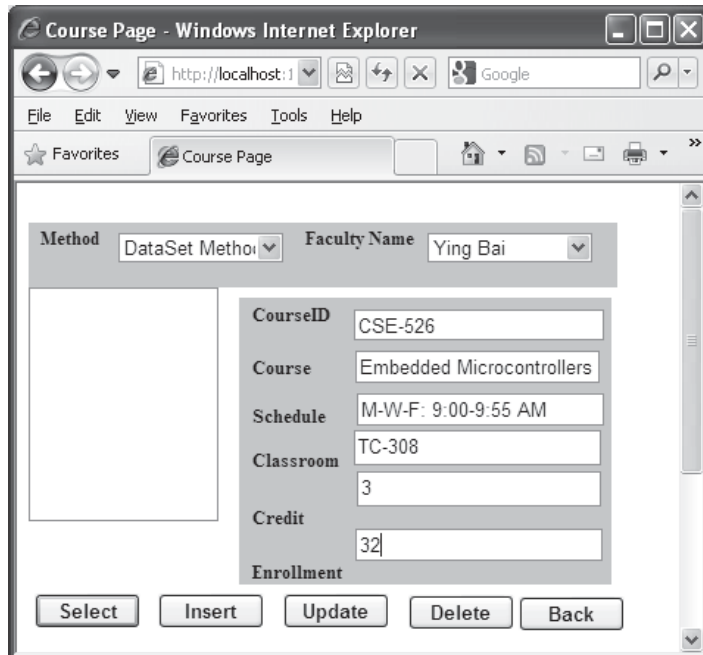


Figure 9.84. The running status of inserting new course records.

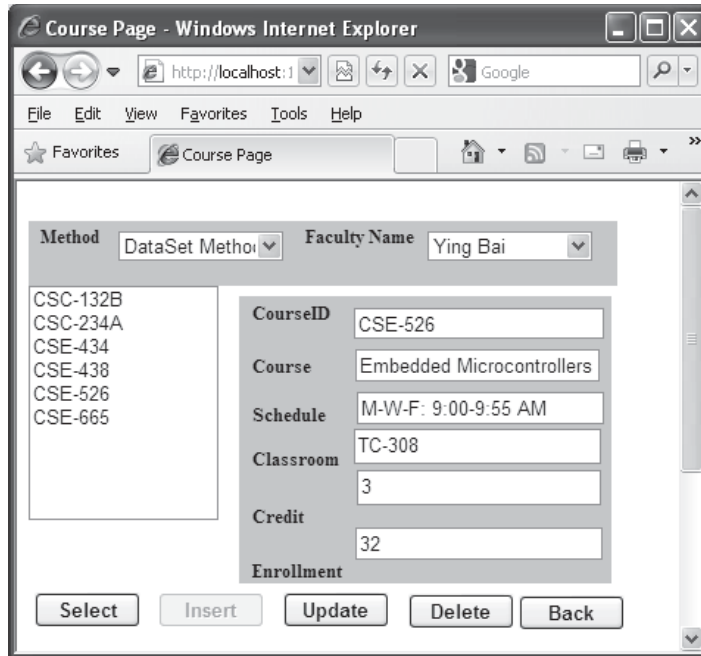


Figure 9.85. The running status of the data validation process.

To get detailed course information for a specific course, click a desired `course_id` from the list box control. Immediately, the detailed course information for the selected `course_id` is displayed on each associated textbox, which is shown in Figure 9.86.

You can try to get the detailed information for different courses by selecting different `course_id` from the list box control via either DataSet or Stored Procedure method. Click on the **Back** button to terminate our Web client project.

A completed Web-based Web service client project `WebClientSQLInsert` can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in 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 against 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.

To save time and space, we can modify an existing Web service project `WebServiceSQLInsert` we developed in Section 9.4 to make it as our new Web service project `WebServiceSQLUpdateDelete`.

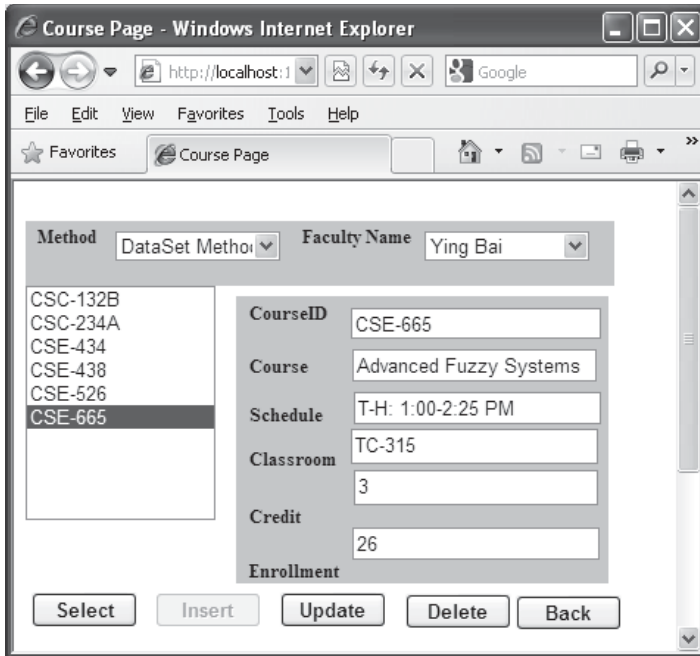


Figure 9.86. The running status of getting the detailed course information.

9.5.1 Modify an Existing Web Service Project

Open the Internet Explorer, browse to the folder DBProjects\Chapter 9 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1), and select the Web service project WebServiceSQLInsert. Copy this project and paste it into our development folder C:\Chapter 9 in our computer. Rename this project to WebServiceSQLUpdateDelete.

Open Visual Studio.NET 2010 and our new Web service project WebServiceSQLUpdateDelete, and 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.vb to SQLBase.vb by right-clicking on this class file and select the Rename item from the pop-up menu.
2. Similarly, rename our code-behind page from WebServiceSQLInsert.vb to WebServiceSQLUpdateDelete.vb.
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.vb"`
To: `CodeBehind = "~/App_Code/WebServiceSQLUpdateDelete.vb"`
 - b. From: `Class = "WebServiceSQLInsert"`
To: `Class = "WebServiceSQLUpdateDelete"`

5. 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 class name from `SQLInsertBase` to `SQLBase`.
 - b. Change `Public SQLInsertOK As Boolean` to `Public SQLOK As Boolean`.
 - c. Change `Public SQLInsertError As Boolean` to `Public SQLError As Boolean`.
6. Double-click on our new code-behind page `WebServiceSQLUpdateDelete.vb` from the Solution Explorer window to open it. Change our Web class name that is located after the access mode `Public Class` 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 Guideline in Modifying 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 as 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 as our new Web method `GetSQLCourse()` that will return all `course_id`, including the original and the updated `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()` that 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 `SQLDeleteSP()`, and this method will be used to 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 the Web Method from `SetSQLInsertSP` to `SQLUpdateSP`

The function of this Web method is to call an SQL Server stored procedure named `dbo.WebUpdateCourseSP()` that will be developed in Section 9.5.3.1 to perform the data updating for a course record based on the `course_id`.

Regularly, 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 very complicated if one wants to update a primary key in a parent table since that primary key may be used as foreign keys in many other child tables. Therefore, one has to update those foreign keys first in many child tables before the primary key can be updated in the parent table. In this application, we concentrate on updating all other columns for a course record without touching the primary key `course_id`.

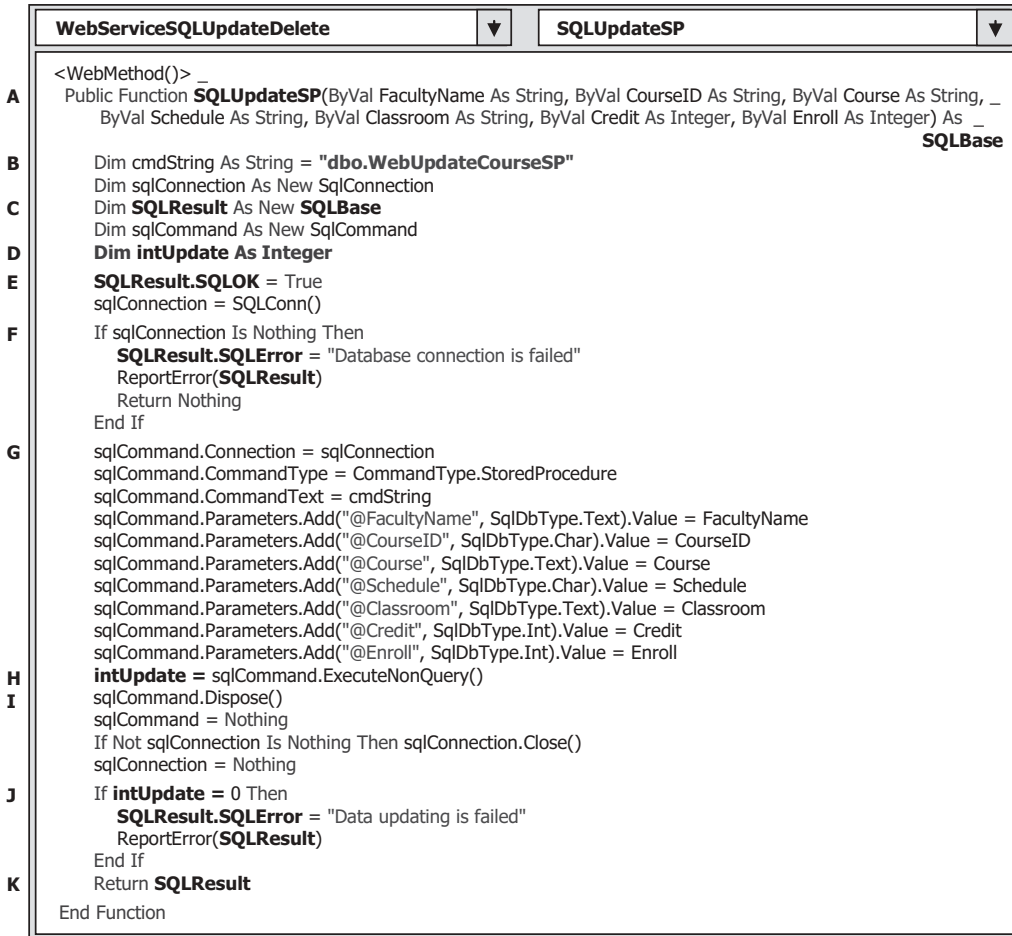


Figure 9.87. The modified codes for the Web method SQLUpdateSP().

Open our new Web service project **WebServiceSQLUpdateDelete** and the Web method **SetSQLInsertSP()**. Perform the modifications that are shown in Figure 9.87 to this method. The modified parts have been highlighted in bold.

Let's have 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 that will be developed in Section 9.5.3.1. Keep in mind that this name must be identical with the name of 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 into the member data **SQLException** and reported using a user-defined subroutine **ReportError()**.
- G. The **Command** object is initialized with associated data objects, such as connection object, command text, and command type. One point to be noted is 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 process for the **Command** object is to assign all input or updating parameters to the associated dynamic parameter in the **UPDATE** statement.
- 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 this value 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, this means that no any row has been updated in our **Course** table and this data updating has failed. An error message is sent to the member data **SQLException** and reported using the subroutine **ReportError()**.
- K. Finally, the instance **SQLResult** that contains the running status of this Web method is returned to the calling procedure.

Go to the **File|Save All** menu item to save these modifications.

9.5.2.2 *Modify the 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 the **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 list box control in the client project.

Open this Web method and perform the modifications that are shown in Figure 9.88 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 **GetSQLInsert** to **GetSQLCourse**. 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 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 a user-defined subroutine **ReportError()**.
- 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



Figure 9.88. The modified codes for the Web method GetSQLCourse().

@fname 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 subroutine FillCourseReader() 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 any row has been retrieved from the Course table, an error message is displayed and reported using the subroutine ReportError().
- 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 procedure.

The only modification to the user-defined subroutine FillCourseReader() is to change the data type of the first input argument sResult from SQLInsertBase to SQLBase.

9.5.2.3 Modify the 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 SQL Server 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 that are shown in Figure 9.89 to this method. The modified parts have been highlighted in bold.

Let's have 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 `SQLError` and reported using the subroutine `ReportError()`.



Figure 9.89. The 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`, which 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 subroutine `FillCourseDetail()` 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`, this means that no row has been retrieved from the Course table. An error message is displayed and reported using the subroutine `ReportError()`.
- 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 subroutine `FillCourseDetail()` is to change the data type of the first input argument `sResult` from `SQLInsertBase` to `SQLBase`.

The last modification to this Web project is to modify the subroutine `ReportError()`. Perform the following modifications to this subroutine:

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 `MsgBox(ErrSource.SQLInsertError)` to `MsgBox(ErrSource.SQLError)`.

Next, let's develop a new Web method `SQLDeleteSP()` to perform the data deleting action against the Course table in our sample database via this project.

9.5.2.4 Add a New 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 operational 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 2.9.4 and Figure 2.19 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. To save time and space, as well as the efficiency, we place these two queries into a stored procedure named **WebDeleteCourseSP()** that we will develop in the following sections. A single input parameter **course_id** is passed into this stored procedure. At this moment, we just assume that we have already developed that stored procedure and will use it in this Web method.

Open our code-behind page **WebServiceSQLUpdateDelete.vb** and create this Web method **SQLDeleteSP()**, which is shown in Figure 9.90.

Let's take a closer look at this piece of codes 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.
- 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 of the **ExecuteNonQuery()** method after this method runs.

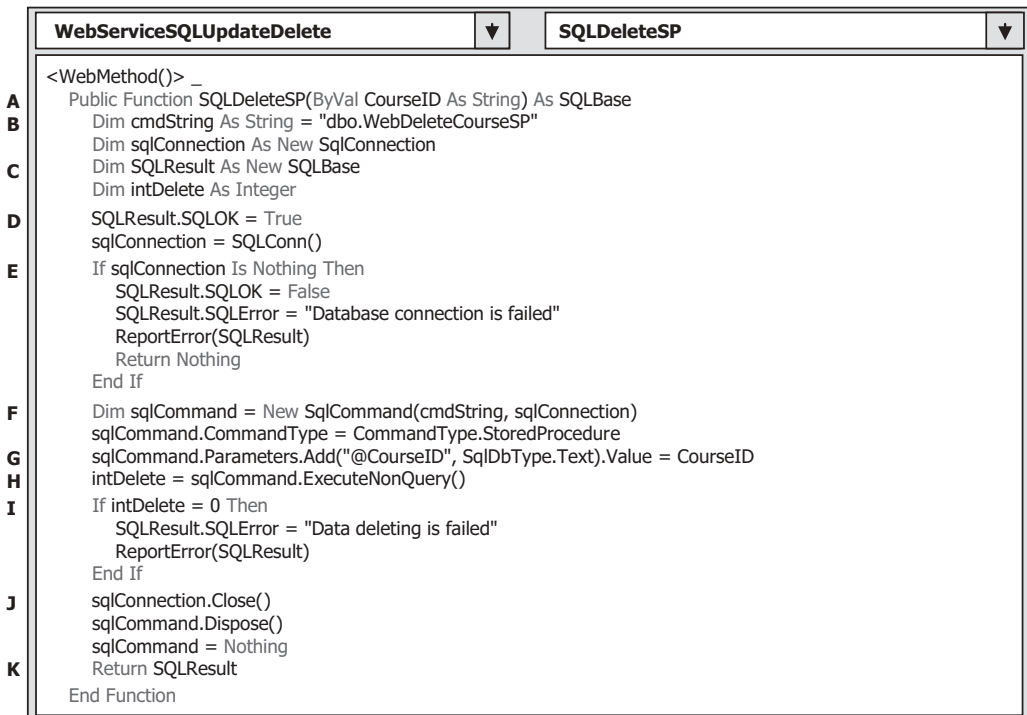


Figure 9.90. The codes for the Web method **SQLDeleteSP()**.

- 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 subroutine `ReportError()`.
- 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 object 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 to indicate the running status of this method, and the returned value is assigned to the local integer variable `intDelete`.
- 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 subroutine `ReportError()`.
- 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, it is the time for us to develop two stored procedures we need to use 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 2010 environment.

9.5.3.1 Develop the Stored Procedure `WebUpdateCourseSP`

Open Visual Studio.NET 2010 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` wizard.

Enter the codes that are shown in Figure 9.91 into this procedure to make it as our new stored procedure. The new entered codes have been highlighted in bold.

The actual name of this procedure is `dbo.WebUpdateCourseSP`, but generally, we call this procedure as `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


```

CREATE PROCEDURE dbo.WebUpdateCourseSP
(
    @FacultyName VARCHAR(30),
    @CourseID VARCHAR(10),
    @Course text,
    @Credit int,
    @Classroom text,
    @Schedule text,
    @Enroll int
)
AS
    DECLARE @FacultyID VARCHAR(10)
    SET @FacultyID = (SELECT faculty_id FROM Faculty
    WHERE faculty_name LIKE @FacultyName)
    UPDATE Course SET course = @Course, credit = @Credit, classroom = @Classroom,
    schedule = @Schedule, enrollment = @Enroll, faculty_id = @FacultyID
    WHERE (course_id LIKE @CourseID)
    RETURN

```

Figure 9.91. The codes for the new stored procedure WebUpdateCourseSP().

The stored procedure <[dbo].[WebUpdateCourseSP]> requires the following parameters:

Type	Direction	Name	Value
varchar	In	@FacultyName	Ying Bai
varchar	In	@CourseID	CSE-665
text	In	@Course	Neural Networks
int	In	@Credit	3
text	In	@Classroom	TC-316
text	In	@Schedule	M-W-F: 11:00-11:...
int	In	@Enroll	28

OK Cancel

Figure 9.92. The input parameters to stored procedure WebUpdateCourseSP().

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.

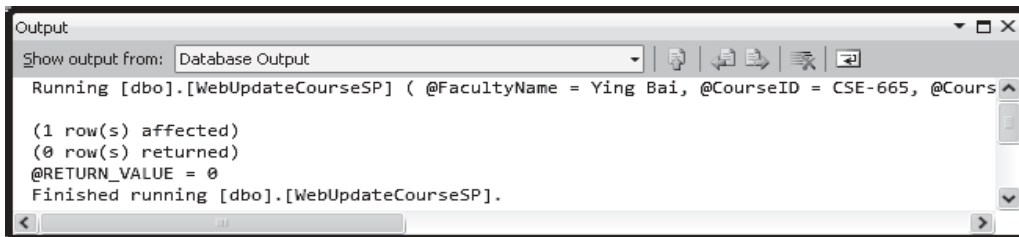
Go to the File|Save StoredProcedure1 menu item to save this new stored procedure.

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 Execute item from the pop-up menu to open the Run Stored Procedure wizard, which is shown in Figure 9.92.

Enter a group of updating parameters shown in Table 9.4 into the Value box in the Run Stored Procedure wizard as the input parameters (refer to Figure 9.92).

Table 9.4. The input parameters to the stored procedure

Parameter Name	Parameter Value
@FacultyName	Ying Bai
@CourseID	CSE-665
@Course	Neural Networks
@Credit	3
@Classroom	TC-316
@Schedule	M-W-F: 11:00-11:55 AM
@Enroll	28

**Figure 9.93.** The running result of the stored procedure WebUpdateCourseSP().

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 9.93.

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, and finally open the Course data table by right-clicking on it and selecting the item **Show Table Data** from the pop-up menu to try to find this updated course record. 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 is executed (sometimes you need to refresh this table to see the updated result).

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 build the second stored procedure **WebDeleteCourseSP()**.

Table 9.5. The recovered course record for CSE-665

Column Name	Column Value
course_id	CSE-665
course	Advanced Fuzzy Systems
credit	3
classroom	TC-315
schedule	T-H: 1:00-2:25 PM
enrollment	26
faculty_id	B78880

```

CREATE PROCEDURE dbo.WebDeleteCourseSP
(
    @CourseID VARCHAR(10)
)
AS
    DELETE FROM StudentCourse WHERE course_id LIKE @CourseID
    DELETE FROM Course WHERE course_id LIKE @CourseID
RETURN

```

Figure 9.94. The codes for the stored procedure WebDeleteCourseSP().

9.5.3.2 Develop the Stored Procedure WebDeleteCourseSP

Open Visual Studio.NET 2010 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** wizard.

Enter the codes that are shown in Figure 9.94 into this procedure to make it as our new stored procedure. The newly entered codes have been highlighted in bold.

The actual name of this procedure is **dbo.WebDeleteCourseSP**. However, generally, we call this procedure as **WebDeleteCourseSP** without the prefix **dbo** since this prefix is added automatically by the SQL Server engine when a new stored procedure is created.

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 selected **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.

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 **WebDeleteCourseSP** from the Server Explorer window and select **Execute** item from the pop-up menu to open the Run Stored Procedure wizard, which is shown in Figure 9.95.

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 9.96.

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** in the Server Explorer window and then expand to our **Course** table under the **Tables** folder, and 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 opened, immediately, you can find that the course with the **course_id** of **CSE-526** has been deleted from our **Course** (parent) table. However, since this is a newly added course and no student has taken this course yet, therefore, you cannot find this course from the **StudentCourse** table.

It is highly recommended to recover those deleted records from both tables since we will use the same input parameter later to delete this record again when we test our Web service project. Because this is a newly added course and no student has taken this course

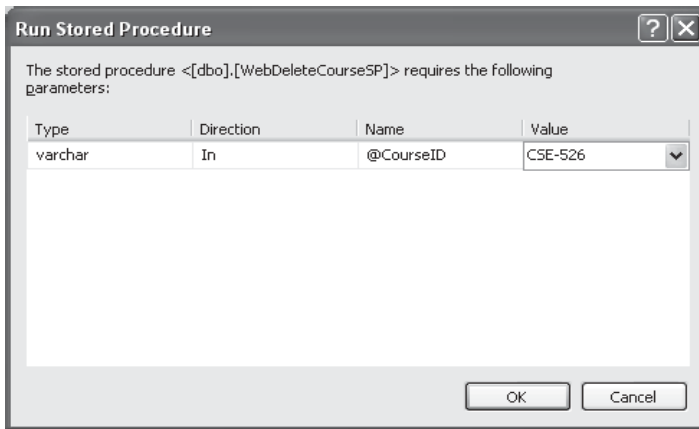


Figure 9.95. The Run Stored Procedure wizard.

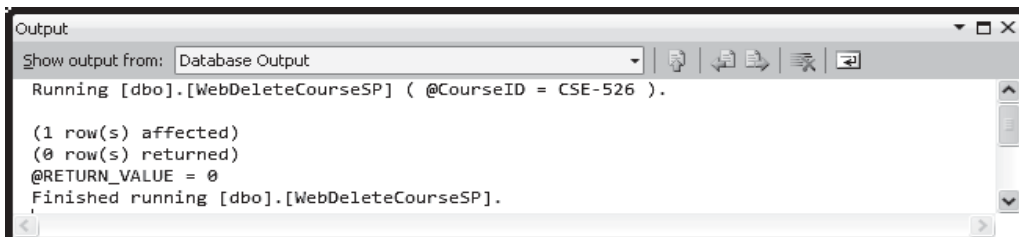


Figure 9.96. The running result of the stored procedure **WebDeleteCourseSP()**.

yet, therefore, we do not need to recover it for the StudentCourse table. So just perform a recovering for the Course table by adding the record that is shown in Table 9.6.

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 9.97.

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 9.98.

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

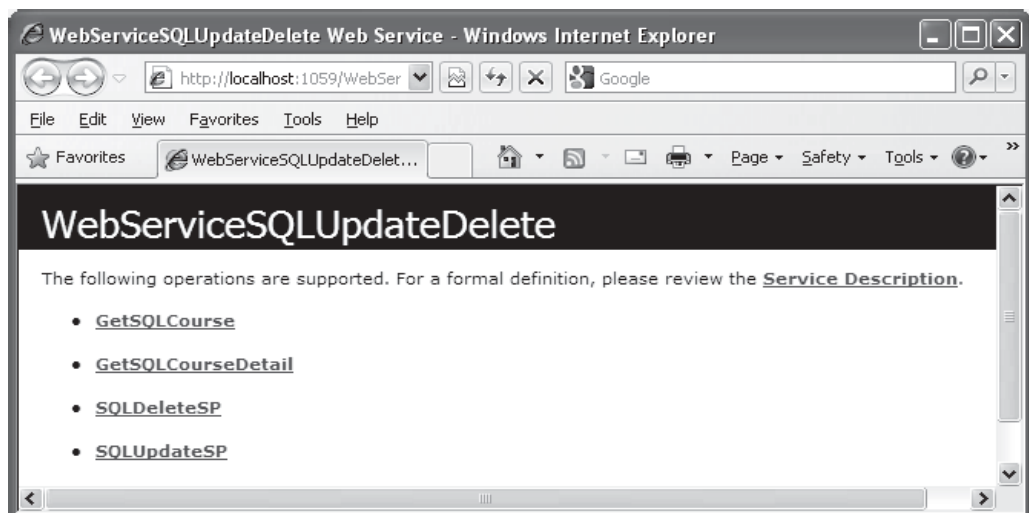


Figure 9.97. The running status of the Web service project.

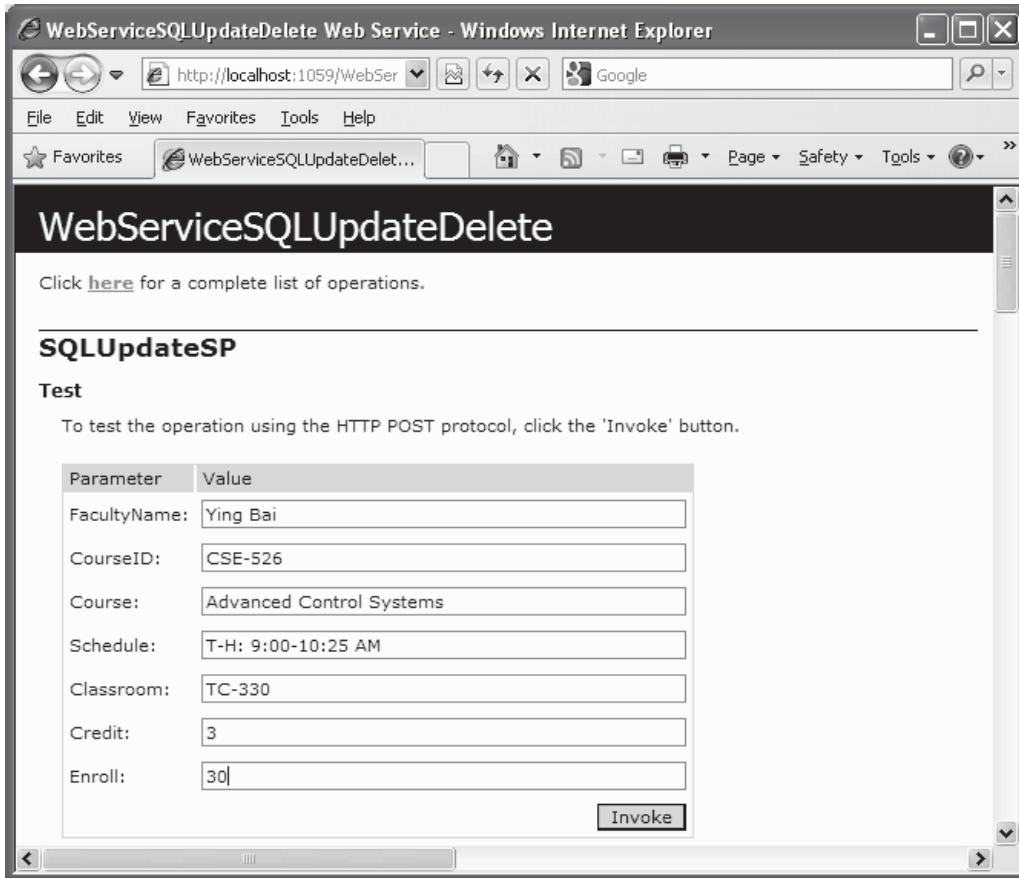


Figure 9.98. The parameter-input interface.

Enter the updated parameters shown in Figure 9.98 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 9.99.

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 any data should be returned from the execution of this data updating, therefore, all data stored in the returned instance, including the `CourseID()` array that has 11 elements and two integers **Credit** and **Enrollment**, are either **true** or zero.

To confirm this data updating, we can call some other Web methods to do this job. First, we want to get back all courses (exactly all `course_id`) taught by the selected faculty. To do that, close the running result interface window shown in Figure 9.99, and click on the **Back** arrow to return to the home page of this built-in interface. Then click on the Web method **GetSQLCourse** to obtain all `course_id`. Enter the faculty name **Ying Bai** into the **Value** box as the input parameter to this Web method, which is shown in Figure 9.100. Click on the **Invoke** button to run this method.

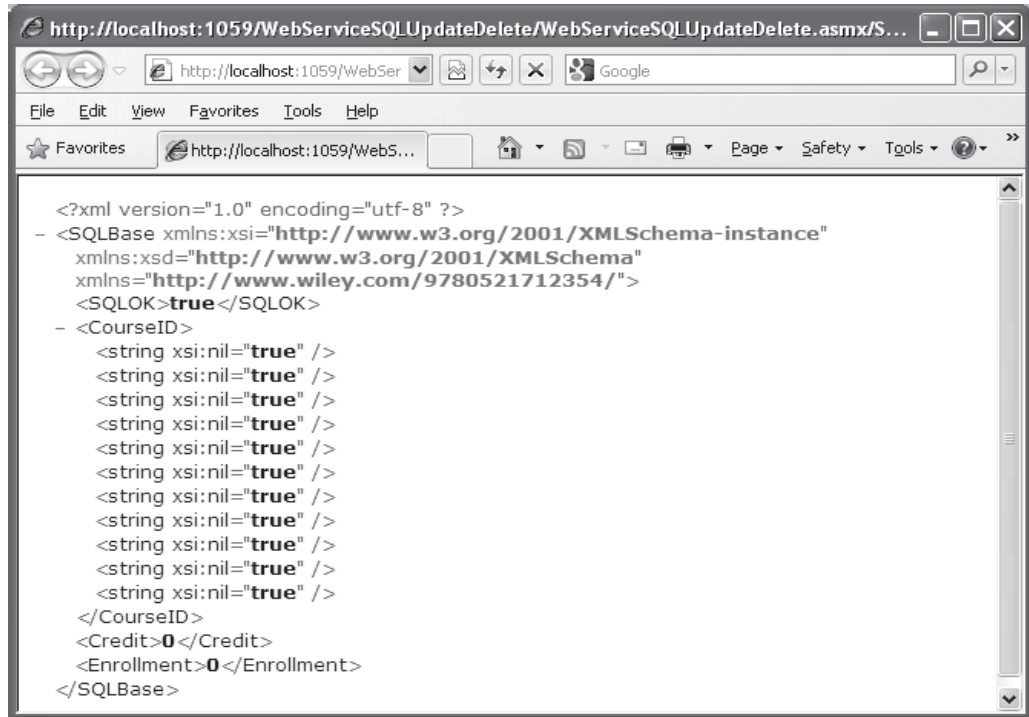


Figure 9.99. The running result for the Web method SQLUpdateSP().

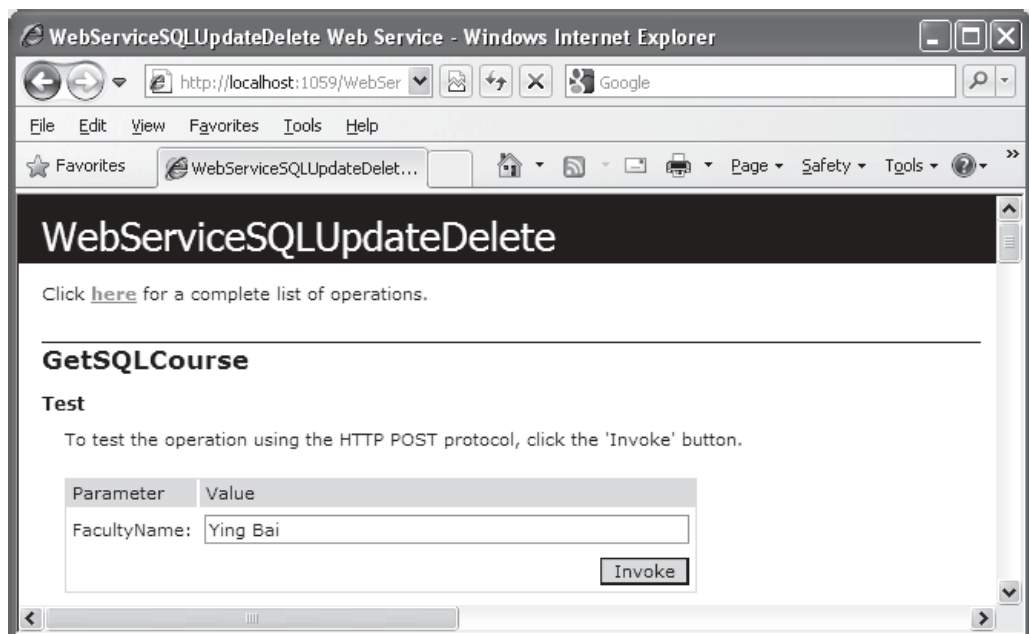


Figure 9.100. The parameter-input built-in Web interface.

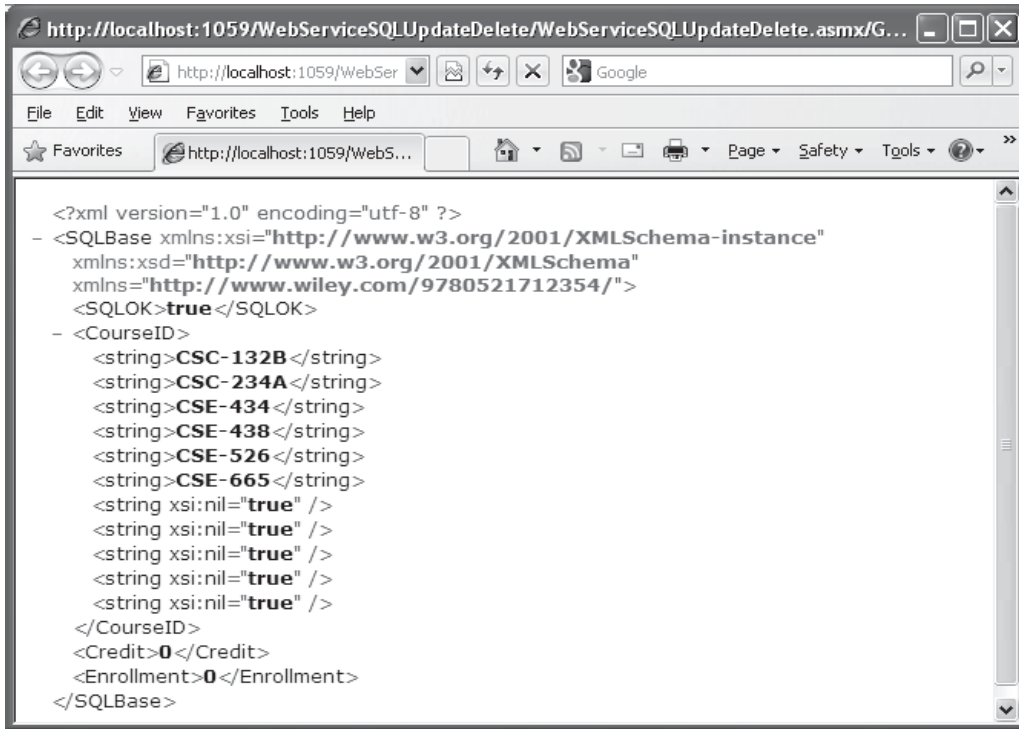


Figure 9.101. The running result for the Web method GetSQLCourse().

The running result for the Web method GetSQLCourse() is shown in Figure 9.101.

It can be found from Figure 9.101 that all six courses (or `course_id`) taught by the selected faculty are retrieved and displayed in XML tags in this built-in Web 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 9.101 and click on the Back arrow to return to the home page of our Web service. Click on the Web method GetSQLCourseDetail() to run it. Then enter CSE-526 as the input parameter that is shown in Figure 9.102 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 9.103.

It can be found that the course CSE-526 has been updated based the input parameters we entered for the Web method SQLUpdateSP() in Figure 9.98.

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 and click on the Back arrow 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 into the Value box to this method. Click on the Invoke button run this method.

The running result is shown in Figure 9.104.

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.

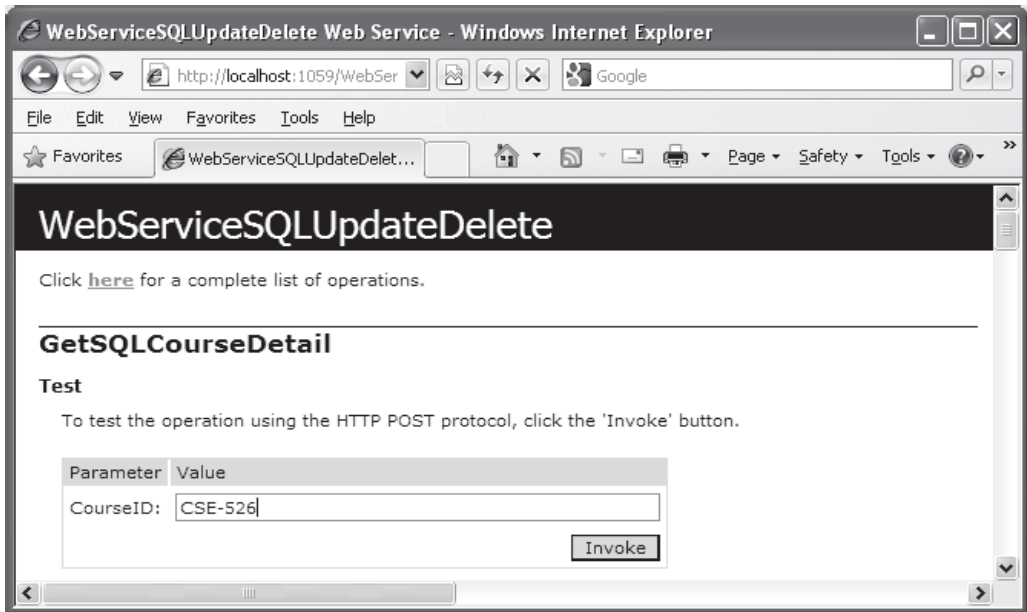


Figure 9.102. The parameter-input Web interface.

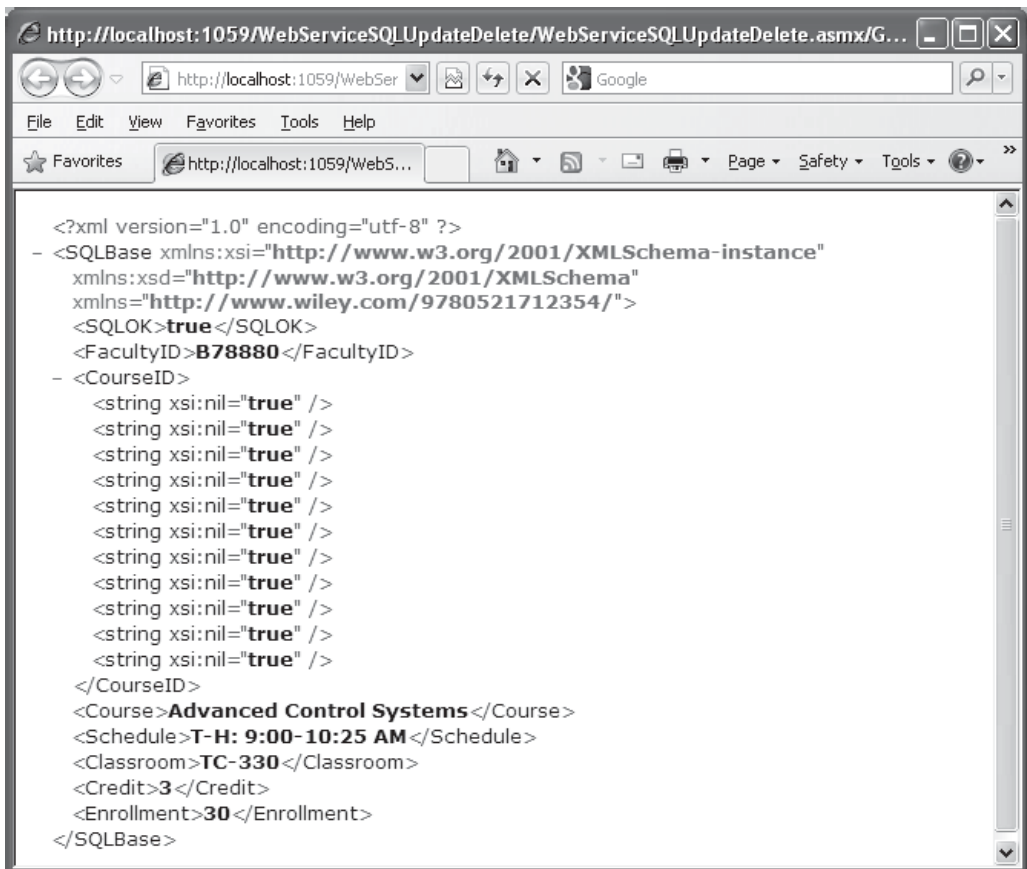


Figure 9.103. The running result of the Web method GetSQLCourseDetail().

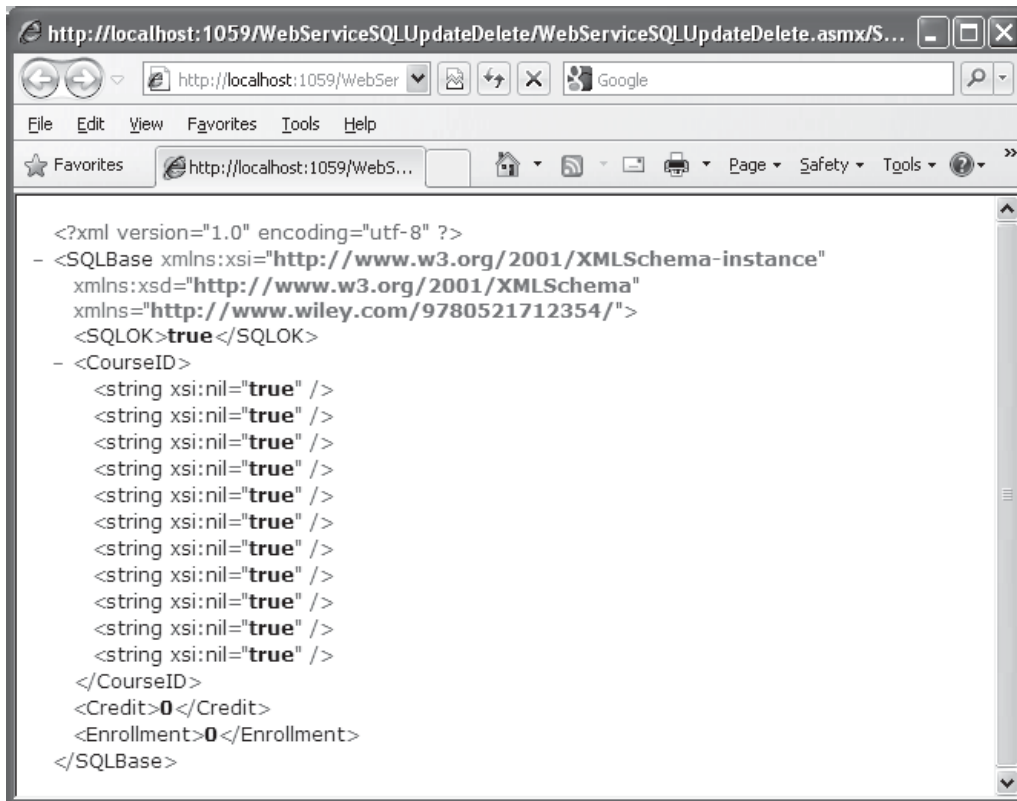


Figure 9.104. The running result of the Web method SQLDeleteSP().

To confirm this data deleting, close the current running result interface and click on the Back arrow 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. 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 9.105.

From this running result shown in Figure 9.105, it can be found that the course with the course_id CSE-526 has been deleted from the Course table since that course is taught by the faculty Ying Bai.

To get a more clear picture for this data deleting, let's try to run another Web method GetSQLCourseDetail(). Close the current running result interface and click the Back arrow to return to the home page. Select and click on 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 that 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 a message box with a message "No matched course found" is shown up. This means that the queried course has been deleted from the Course table and it cannot be found from that table again.

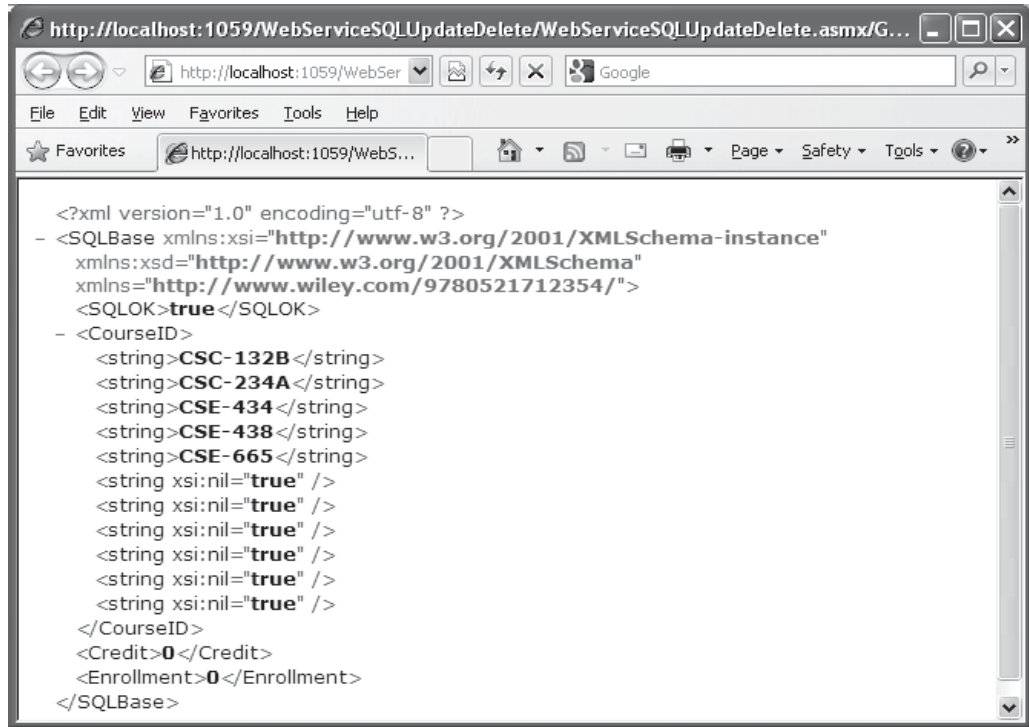


Figure 9.105. The running result of the Web method GetSQLCourse().

Click on the OK button to the message box to close it, and the running result is displayed, which is shown in Figure 9.106. 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 Studio, and add all columns shown in Table 9.6 for the deleted course CSE-526 into our Course table.

You can remove all message box functions `MsgBox()` 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 in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Next, let's take care of building some Windows-based and Web-based client projects to consume this Web service.

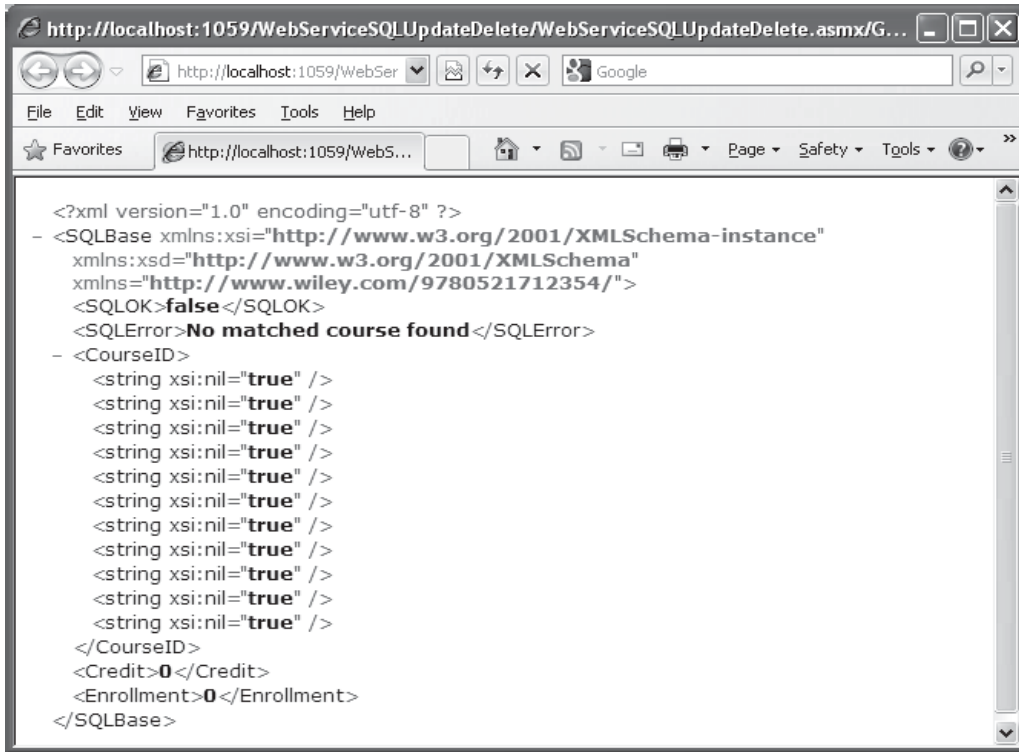


Figure 9.106. The running result of the Web method GetSQLCourseDetail().

9.6 BUILD WINDOWS-BASED WEB SERVICE CLIENTS TO CONSUME THE WEB SERVICES

To save the time and the space, we do not need to create any new project and perform a full development. Instead, we can copy and modify an existing Windows-based client project WinClientSQLInsert we developed in Section 9.4.4 in this chapter to make it as our new client project WinClientSQLUpdateDelete. To do that, create a new folder Chapter 9 at our root directory if you have not done that. Copy this client project from the folder DBProjects\Chapter 9 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1) and paste it into our new folder C:\Chapter 9. Rename the copied project to WinClientSQLUpdateDelete.

Now let's perform the necessary modifications to this project to make it as our new project. The modifications can be divided into three parts:

1. Modifications to the file folder and the project files.
2. Add a new Web reference to our new client project.
3. Modifications to the codes in the code window.

First, let's perform the modifications to the first part.

9.6.1 Modifications to the File Folder and Project Files

Open the Windows Explorer and browse to our new project folder WinClientSQLUpdateDelete that is located at the folder C:\Chapter 9. Perform the following modifications to this project:

1. Rename the project folder from WinClientSQLInsert to WinClientSQLUpdateDelete.
2. Rename the project file from WinClientSQLInsert.vbproj to WinClientSQLUpdateDelete.vbproj.
3. Double-click on the project file WinClientSQLUpdateDelete.vbproj to open the project. On the opened project window, click on our form WinClient Form.vb from the Solution Explorer window and go to Project!WinClientSQLUpdateDelete Properties menu item to open the project property wizard. Perform the following modifications:
 - a. Change the Assembly name to WinClientSQLUpdateDelete.
 - b. Change the Root namespace to WinClientSQLUpdateDelete.
 - c. Click on the Assembly Information button to open the associated wizard. Change the Title and the Product to WinClientSQLUpdateDelete. Click on the OK button to close this wizard.
 - d. Click on the Start Debugging button to run the project to make these modifications updated, and then click on the Back button to terminate the project.
4. Reopen Windows Explorer, browse to our new project folder WinClientSQLUpdateDelete\bin\Debug, and remove all old project files that have an old name WinClientSQLInsert with extensions, such as .exe, .pdb, .config, and .xml.
5. Go to the subfolder WinClientSQLUpdateDelete\obj\x86\Debug and remove all old resource files with the name of WinClientSQLInsert followed with extensions, such as resources and Cache.
6. Remove the Web Reference folder from both Windows Explorer and Solution Explorer windows. To remove the Web Reference folder from Solution Explorer window, one needs first to delete the Web reference object and then the folder.

Go to the File!Save All menu item to save these modifications.

9.6.2 Add a New Web Reference to Our Client Project

To consume or use the Web service WebServiceSQLUpdateDelete we developed in the last section, we need first to set up a Web reference to connect to that Web service with our client project together. Perform the following operations to add this Web reference:

1. Open our Web service project WebServiceSQLUpdateDelete, and click on the Start Debugging button to run it.
2. Copy the URL from the Address bar in our running Web service project.
3. Then open another Visual Studio.NET 2010 and our Windows-based client project WinClientSQLUpdateDelete.

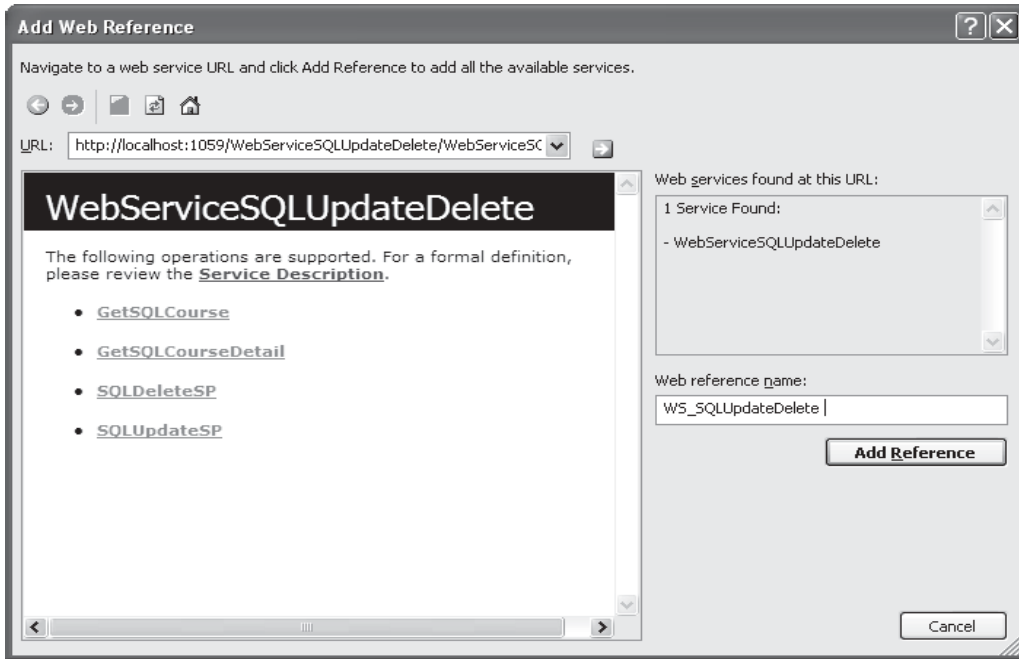


Figure 9.107. The opened Add Web Reference wizard.

4. Right-click on our client project `WinClientSQLUpdateDelete` from the Solution Explorer window, and select the item **Add Service Reference** from the pop-up menu to open the Add Service Reference wizard.
5. Click on the **Advanced** button located at the lower-left corner on this wizard to open the Service Reference Settings wizard.
6. Click on the **Add Web Reference** button to open the Add Web Reference wizard, which is shown in Figure 9.107.
7. Paste the URL we copied from step 2 into the URL box in the Add Web Reference wizard, and click on the **Green Arrow** button to enable Visual Studio.NET 2010 to begin to search it.
8. When the Web service is found, the name of our Web service is displayed in the right pane, which is shown in Figure 9.107.
9. Alternately, you can change the name for this Web reference from `localhost` to any meaningful name, such as `WS_SQLUpdateDelete` in our case. Click on the **Add Reference** button to add this Web service as a reference to our new client project.
10. Click on the **Close** button from our Web service built-in Web interface window to terminate our Web service project.

Next, let's modify the codes in the related event procedures and user-defined subroutines to call our Web service to perform the desired data actions.

9.6.3 Modify the Codes for the Different Event Procedures and Subroutines

The modifications to the codes include the following parts:

1. Modify the codes for the `Form_Load` event procedure and form-level variables.
2. Modify the codes for the `Select` button's click event procedure and related subroutines, `ProcessObject()` and `FillCourseListBox()`, to make them perform the data validation after the data updating and deleting actions.
3. Remove the `Insert` button's click event procedure since we do not need this action in this application.
4. Modify the codes for the `SelectedIndexChanged` event procedure of the Course List Box control and related subroutine `FillCourseDetail()` to perform the confirmation for the data updating actions.
5. Develop the codes for the `Update` button's click event procedure to perform the data updating actions.
6. Develop the codes for the `Delete` button's click event procedure to perform the data deleting actions.

Let's perform these modifications starting from the first part.

9.6.3.1 Modify the Codes of the Form_Load Event Procedure and Form-Level Variables

Perform the following modifications to this part:

1. Change the class name of the form-level instance from `WS_SQLInsert.SQLInsertBase` to `WS_SQLUpdateDelete.SQLBase`.
2. Remove the second method `DataSet Method` from the `Form_Load` event procedure.

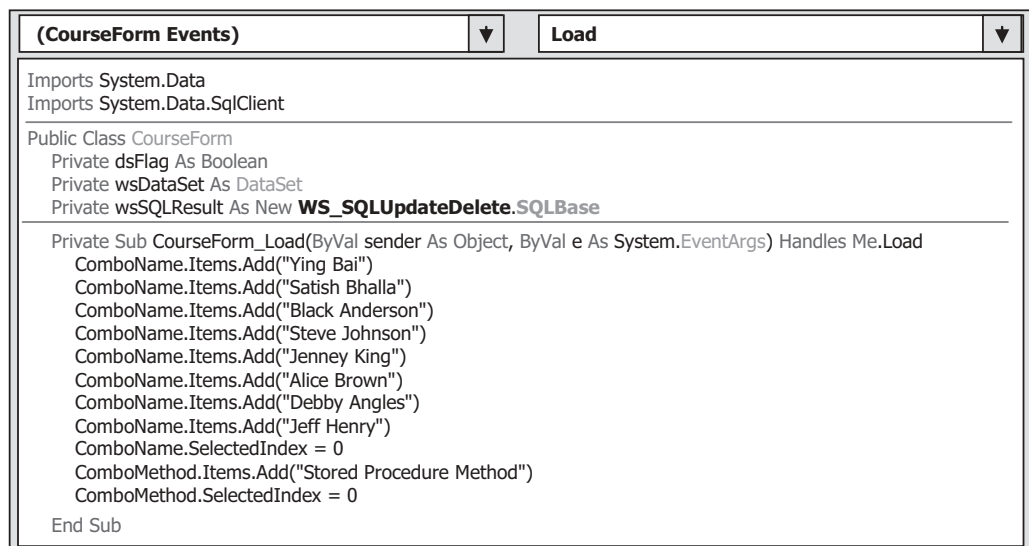


Figure 9.108. The modified `Form_Load` event procedure.

Your modified `Form_Load` event procedure and form-level variables should match those that are shown in Figure 9.108. The modified parts have been highlighted in bold.

Next, let's modify the codes in the **Select** button's click event procedure and related user-defined subroutine procedures to perform the validation functions for our data updating and deleting actions.

9.6.3.2 Modify the Codes for the Select Button Click Event Procedure and Related User-defined Subroutine Procedures

The function of this event procedure 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 member should be selected from the Faculty Name combo box control, and the **Select** button should be clicked by the user. Then this event procedure will call the Web method `GetSQLCourse()` in our Web service project, and an instance that contains all retrieved `course_id` taught by the selected faculty is returned from that Web method. Some related subroutines are executed to extract those `course_id` from the returned instance and display them in the list box control in our client form window.

Open this event procedure and perform the modifications that are shown in Figure 9.109 to this event procedure.

Let's have a closer look at this piece of modified codes 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. Remove the `If . . . Else . . . End If` block for the method-checking process since we have 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 the member data from `SQLInsertError` to `SQLError`.

All modification parts have been highlighted in bold.

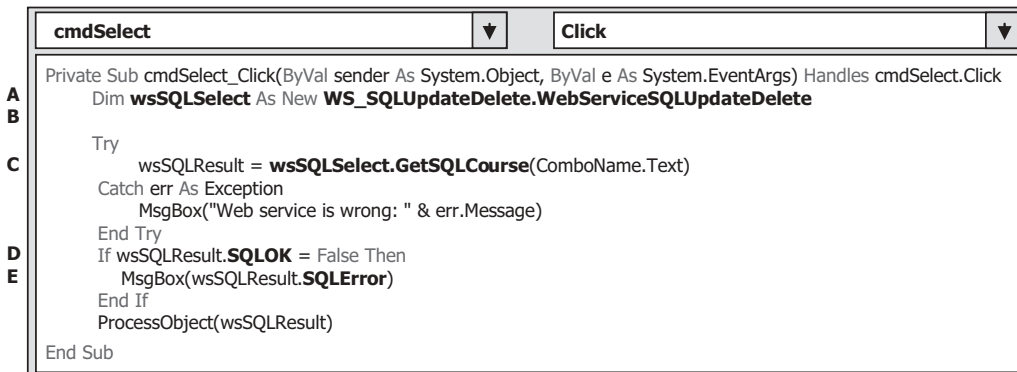


Figure 9.109. The modified codes for the Select button event procedure.

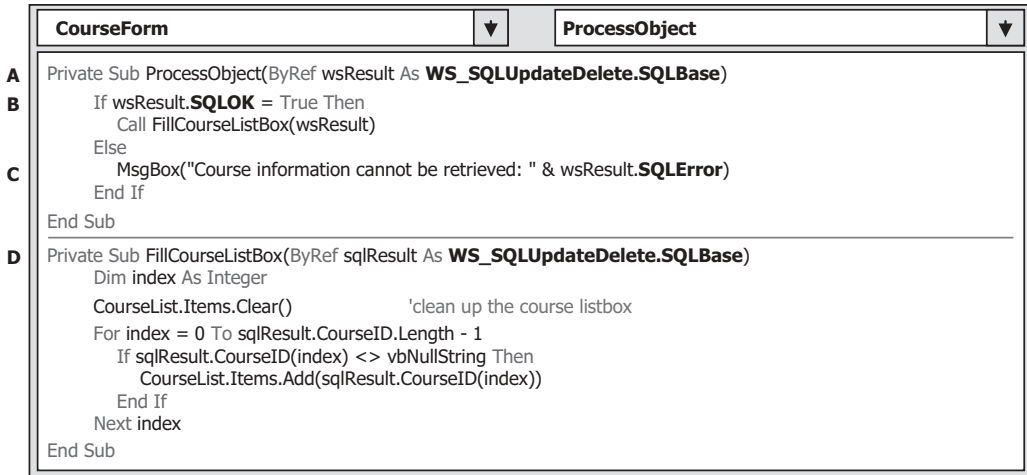


Figure 9.110. The modified subroutines ProcessObject() and FillCourseListBox().

Two user-defined subroutines are associated with this **Select** button's click event procedure, and they are: **ProcessObject()** and **FillCourseListBox()**. The modifications to these two subroutines include the following steps (refer to Figure 9.110):

- 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 subroutines are shown in Figure 9.110, and the modified parts have been highlighted in bold.

9.6.3.3 Remove the Insert Button Click Event Procedure

Since we do not need this data action in this application, we can remove this procedure from our current project. Select all codes of this event procedure, including the procedure header and ender, and press the **Delete** key from your keyboard to remove this entire event procedure.

9.6.3.4 Modify the Codes for the SelectedIndexChanged Event Procedure

Open the **SelectedIndexChanged** event procedure of the **CourseList** control and perform the modifications that are shown in Figure 9.111 to this event procedure. The modified parts have been highlighted in bold.

Let's take a closer look at this piece of modified codes 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**.

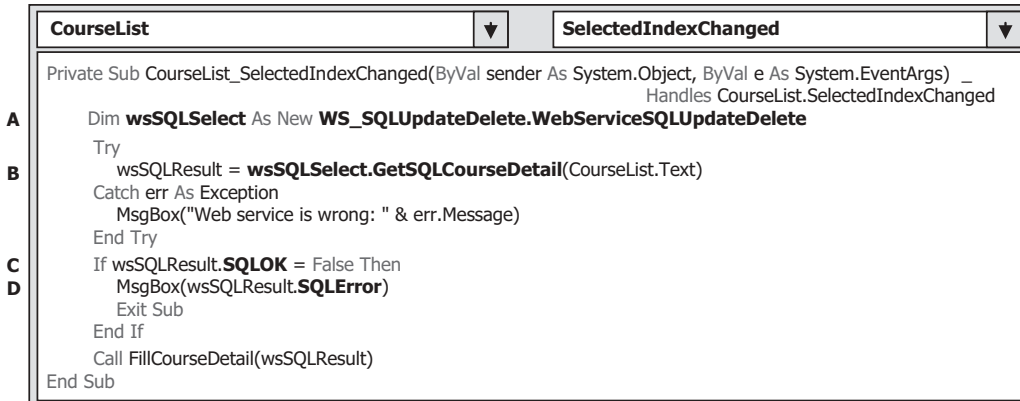


Figure 9.111. The modified codes for the SelectedIndexChanged event procedure.

- B.** Change the instance name of our Web proxy class from `wsSQLInsert` to `wsSQLSelect`, and the 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 subroutine `FillCourseDetail()` is to change the data type of the argument from `WS_SQLInsert.SQLInsertBase` to `WS_SQLUpdateDelete.SQLBase`.

Next, let's concentrate on the code development for the `Update` button's click event procedure.

9.6.3.5 Develop the Codes for the Update Button Event Procedure

The function of this event procedure is: when a faculty name is selected and all six pieces of updated course information are entered in the six-textbox controls, the `Update` button is clicked by the user. The updated course information will be passed to the Web method `SQLUpdateSP()` in our Web service project, and the SQL Server stored procedure `WebUpdateCourseSP()` is executed to perform this course updating.

Now let's double click on the `Update` button to open its click event procedure, and enter the codes that are shown in Figure 9.112 into this event procedure.

Let's take a closer look at this piece of codes 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 methods we developed in our Web service class `WebServiceSQLUpdateDelete`.
- B.** A `Try . . . Catch` block is used to call the Web method `SQLUpdateSP()` with six pieces of course updated information to execute a stored procedure `WebUpdateCourseSP()` to perform this course updating action against our sample database.
- C.** An error message will be displayed if any error is encountered during that data updating action.
- D.** Besides the system error-checking methods, we also need to check the member data `SQLOK` defined in our base class in the Web service project to make sure that this data

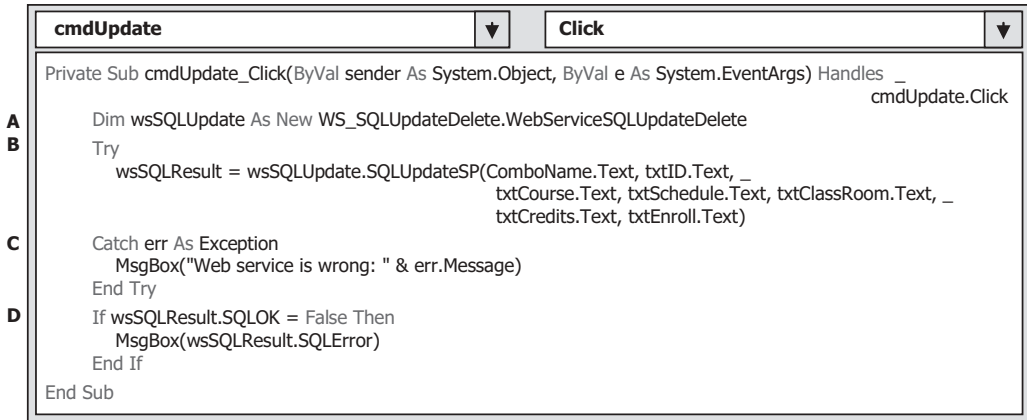


Figure 9.112. The codes for the Update button click event procedure.

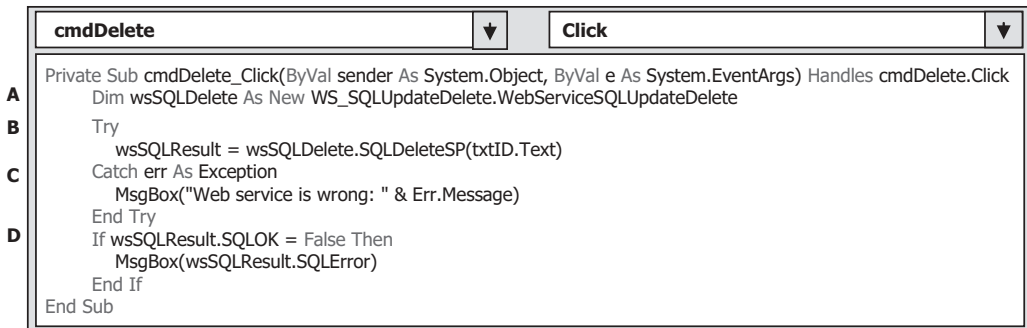


Figure 9.113. The codes for the Delete button click event procedure.

updating is application-error free. A returned **False** indicates that this data updating encountered some application errors, and the error source stored in another member data **SQLError** is displayed.

It looks like that this piece of codes is very simple—yes, it is! As long as the Web service is developed and is ready to be used, developing some client projects to consume that Web service is very simple and easy.

Similarly, we can develop the codes for the **Delete** button's click event procedure to perform the data deleting actions against our sample database.

9.6.3.6 Develop the Codes for the Delete Button Event Procedure

The function of this event procedure is: when a **course_id** has been selected from the **Course ID** textbox control in this client form 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, in our sample relational database.

Double-click on the **Delete** button from our client form window to open the **Delete** button Click event procedure, and enter the codes that are shown in Figure 9.113 into this event procedure.

Let's take a closer look at this piece of codes 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 methods we developed in our Web service class `WebServiceSQLUpdateDelete`.
- B. 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 against our sample database.
- C. An error message will be displayed if any error is encountered during that data deleting action.
- D. Besides the system error-checking methods, 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` indicates that this data deleting encountered some application errors, 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.

At this point, we have finished all modifications to this client project and now it is the 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 running status. This can be identified by a small white icon located in the status bar on 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 one time, you can close our Web service page. However, the small white icon should be still in there, and this means that our Web service is still running and ready to be accessed and consumed.

Now click on the **Start Debugging** button from our client project to run it. First, let's test the data updating function by updating a course record **CSE-665**. 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 (`course_id`) currently taught by the selected faculty member **Ying Bai**. All `course_id` will be retrieved and displayed in the list box control. Click on the `course_id` **CSE-665** from the list box control, and immediately, the detailed information related to that course is displayed in the associated textbox control, which is shown in Figure 9.114.

Now enter the updating information for the course **CSE-665** into the associated textbox, which is shown in Figure 9.115.

Click on the **Update** button to call the Web method `SQLUpdateSP()` in our Web service project to update this course record.

To confirm this data updating, click on the **Select** button to try to retrieve all courses taught by the selected faculty **Ying Bai**. All `course_id` taught by that faculty are returned and displayed in the list box control, which is shown in Figure 9.116.

To check whether the course **CSE-665** has been updated or not, first, let's select another `course_id` from the list box, such as **CSC-132B**, and then click on the `course_id` **CSE-665** from the list box control. Immediately, the detailed information about this updated course is displayed in the associated textbox, as shown in Figure 9.116. It can be found that this course has been updated successfully.

CSE DEPT Course Form

Faculty Name _Query Method

Faculty Name Ying Bai

Query Method Stored Procedure Method

Course List

- CSC-132B
- CSC-234A
- CSE-434
- CSE-438
- CSE-665**

Course Information

Course ID CSE-665

Course Advanced Fuzzy Systems

Schedule T-H: 1:00-2:25 PM

Classroom TC-315

Credits 3

Enrollment 26

Select **Insert** **Update** **Delete** **Back**

Figure 9.114. The detailed information of the course CSE-665.

CSE DEPT Course Form

Faculty Name _Query Method

Faculty Name Ying Bai

Query Method Stored Procedure Method

Course List

- CSC-132B
- CSC-234A
- CSE-434
- CSE-438
- CSE-665**

Course Information

Course ID CSE-665

Course Neural Networks

Schedule T-H: 9:00-10:25 AM

Classroom TC-330

Credits 3

Enrollment 30

Select **Insert** **Update** **Delete** **Back**

Figure 9.115. The updating information for the course CSE-665.

To test the deleting function, keep the course **CSE-665** selected from the list box and click on the **Delete** button to try 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 `course_id` are returned and displayed in the list box control. It can be found that the course **CSE-665** has been removed from the Course table, and you cannot find it from the list box now.

Click on the **Back** button to terminate our client project. Our client project is very successful.

Figure 9.116. The updated course CSE-665.

However, the story is not finished. 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. Using the Server Explorer window in Visual Studio.NET to open our sample database **CSE_DEPT.mdf** and our Course data table.
2. Using the Microsoft SQL Server Management Studio or Studio Express to open our sample database **CSE_DEPT.mdf** and our Course data table.
3. Using our Web service project **WebServiceSQLInsert** to insert a new course to perform this course recovering.
4. Using our Windows-based Web service client project **WinClientSQLInsert** to perform this course recovering.
5. Using 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 record is professional since regularly, no one wants to access and change the content of a database directly by opening the database to do modifications.

A complete Windows-based Web service client project **WinClientSQLUpdateDelete** can be found in the folder **DBProjects\Chapter 9** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

9.7 BUILD WEB-BASED WEB SERVICE CLIENTS TO CONSUME THE WEB SERVICES

There is no significant difference between building a Windows-based client project and building a Web-based client project to consume 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 as our new Web-based client project `WebClientSQLUpdateDelete`.

In fact, we can copy and rename that entire project as our new Web-based client project. But 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.
3. Modify the codes in the related event procedures of the `Course.aspx.vb` file to call the associated Web method to perform our data updating and deleting. The code modifications include the following sections:
 - A. Modify the codes in the `Page_Load` event procedure.
 - B. Modify the codes in the `Select` button's click event procedure and the related subroutines, `ProcessObject()` and `FillCourseListBox()`.
 - C. Modify the codes in the `SelectedIndexChanged` event procedure of the course list box control and the related subroutine `FillCourseDetail()`.
 - D. Remove the `Insert` button's click event procedure `cmdInsert_Click()` since we do not need any data insertion action in this application.
 - E. Remove the `TextChanged` event procedure of the `Course ID` textbox since we do not need this event and its event procedure in this application.
 - F. Develop the codes for the `Update` button's click event procedure.
 - G. Develop the codes for the `Delete` button's click event procedure.

Now let's start with the first step listed above.

9.7.1 Create a New Web Site 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 Web site project. Enter `C:\Chapter 9\WebClientSQLUpdateDelete` into the `Name` box that is next to the `Web location` box, and click on the `OK` button 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` wizard. Browse to our Web project `WebClientSQLInsert` that can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Double-click on it and select the `Course.aspx` from the list. Click on the `Add` button to add this item into our new Website project.

9.7.2 Add a Web Service Reference and Modify the 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`

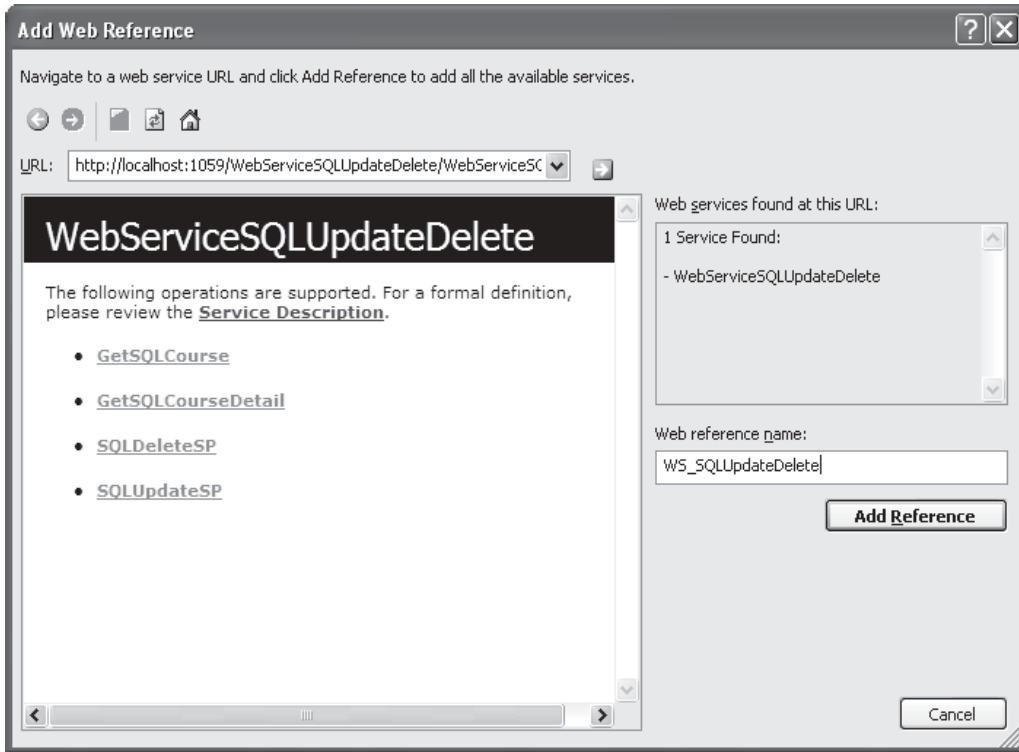


Figure 9.117. The finished Add Web Reference wizard.

Reference from the pop-up menu. Open another Visual Studio.NET and our Web service project `WebServiceSQLUpdateDelete`, 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 wizard. Then click on the green button 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_SQLUpdateDelete`. Your finished Add Web reference wizard should match the one that is shown in Figure 9.117.

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`

Now let's take care of modifications to the codes in related event procedures and subroutines in the `Course.aspx` page.

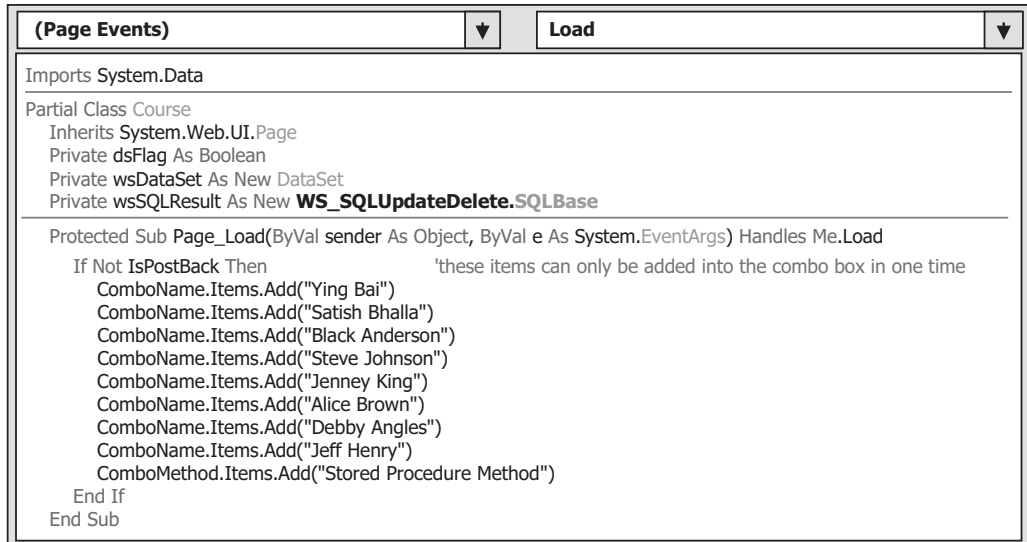


Figure 9.118. The modified Page_Load event procedure.

9.7.3 Modify the Codes for the Related Event Procedures and Subroutines

The first modification is to change the codes in the Page_Load event procedure and modify some form-level variables.

9.7.3.1 Modify the Codes in the Page_Load Event Procedure

Perform the following changes to complete this modification:

1. Change the name of the base class for the form level instance `wsSQLResult` from `WS_SQLInsert.SQLInsertBase` to `WS_SQLUpdateDelete.SQLBase`.
2. In the Page_Load event procedure, remove the code that is used to add and display the second Web method, `DataSet Method`, in the combo box control.

Your modified codes for the Page_Load event procedure should match one that is shown in Figure 9.118. The modified codes have been highlighted in bold.

9.7.3.2 Modify Codes in the Select Button Event Procedure and Related Subroutines

The function of this event procedure 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 combo box control, and the **Select** button should be clicked by the user. Then this event procedure will call the Web method `GetSQLCourse()` in our Web service, and an instance that contains all retrieved `course_id` taught by the

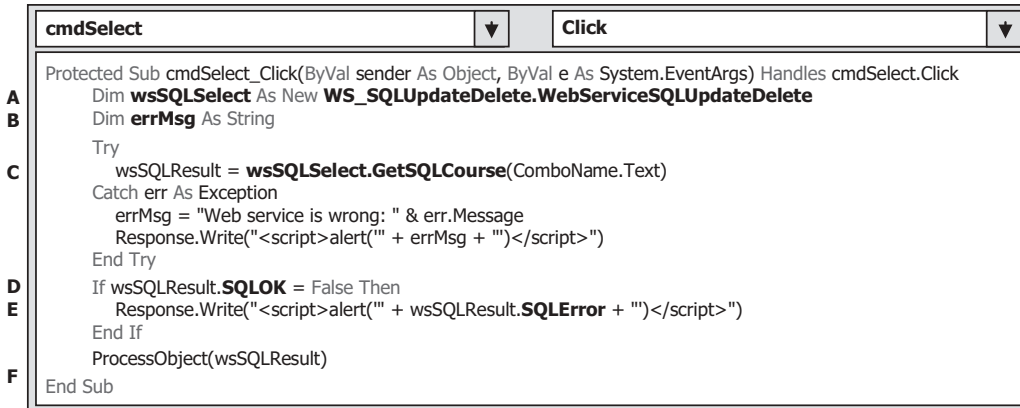


Figure 9.119. The modified codes for the Select button event procedure.

selected faculty is returned from that Web method. Some subroutines are executed to extract those `course_id` from the returned instance and display them in the list box control in our client page window.

Open this event procedure and perform the modifications that are shown in Figure 9.119 to this event procedure. All modification parts have been highlighted in bold.

Let's have a closer look at this piece of modified codes 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 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 the 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 statements: `Call FillCourseDataSet()` and `Application("dsFlag") = False` since we do not need these two operations in this application.

Two user-defined subroutines are associated with this **Select** button's click event procedure, **ProcessObject()** and **FillCourseListBox()**. The modifications to these two subroutines include the following steps:

- A.** Change the data type of passed argument **wsResult** from **WS_SQLInsert.SQLOK** to **WS_SQLUpdateDelete.SQLOK**.
- 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.SQLOK** to **WS_SQLUpdateDelete.SQLOK**.

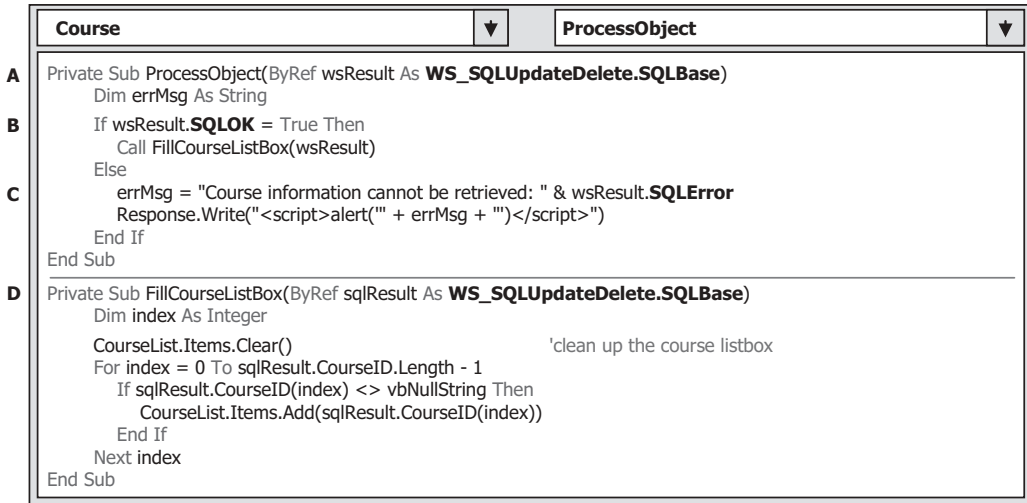


Figure 9.120. The modified subroutines ProcessObject() and FillCourseListBox().

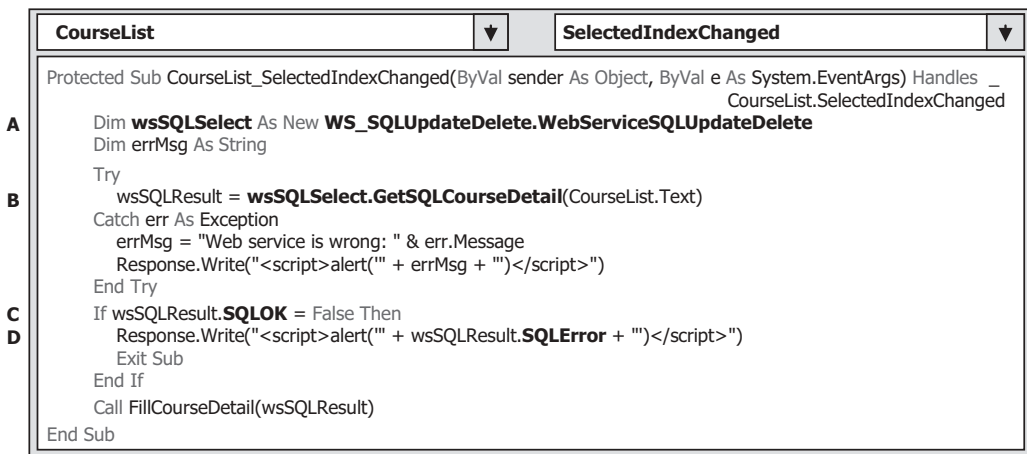


Figure 9.121. The modified codes for the SelectedIndexChanged event procedure.

Two modified subroutines are shown in Figure 9.120, and the modified parts have been highlighted in bold.

Go to File/Save All menu item to save these modifications. Next, we will perform the modifications to another event procedure SelectedIndexChanged, which is the event procedure of the course list box control CourseList in our page window.

9.7.3.3 Modify the Codes in the SelectedIndexChanged Event Procedure of the Course List Box Control and Related Subroutines

Open the SelectedIndexChanged event procedure of the CourseList control and perform the modifications that are shown in Figure 9.121 to this event procedure.

Let's take a closer look at this piece of modified codes 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.** 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 subroutine `FillCourseDetail()` is to change the data type of the argument from `WS_SQLInsert.SQLInsertBase` to `WS_SQLUpdateDelete.SQLBase`.

9.7.3.4 Remove the Insert Button Click Event Procedure and the TextChanged Event Procedure of the Course ID Textbox

Since we do not need these two event procedures in this project, just select all codes in these two event procedures, including the procedure headers and enders, and press the **Delete** key from the keyboard to delete these procedures from this project.

The next step is to develop the codes for the **Update** button's click event procedure.

9.7.3.5 Develop Codes for the Update Button Click Event Procedure

The function of this event procedure is: when a faculty name is selected and all six pieces of updated course information are entered into the six textbox controls, the updated course information will be passed to the Web method `SQLUpdateSP()` in our Web service project, and a stored procedure `WebUpdateCourseSP()` is executed to perform this course updating action as the **Update** button is clicked by the user. Now let's develop the codes for this event procedure by double-clicking on the **Update** button to open its click event procedure, and enter the codes that are shown in Figure 9.122 into this event procedure.

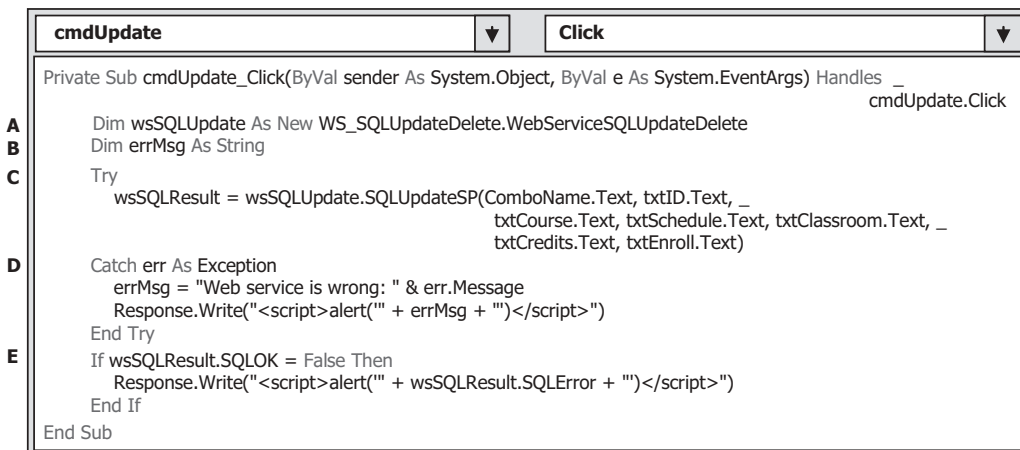


Figure 9.122. The codes for the Update button click event procedure.

Let's take a closer look at this piece of codes 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 `WebServiceSQLUpdateDelete`.
- 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 updated information to execute a stored procedure `WebUpdateCourseSP()` to perform this course updating action against our sample database.
- D.** An error message will be displayed if any error is encountered during that data updating action. A point to be noted is that the display 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 methods, 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` 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 the next section, we will develop the codes for the Delete button's click event procedure to perform the data deleting actions against our sample database.

9.7.3.6 Develop Codes for the Delete Button Click Event Procedure

The function of this event procedure is: when a `course_id` has been selected 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 button click event procedure, and enter the codes that are shown in Figure 9.123 into this event procedure.

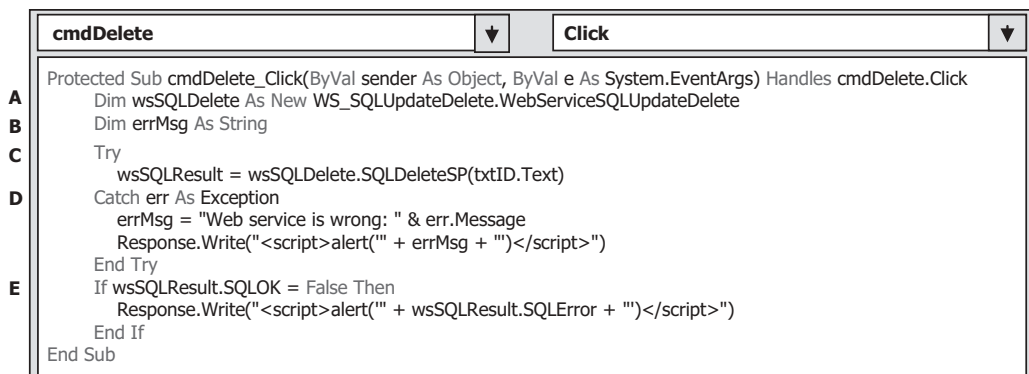


Figure 9.123. The codes for the Delete button click event procedure.

Let's take a closer look at this piece of codes 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 later.
- 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 against our sample database.
- D. An error message will be displayed if any error is encountered during that data deleting action. A point to be noted is that the display 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 methods, 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 **FileSave All** menu item to save these modifications and developments.

At this point, we have finished all modifications to this Web-based client project and now it is the 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 running status. This can be identified by a small white icon located in the status bar on 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 one time, you can close our Web service page by clicking on the Close button. However, the small white icon should still be in there, which means that our Web service is running and ready to be accessed and consumed.

Now click on the Start Debugging button from our client project to run it. First, let's test the data updating function by updating a course record **CSE-665**. 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 `course_id` currently taught by the selected faculty **Ying Bai**. All `course_id` will be retrieved and displayed in the list box control. Click on the course **CSE-665** from the list box control to get the detailed information for this course, which is shown in Figure 9.124.

Now enter the updating information for the course **CSE-665** into the associated textbox, as shown in Figure 9.125.

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, let's select another course from the list box, such as **CSC-234A**, and then click on the course **CSE-665** from the list box control. Immediately, the detailed information about this updated course is displayed in the associated textbox, as shown in Figure 9.126.

The screenshot shows a web browser window titled "Course Page - Windows Internet Explorer". The address bar shows "http://localhost:". The browser has a menu bar (File, Edit, View, Favorites, Tools, Help) and a toolbar with navigation buttons and a search box. Below the browser window is a web form titled "Course Page". The form has two dropdown menus at the top: "Method" set to "Stored Procedure" and "Faculty Name" set to "Ying Bai". On the left is a list box containing course IDs: CSC-132B, CSC-234A, CSE-434, CSE-438, and CSE-665. CSE-665 is selected. To the right of the list box are several input fields: "CourseID" (CSE-665), "Course" (Advanced Fuzzy Systems), "Schedule" (T-H: 1:00-2:25 PM), "Classroom" (TC-315), "Credit" (3), and "Enrollment" (26). At the bottom of the form are five buttons: "Select", "Insert", "Update", "Delete", and "Back".

Figure 9.124. The detailed information of the course CSE-665.

The screenshot shows the same web browser window and form as Figure 9.124. The "Method" dropdown is still "Stored Procedure" and "Faculty Name" is "Ying Bai". The list box on the left still has CSE-665 selected. However, the input fields on the right have been updated: "CourseID" is CSE-665, "Course" is Neural Networks, "Schedule" is T-H: 9:00-10:25 AM, "Classroom" is TC-330, "Credit" is 3, and "Enrollment" is 30. The buttons at the bottom remain "Select", "Insert", "Update", "Delete", and "Back".

Figure 9.125. The updating information for the course CSE-665.

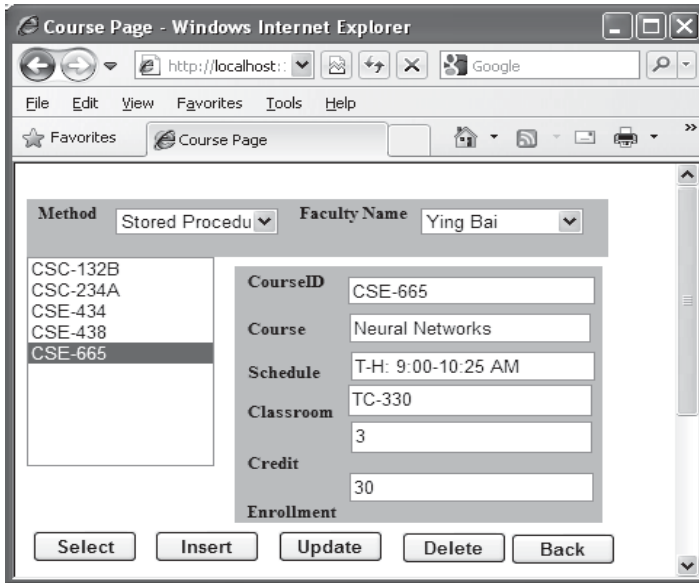


Figure 9.126. The updated course CSE-665.

It can be found that this course has been updated with our updating information successfully.

To test the deleting function, keep the course **CSE-665** selected from the list box, click on the **Delete** button to try 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 (course_id) taught by the selected faculty. Immediately, all course_id are returned and displayed in the list box control. It can be found that the course **CSE-665** has been removed from the Course table, and you cannot find it from the list box now.

Click on the **Back** button to terminate our client project. Our client project is very successful.

However, the story is not finished. It is highly recommended to recover that deleted course **CSE-665** for our Course table since we want to keep our database neat and complete. You can recover this record by using one of the following five methods:

1. Using the Server Explorer window in Visual Studio.NET to open our sample database **CSE_DEPT.mdf** and our Course data table.
2. Using the Microsoft SQL Server Management Studio or Studio Express to open our sample database **CSE_DEPT.mdf** and our Course data table.
3. Using our Web service project **WebServiceSQLInsert** to insert the course **CSE-665** to perform this course recovering.
4. Using our Windows-based Web service client project **WinClientSQLInsert** to perform this course recovering.
5. Using our Web-based Web service client project **WebClientSQLInsert** to insert this course to recover this course record.

Table 9.7. The recovered course record for CSE-665

Column Name	Column Value
course_id	CSE-665
course	Advanced Fuzzy Systems
credit	3
classroom	TC-315
schedule	T-H: 1:00-2:25 PM
enrollment	26
faculty_id	B78880

Relatively speaking, using the last three methods to recover this course record is professional since regularly, no one wants to access and change the content of a database directly by opening the database to do any modifications.

Refer to Table 9.7 to recover this course record.

A complete Web-based Web service client project `WebClientSQLUpdateDelete` can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

At this point, we have finished the discussion about how to access and manipulate data against the SQL Server database via ASP.NET Web services. In the next section, we will discuss how to access and manipulate data against 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 of building an ASP.NET Web service to access the SQL Server 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 distinguished points exist between the procedures to build a Web service to access two kinds of databases. Let's give a little more detailed discussion for these issues and have a closer look at those issues.

First, when connecting to the 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 of 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. These Data Providers are database-dependent, which means that a different Data Provider is needed to use to access the different database. 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 ADO.NET to access an Oracle database, one must import 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 a 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 database. Also the nominal name of the dynamic parameter is distinguished when it is used for the different databases.

Based on the discussion and analysis above, 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 we built in the previous sections. In this part, we concentrate on the modifications to the Web service project **WebServiceSQLSelect** and make it as our new service project **WebServiceOracleSelect**.

9.8.1 Build a Web Service Project **WebServiceOracleSelect**

Open the Windows Explorer and create a new folder **Chapter 9** under the root directory if you have not done that. Then browse to our desired source Web service project **WebServiceSQLSelect** that can be found in the folder **DBProjects\Chapter 9** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy and paste it into our new folder **C:\Chapter 9**. Then rename this copied project to **WebServiceOracleSelect**. In the opened Windows Explorer window, perform the following modifications to this project to make it as our new Web service 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.vb** to **OracleSelectBase.vb**.
3. Open the **App_Code** folder and change the name of our derived class file from **SQLSelectResult.vb** to **OracleSelectResult.vb**.
4. Open the **App_Code** folder and change the name of our code-behind page from **WebServiceSQLSelect.vb** to **WebServiceOracleSelect.vb**.

Now open Visual Studio.NET 2010 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.vb`" to `CodeBehind` = "`~/App_Code/WebServiceOracleSelect.vb`"
- Change `Class` = "`WebServiceSQLSelect`" to `Class` = "`WebServiceOracleSelect`"

Second, open the base class file `OracleSelectBase.vb` and perform the following modifications:

- Change the class 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.vb` by double-clicking on it from the Solution Explorer window, and perform the following modifications:

- Change the class name from `SQLSelectResult` to `OracleSelectResult`.
- Change the base class name (after the keyword `Inherits`) from `SQLSelectBase` to `OracleSelectBase`.

9.8.2 Modify the 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 installing the Oracle Database 11g Express Edition in the local computer. In our case, the password we utilized is reback.

9.8.3 Add Oracle Database References and Modify the Namespace Directories

First, we need to add an Oracle Data Provider reference to our Web service project. As you know, starting from .NET Framework 4.0, Microsoft no longer supports Oracle database-related operations. Therefore, we need to use an Oracle database driver provided by a third-party vendor. As we discussed in Section 5.20.3 in Chapter 5, we utilized a third-party product, dotConnect for Oracle 6.30 Express, developed by Devart™ Inc.

Now let's add some Oracle Data Provider references to our project. Perform the following operations to complete this addition operation:

1. Right-click on our project **Chapter 9\WebServiceOracleSelect** from the Solution Explorer window and select the **Add Reference** item from the pop-up menu to open the Add reference wizard.
2. With the **.NET** tab selected, scroll down the list until you find the items **Devart.Data** and **Devart.Data.Oracle**, then click on both to select them and click on the **OK** button to add these two references to our project.

Now double-click our code-behind page **WebServiceOracleSelect.vb** to open it. On the opened page, add two namespaces shown below to the top of this page:

```
Imports Devart.Data
Imports Devart.Data.Oracle
```

Also change the name of our Web service class, which is located after the accessing mode **Public Class**, from **WebServiceSQLSelect** to **WebServiceOracleSelect**.

Next, we will perform the necessary modifications to three Web methods and related five differences listed above.

9.8.4 Modify the Web Method GetSQLSelect and Related Subroutines

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 subroutines **SQLConn()** and **ReportError()**.
5. The name of the dynamic parameter.

Let's perform those modifications step by step according to this sequence.

Open this Web method and perform the modifications shown in Figure 9.127 to this method. Let's have a closer look at this piece of modified codes to see how it works.

- A. Rename this Web method to **GetOracleSelect** and the name of returned class to **OracleSelectResult**.
- B. Modify the query string by replacing the **LIKE @** comparator before the dynamic parameter **facultyName** with the comparator **=**, which is a comparison operator used in the Oracle database.
- 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**, and change the derived class name from **SQLSelectResult** to **OracleSelectResult**.
- D. Change the name of the returned instance from **SQLResult** to **OracleResult**, and member data from **SQLRequestOK** to **OracleRequestOK**.
- 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 name of the dynamic parameter by removing the **@** symbol before the nominal name **facultyName**. Also, change its data type from **SqlDbType.Text** to **OracleDbType.VarChar**.

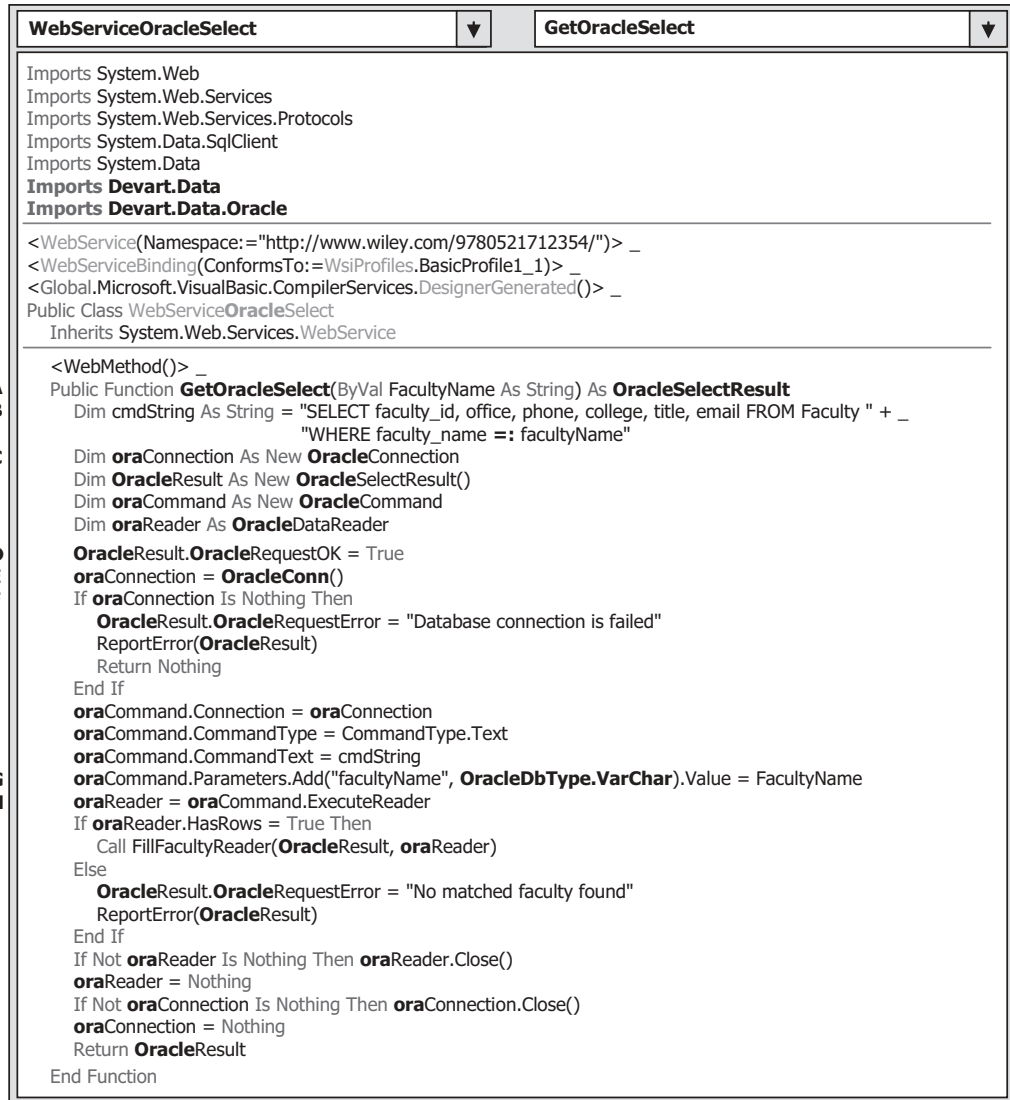


Figure 9.127. The modified Web method GetOracleSelect().

- H.** Change the name of the returned instance from **SQLResult** to **OracleResult**, and change the prefix from **sql** to **ora** for all data objects.

Now let's perform the modifications to three related subroutines. Perform the following modifications to the subroutines **SQLConn()**, **FillFacultyReader()**, and **ReportError()**:

- A.** Change the name of this subroutine 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**.

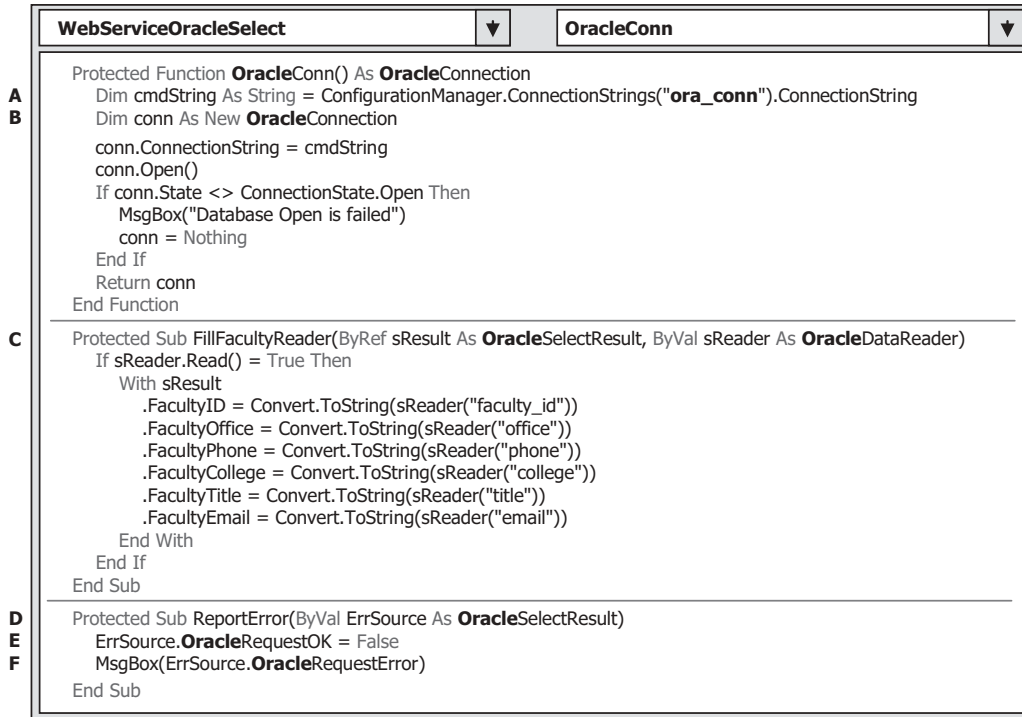


Figure 9.128. Three modified user-defined subroutines.

- C.** For the subroutine **FillFacultyReader()**, 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**. Perform the following modifications to the subroutine **ReportError()**:
- 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**.

Your modified subroutines **OracleConn()**, **FillFacultyReader()**, and **ReportError()** are shown in Figure 9.128. All modified parts have been highlighted in bold.

9.8.5 Modify the Web Method **GetSQLSelectSP** and Related Subroutines

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 with 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 the Stored Procedure *WebSelectFacultySP*

Basically, the modifications to this stored procedure are 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 is to build a Package in the Oracle database since a stored procedure developed in the Oracle database won't return any data. Refer to Section 5.20.7.2 in Chapter 5 to get more detailed discussions about building and developing a Package in 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 11g Express Edition (XE) to build this Package.

Open the Oracle Database 11g XE home page by going to the **start|All Programs|Oracle Database 11g Express Edition|Get Started** items. Perform the following operations to create this package:

1. Click on the **APEX** button to open the Login to APEX page.
2. Enter **SYSTEM** and reback into the Username and Password box to complete the login process for the APEX.
3. Since we have already created our sample database CSE_DEPT in Chapter 2, click on the **Already have an account? Login Here** button.
4. Enter reback into the Password box and click on the **Login** button.
5. Click on the **SQL Workshop** icon to open this workshop window.
6. Click on the **Object Browser** icon and click on the drop-down arrow on the **Create** button, and select the **Package** item to open the Create Package wizard, which is shown in Figure 9.129.

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 9.130.

Enter the package name, **WebSelectFaculty**, into the Package Name box, and click on the **Next** button to go to the specification page, which is shown in Figure 9.131.

A default package specification prototype, which includes a procedure and a function, is provided in this page. 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**. Enter the codes shown in Figure 9.132 into the Specification part as the definition for our package.

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 Oracle database programming.

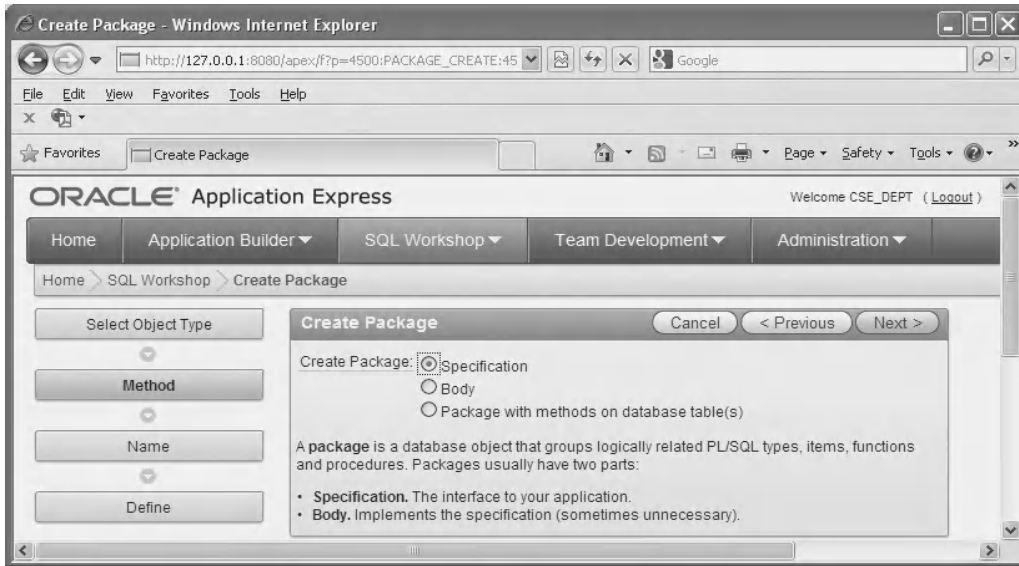


Figure 9.129. The opened Create Package wizard.

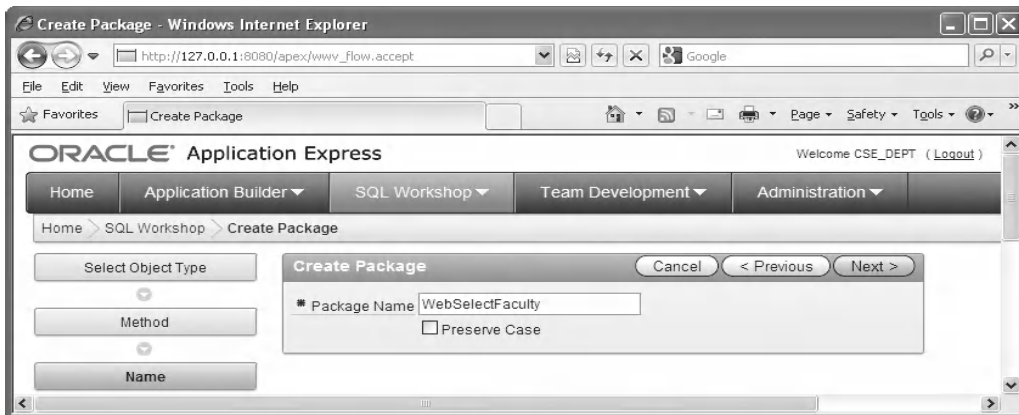


Figure 9.130. The Name page of the Package wizard.

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 be ended by a semi-colon, and this rule is also applied to the `end` statement. Your finished

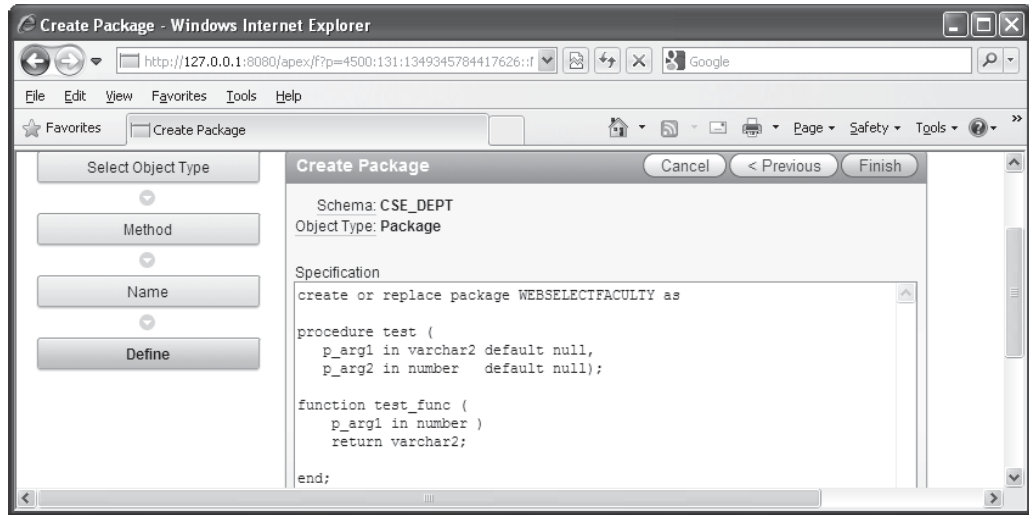


Figure 9.131. The opened Specification page.

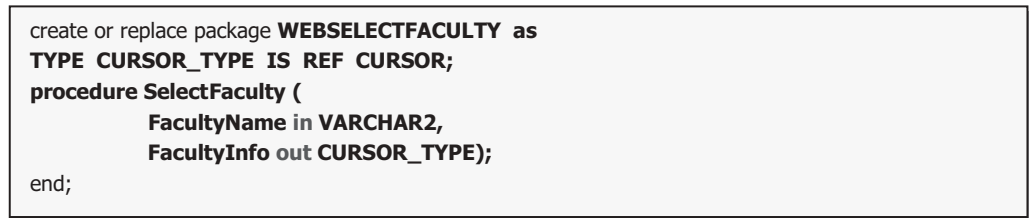


Figure 9.132. The codes for the Specification page.

Specification page should match the one that is shown in Figure 9.133. Click on the **Finish** button to complete this step.

Click on the **Body** tab to open the Body page. Then click on the **Edit** button to begin to create our body part. Enter the PL-SQL codes shown in Figure 9.134 into this body. Your finished body part for this package is shown in Figure 9.135.

The procedure prototype is redeclared in line 2. Starting from **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 5.19.8.4 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.

Ok, now let's compile our package by clicking on the **Save & Compile** button. A successful compiling message **PL/SQL code successfully compiled (10:56:50)** is displayed if this package is bug-free, which is shown in Figure 9.136.

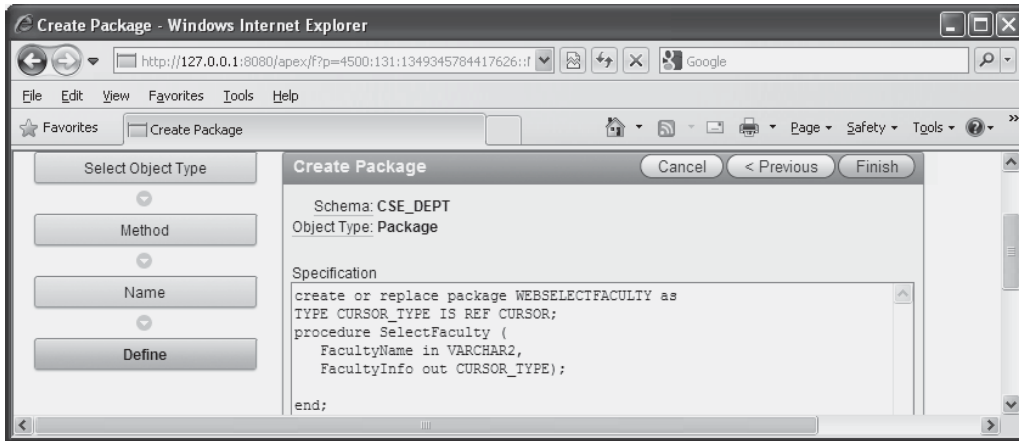


Figure 9.133. The finished Specification page.

```
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 9.134. The body part of the package WebSelectFaculty.

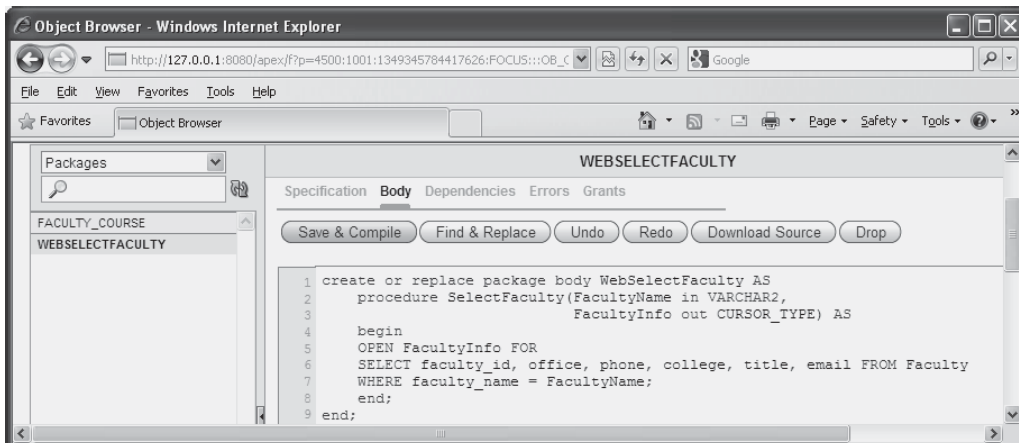


Figure 9.135. The finished Body part of the package.

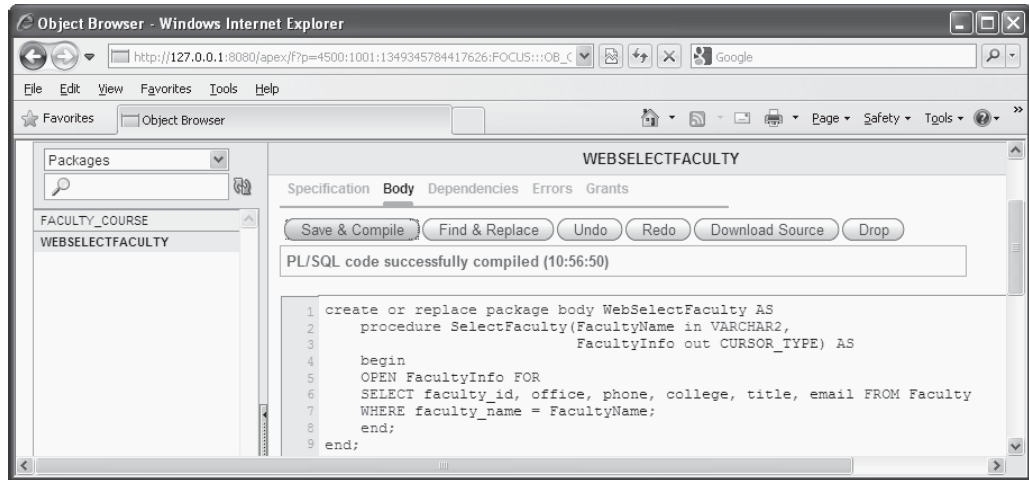


Figure 9.136. The compiled codes for the body part of the package.

The development of our Oracle package is completed, and now let's return to the Visual Studio.NET to call this package to perform our faculty data query for our Web service project. Close the Oracle Database 11g XE and open the Visual Studio.NET.

9.8.5.2 Modifications to the Codes in the 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 the modifications shown in Figure 9.137 to this method. The modified parts have been highlighted in bold.

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

- A. Rename this Web method to **GetOracleSelectSP** and the name of returned class to **OracleSelectResult**.
- B. 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 that is embedded in an Oracle Package will be called to perform this faculty data query as the project runs.
- 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**; change the derived class name from **SQLSelectResult** to **OracleSelectResult**.

WebServiceOracleSelect	GetOracleSelectSP
------------------------	-------------------

```

<WebMethod()> _
A Public Function GetOracleSelectSP(ByVal FacultyName As String) As OracleSelectResult
B     Dim cmdString As String = "WebSelectFaculty.SelectFaculty"
C     Dim oraConnection As New OracleConnection
        Dim OracleResult As New OracleSelectResult()
        Dim oraCommand As New OracleCommand
        Dim oraReader As OracleDataReader
D     Dim paramFacultyName As New OracleParameter
        Dim paramFacultyInfo As New OracleParameter
E     OracleResult.OracleRequestOK = True
F     oraConnection = OracleConn()
        If oraConnection Is Nothing Then
            OracleResult.OracleRequestError = "Database connection is failed"
            ReportError(OracleResult)
            Return Nothing
        End If
G     paramFacultyName.ParameterName = "FacultyName"
        paramFacultyName.OracleDbType = OracleDbType.VarChar
        paramFacultyName.Value = FacultyName
        paramFacultyInfo.ParameterName = "FacultyInfo"
        paramFacultyInfo.OracleDbType = OracleDbType.Cursor
        paramFacultyInfo.Direction = ParameterDirection.Output 'this line is very important
H     oraCommand.Connection = oraConnection
        oraCommand.CommandType = CommandType.StoredProcedure
        oraCommand.CommandText = cmdString
I     oraCommand.Parameters.Add(paramFacultyName)
        oraCommand.Parameters.Add(paramFacultyInfo)
J     oraReader = oraCommand.ExecuteReader
K     If oraReader.HasRows = True Then
        Call FillFacultyReader(OracleResult, oraReader)
        Else
            OracleResult.OracleRequestError = "No matched faculty found"
            ReportError(OracleResult)
        End If
        If Not oraReader Is Nothing Then oraReader.Close()
        oraReader = Nothing
        If Not oraConnection Is Nothing Then oraConnection.Close()
        oraConnection = Nothing
        Return OracleResult
    End Function
  
```

Figure 9.137. The modified Web method GetOracleSelectSP().

- D.** 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.
- 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 subroutine 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. But for the second parameter, as we discussed in step **D**, which is an output parameter with a data type of `Cursor`, and makes our initialization a little complicated. Two points must be noted for this initialization: first, the data type of this parameter must be `OracleDbType.Cursor`,

which is identical with the data type we defined in our stored procedure `SelectFaculty()`. Second, the direction of this parameter must be `Output`, which is the value of the `ParameterDirection` property.

- H. The assignment for the parameter direction property is very important for this output parameter. 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.

9.8.5.3 Modify the 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:

- 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 subroutine `SQLConn()`.
- 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 the one that is shown in Figure 9.138. All modified parts have been highlighted in bold.

Let's take a closer look at this modified Web method to see how it works.

- A. Rename this Web method to `GetOracleSelectDataSet`.
- B. Modify the query string by replacing the `LIKE @` symbol before the dynamic parameter `facultyName` with the symbol `=:`, which is a comparison operator used in the Oracle database.
- 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`, and change the derived class name from `SQLSelectResult` to `OracleSelectResult`.
- D. Change the name of the returned instance from `SQLResult` to `OracleResult`, and member data from `SQLRequestOK` to `OracleRequestOK`.
- 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 name of the dynamic parameter by removing the `@` symbol before the nominal name `facultyName`. Also, change its data type from `SqlDbType.Text` to `OracleDbType.VarChar`.



Figure 9.138. The modified Web method GetOracleSelectDataSet().

- 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 **Yes** on the message box to allow the project to run in the Debug mode.

First, let's test the function 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, 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 9.139.

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 9.139.

Now let's test the next Web method `GetOracleSelectSP()`. To do that, close the running result page by clicking on the Close button that is located at the upper-right corner of this page, and click on the Back arrow on the top to return the initial page. Then

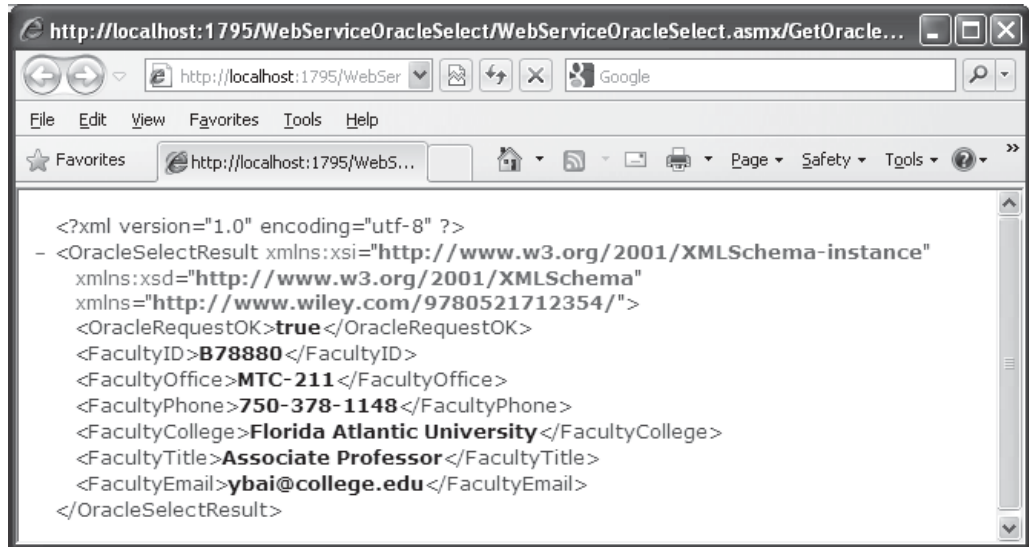


Figure 9.139. The running result of the Web method GetOracleSelect().

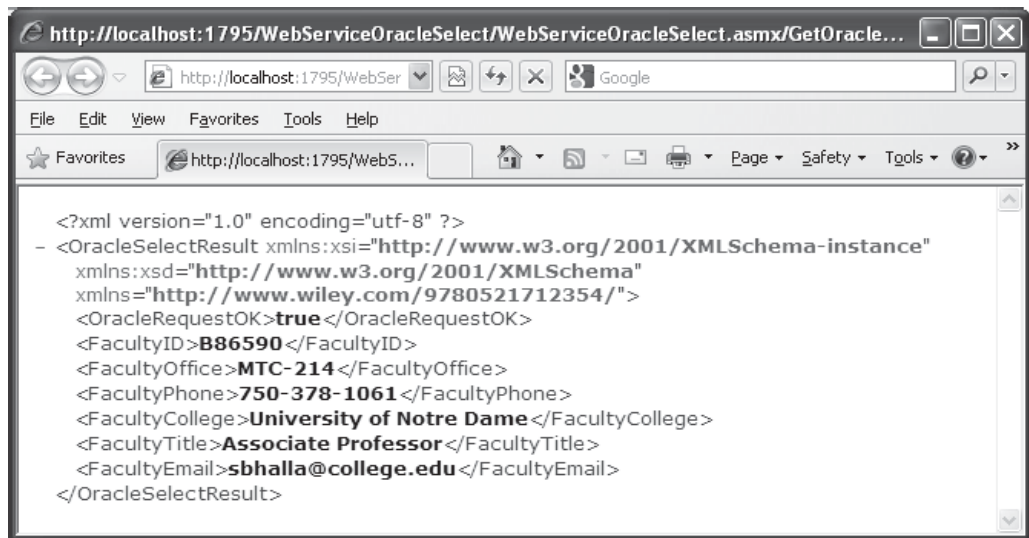


Figure 9.140. The running result of the Web method GetOracleSelectSP().

click on the Web method GetOracleSelectSP to open its parameter-input page. Enter the selected faculty name 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 the detailed information for the selected faculty. The running result of this Web method is shown in Figure 9.140.

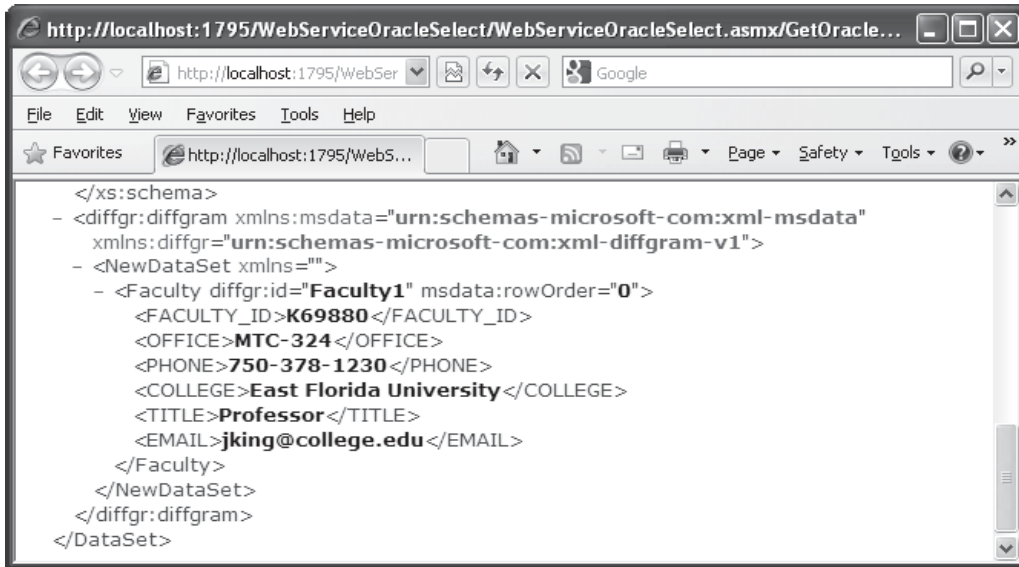


Figure 9.141. The running result of the Web method `GetOracleSelectDataSet()`.

The detailed information for the selected faculty Satish Bhalla is displayed in this built-in Web interface with XML tags.

Finally, let's test the Web method `GetOracleSelectDataSet()`. Close the current running result page and click on the Back arrow 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 Jenney King and click the Invoke button to run this method. The running result of this Web method is shown in Figure 9.141.

As shown in Figure 9.141, the detailed information that is contained in a DataSet for the selected faculty Jenney King 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 in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

9.9 BUILD WEB SERVICE CLIENT PROJECTS TO CONSUME THE WEB SERVICE

To consume 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 consume a Web service to access the SQL Server database and building a client project to consume a Web service to access the Oracle database. For example, you can use any client project, 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 newly 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 wizard, 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 wizard 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 in the folder **DBProjects\Chapter 9** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

Next, let's develop a Web service project to perform the data insertion for 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 Oracle database is very similar to the procedure of building an ASP.NET Web service to insert data into the SQL Server 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 distinguished 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 Oracle databases, we try to develop our Web service projects to insert data into the Oracle database by modifying some existing Web service projects. In this section, we concentrate on the modifications to the Web service project **WebServiceSQLInsert** and make it as our new Web service project **WebServiceOracleInsert**.

9.10.1 Build a Web Service Project **WebServiceOracleInsert**

In this section, we try to modify an existing Web service project **WebServiceSQLInsert** to make it as our new Web service project **WebServiceOracleInsert**, and allow it to insert data into the Oracle database.

Open the Windows Explorer and create a new folder **Chapter 9** under the root directory if you have not done that. Then browse to our desired source Web service project **WebServiceSQLInsert** that can be found in the folder **DBProjects\Chapter 9** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy and paste it into our new folder **Chapter 9**. Rename it to **WebServiceOracleInsert**. In the opened Windows Explorer window, perform the following modifications to this project:

1. Change the main Web service page from **WebServiceSQLInsert.asmx** to **WebServiceOracleInsert.asmx**.
2. Open the **App_Code** folder, change the name of our base class file from **SQLInsertBase.vb** to **OracleInsertBase.vb**.
3. Open the **App_Code** folder and change the name of our code-behind page from **WebServiceSQLInsert.vb** to **WebServiceOracleInsert.vb**.

Now open Visual Studio.NET 2010 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.vb**" to **CodeBehind** = "**~/App_Code/WebServiceOracleInsert.vb**"
- Change **Class** = "**WebServiceSQLInsert**" to **Class** = "**WebServiceOracleInsert**"

Second, open the base class file **OracleInsertBase.vb** and perform the following modifications:

- Change the class 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 the Connection String

Double-click 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 installing the Oracle Database 11g Expression Edition in the local computer. In our case, we used **reback** as the password for our sample database.

9.10.3 Add Oracle Database Reference and Modify the Namespace Directories

First, we need to add an Oracle Data Provider Reference to our Web service project. As you know, starting from .NET Framework 4.0, Microsoft no longer supports Oracle database-related operations. Therefore, we need to use an Oracle database driver provided by a third-party vendor. As we discussed in Section 5.20.3 in Chapter 5, we utilized a third-party product, dotConnect for Oracle 6.30 Express developed by Devart™ Inc.

Now let's add some Oracle Data Provider references to our project. Perform the following operations to complete this addition operation:

1. Right-click on our project Chapter9\WebServiceOracleInsert from the Solution Explorer window and select the **Add Reference** item from the pop-up menu to open the Add reference wizard.
2. With the .NET tab selected, scroll down the list until you find the items **Devart.Data** and **Devart.Data.Oracle**, click on both to select them, and click on the OK button to add these two references to our project.

Now double-click our code-behind page **WebServiceOracleInsert.vb** to open it. On the opened page, add two namespaces shown below to the top of this page:

```
Imports Devart.Data
Imports Devart.Data.Oracle
```

Also, change the name of our Web service class, which is located after the accessing mode **Public Class**, from **WebServiceSQLInsert** to **WebServiceOracleInsert**.

Next, we will perform the necessary modifications to four Web methods and related five differences listed above.

9.10.4 Modify the Web Method SetSQLInsertSP and Related Subroutines

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 subroutines **SQLConn()** and **ReportError()**.
5. The names of the dynamic parameters.

Let's perform those modifications starting from the first one.

Open this Web method and perform the modifications shown in Figure 9.142 to this method. All modified parts have been highlighted in bold.

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

- A. Rename this Web method to **SetOracleInsertSP** and the name of returned class to **OracleInsertBase**.

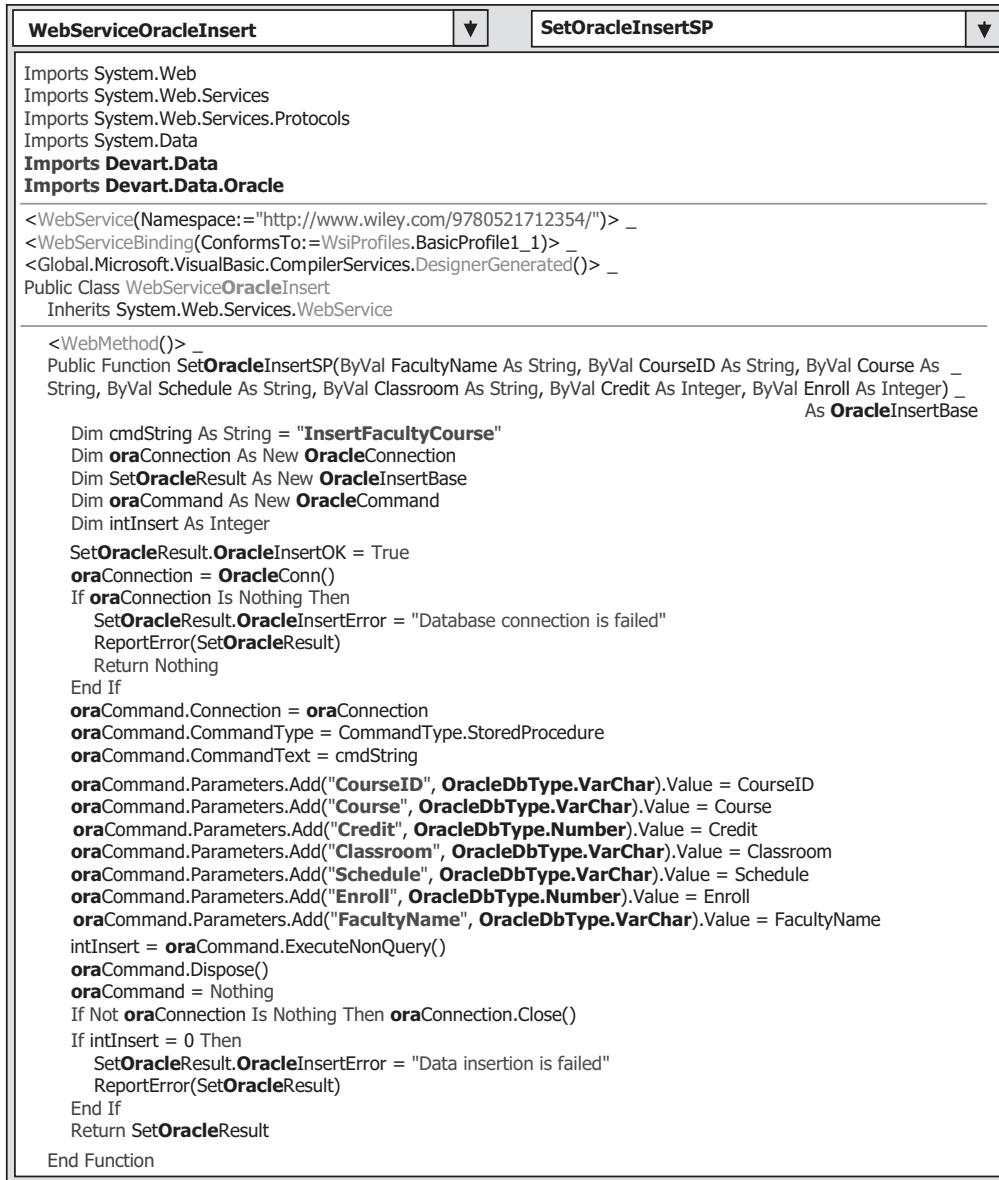


Figure 9.142. The modified Web method SetOracleInsertSP().

- B.** 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.
- 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 `SetSQLResult` to `SetOracleResult`.
- D.** Change the name of the returned instance from `SetSQLResult` to `SetOracleResult`, and member data from `SQLInsertOK` to `OracleInsertOK`.

- 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 **OracleDbType.VarChar**. Change the data type for the last two input parameters from **SqlDbType.Text** to **OracleDbType.Number**.
- H. 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 the modifications to two related subroutines **SQLConn()** and **ReportError()**. Perform the following modifications to the subroutine **SQLConn()**:

- A. Change the name of this subroutine from **SQLConn** to **OracleConn**, and return the 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 subroutine **ReportError()**:

- C. Change the data type of the 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 user-defined subroutine procedures **OracleConn()** and **ReportError()** should match one that is shown in Figure 9.143. All modified parts have been highlighted in bold.

For the stored procedure **InsertFacultyCourse**, we do not need to perform any modification to this procedure since we successfully developed this stored procedure in Section 6.8.2.1 in Chapter 6. Therefore, we can directly use this procedure without any problem. Refer to Section 6.8.2.1 to get more detailed information and discussion about the development process for this stored procedure if you like.

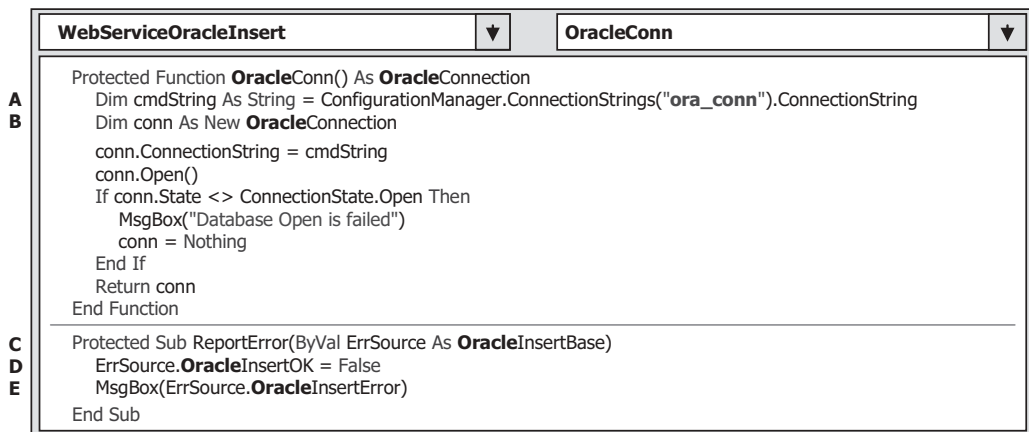


Figure 9.143. Modified subroutines **OracleConn()** and **ReportError()**.

9.10.5 Modify the Web Method GetSQLInsert and Related Subroutines

The function of this Web method is to retrieve all `course_id`, which includes the original and the newly 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 list box control in the client project.

Recall that in Section 5.19.6 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 only 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 perform a query to the Faculty table to obtain it. In this situation, a join query is a desired method to complete this function.

Open this Web method and perform the modifications that are shown in Figure 9.144 to this method. 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`.

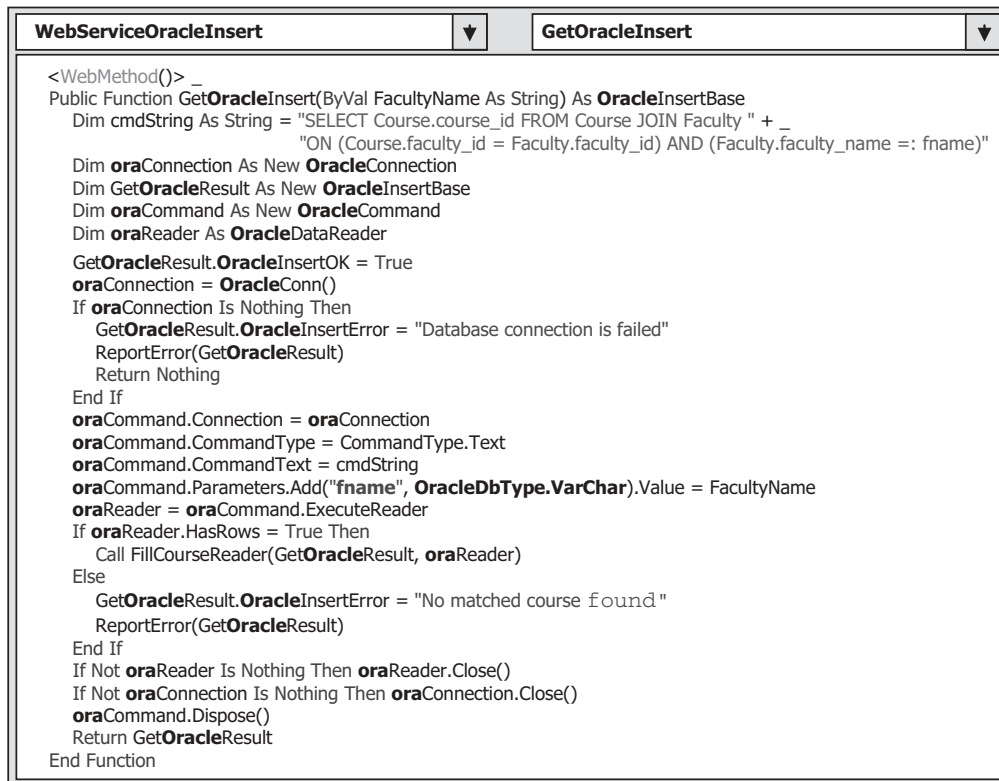


Figure 9.144. The modified Web method `GetOracleInsert()`.

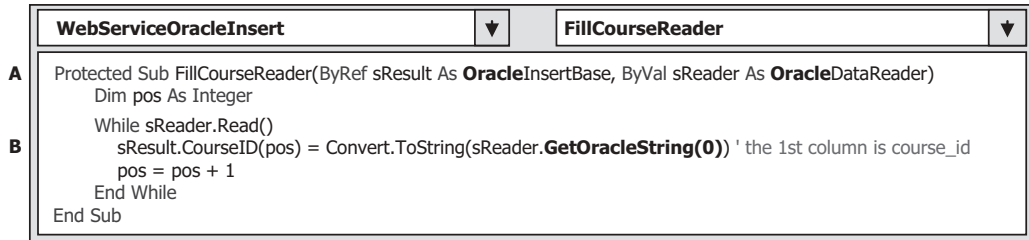


Figure 9.145. The modified subroutine FillCourseReader().

- B.** Modify the query string to match it to the ANSI 92 standard. Recall that we developed a join-table query string for SQL Server database using the ANSI 92 standard in Section 5.19.6 in Chapter 5. Use the Oracle comparison operator `=:` to replace the SQL Server comparator `LIKE@` for the second `WHERE` clause and the `=` to replace the `LIKE` in the first `WHERE` clause for this query string.
- 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 subroutine 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 `OracleDbType.VarChar`.
- G.** Change the names of two arguments passed to the subroutine `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 subroutine `FillCourseReader()` are relatively simple. Perform the following modifications to this subroutine:

- A.** Modify the data types of two passed arguments by changing the data type of the first argument from `SQLInsertBase` to `OracleInsertBase`, and by changing the data type of the second argument from `SqlDataReader` to `OracleDataReader`.
- B.** Change the method from `GetSQLString(0)` to `GetOracleString(0)`.

The modified subroutine `FillCourseReader()` is shown in Figure 9.145.

9.10.6 Modify the Web Method `SQLInsertDataSet`

The function of this Web method is to use an insert query to perform the course insertion and then retrieve the newly inserted course record and store it in a `DataSet` that will be returned to the calling procedure. Perform the modifications that are shown in Figure 9.146 to this Web method:

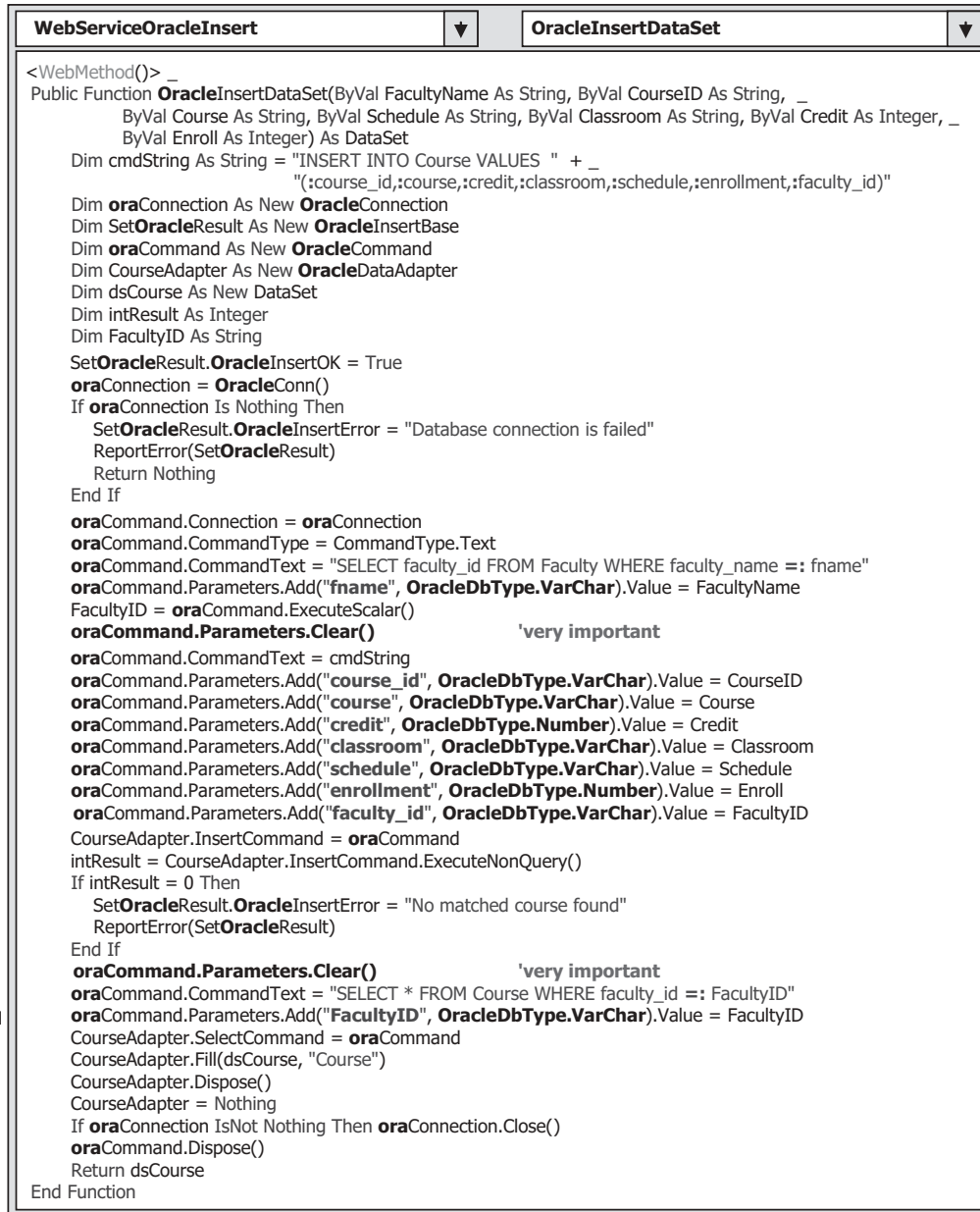


Figure 9.146. The modified Web method OracleInsertDataSet().

- 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 subroutine 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 comparison operator `LIKE @` before the nominal name `fname` with the Oracle comparison operator `=:`.
- G. Modify the nominal name of the input parameter by removing the `@` symbol before the nominal name `fname`. Also change the data type of this input parameter from `SqlDbType.Text` to `OracleDbType.VarChar`.
- H. Since the next query needs to use the different parameters, therefore, the Parameters collection must be cleaned up using the `Clear()` method to remove the previous parameter objects, and this is very important. An Oracle database exception may be encountered without this cleaning job 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.Text` to `OracleDbType.Number`.
- 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. Same function as we discussed in step H. However, this cleaning job is unnecessary in SQL Server database operations.
- L. Modify the dynamic name for the input parameter by replacing the SQL comparison operator `LIKE @` before the nominal name `FacultyID` with the Oracle comparison 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 `OracleDbType.VarChar`.
- N. Change the prefix from `sql` to `ora` for all data objects used in this method.

9.10.7 Modify the Web Method `GetSQLInsertCourse` and Related Subroutines

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 are needed to be performed for this Web method:

1. Modify the codes of this Web method.
2. Modify the related subroutine `FillCourseDetail()`.
3. Modify the content of the query string and create a new Package that contains a stored procedure.



Figure 9.147. The modified Web method GetOracleInsertCourse().

Open this Web method and perform the modifications shown Figure 9.147 to this Web method. Let's have 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 WebSelectCourseSP is an Oracle Package, and the SelectCourse is an Oracle 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.

- D. Add two Oracle Parameter objects `paramCourseID` and `paramCourseInfo`. Because of some differences existed between the SQL Server and Oracle databases, we need to use the different way to assign parameters to the Command object later.
- E. Change the name of the subroutine 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. 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 may 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 arguments passed to the subroutine `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`.

The modifications to the related subroutine `FillCourseDetail()` are simple. The only modifications are to change the data types of two passed arguments `sResult` and `sReader`. Change the data type of the first argument from `SQLInsertBase` to `OracleInsertBase`, and 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. We will use the Object Browser page in Oracle Database 11g XE to build this Package.

9.10.8 Build the Oracle Package `WebSelectCourseSP`

Open the Oracle Database 11g XE home page by going to `start|All Programs|Oracle Database 11g Express Edition|Get Started` items. Perform the following operations to create this package:

1. Click on the APEX button to open the Login to APEX page.
2. Enter `SYSTEM` and reback into the Username and Password box to complete the login process for the APEX.
3. Since we have already created our sample database `CSE_DEPT` in Chapter 2, click on the `Already have an account? Login Here` button.
4. Enter reback into the Password box and click on the Login button.
5. Click on the SQL Workshop icon to open this workshop window.
6. Click on the Object Browser icon and click on the drop-down arrow on the Create button, and select the Package item to open the Create Package wizard.

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 9.148.

Enter the package name `WebSelectCourseSP` into the Package Name box and click on the Next button to go to the specification page.

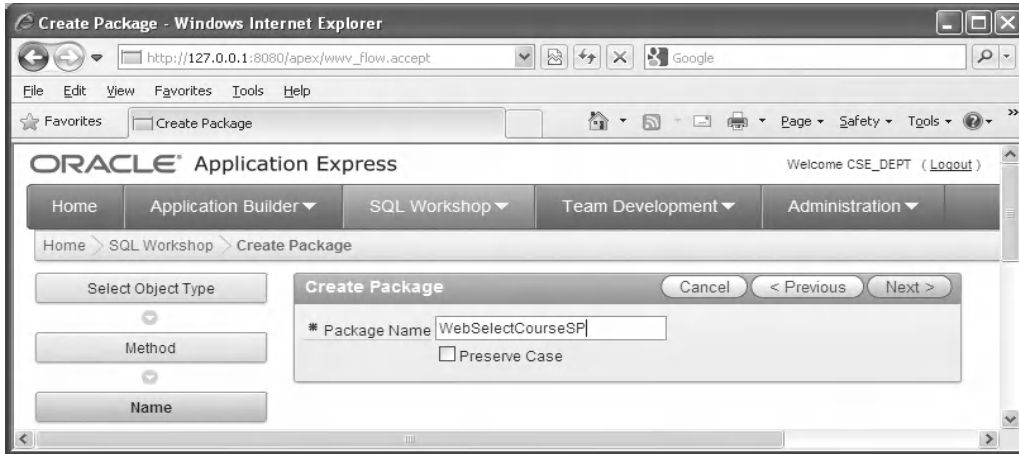


Figure 9.148. The Name page of the Package window.

```
create or replace package WEBSELECTCOURSESP as
TYPE CURSOR_TYPE IS REF CURSOR;
procedure SelectCourse (
    CourseID in VARCHAR2,
    CourseInfo out CURSOR_TYPE);
end;
```

Figure 9.149. The codes for 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 for our application, remove the default function prototype, and change the default procedure name from the **test** to our procedure name **SelectCourse**. Enter the codes shown in Figure 9.149 into the Specification space as the definition for our package.

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 **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 be ended by a semi-colon, and this rule is also applied to the **end** statement. Click on the **Finish** button to create this specification body.

Click on the **Body** tab to open the Body page, and click on the **Edit** button to begin to create our body part. Enter the PL-SQL codes that are shown in Figure 9.150 into this body space.

```

create or replace package body WebSelectCourseSP AS
  procedure SelectCourse(CourseID in VARCHAR2,
                        CourseInfo out CURSOR_TYPE) AS
  begin
    OPEN CourseInfo FOR
    SELECT course, schedule, classroom, credit, enrollment, faculty_id FROM Course
    WHERE course_id = CourseID;
  end;
end;

```

Figure 9.150. The codes for the Body part of the package.

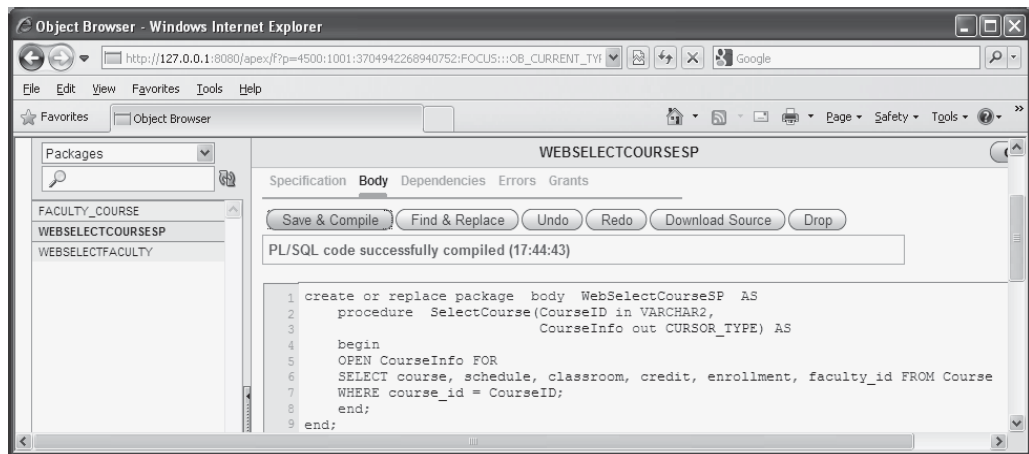


Figure 9.151. The compiled codes for the body part of the package.

The procedure prototype is re-declared in line 2. Starting from the **begin**, our real SQL statements are included in lines 6 and 7. The **OPEN CourseInfo FOR** command is used 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 5.19.8.4 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.

Ok, now let's compile our package by clicking on the **Save & Compile** button. A successful compiling information **PL/SQL code successfully compiled (17:44:43)** is displayed if this package is bug-free, which is shown in Figure 9.151.

At this point, we finished the development of our Oracle package and all modifications to our new Web service project. Close the Oracle Database 11g 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 on 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 that are shown in Figure 9.152 into the associated **Value** boxes in the opened parameter-input page, and click on the **Invoke** button to run this method.

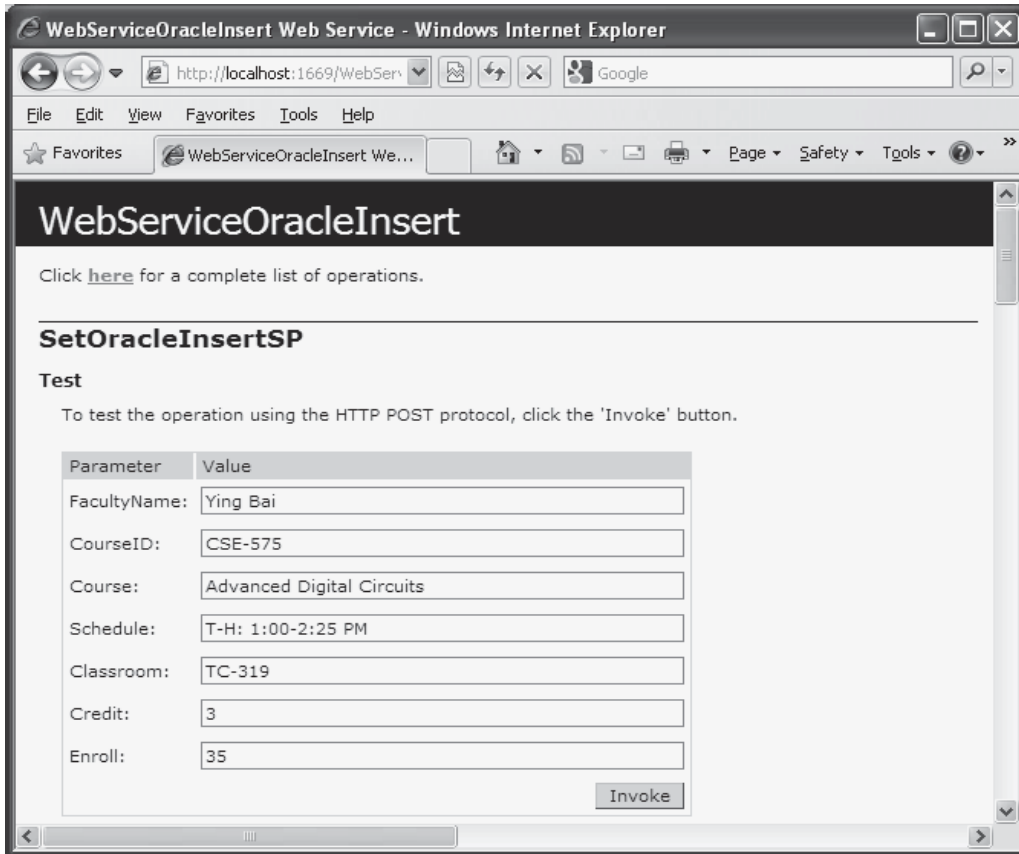


Figure 9.152. The parameter-input built-in Web interface.

The running result of this Web method is shown in Figure 9.153.

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 the Course table in our sample database CSE_DEPT from the Oracle Database 11g XE using the Browser Object to check this newly inserted course.

An example of an opened Course table is shown in Figure 9.154, and you can find that the newly inserted course CSE-575 is located at the first row.

Now let's test the second Web method `GetOracleInsert()` by clicking on it. 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 9.155.

It can be found that the newly inserted course CSE-575 is indeed added into the Course table, and this data insertion is successful.

Close the current running result page and click the Back arrow to return to our Web start page. Click on the Web method `OracleInsertDataSet()` to test this method. Enter the input parameters that are shown in Figure 9.156 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.

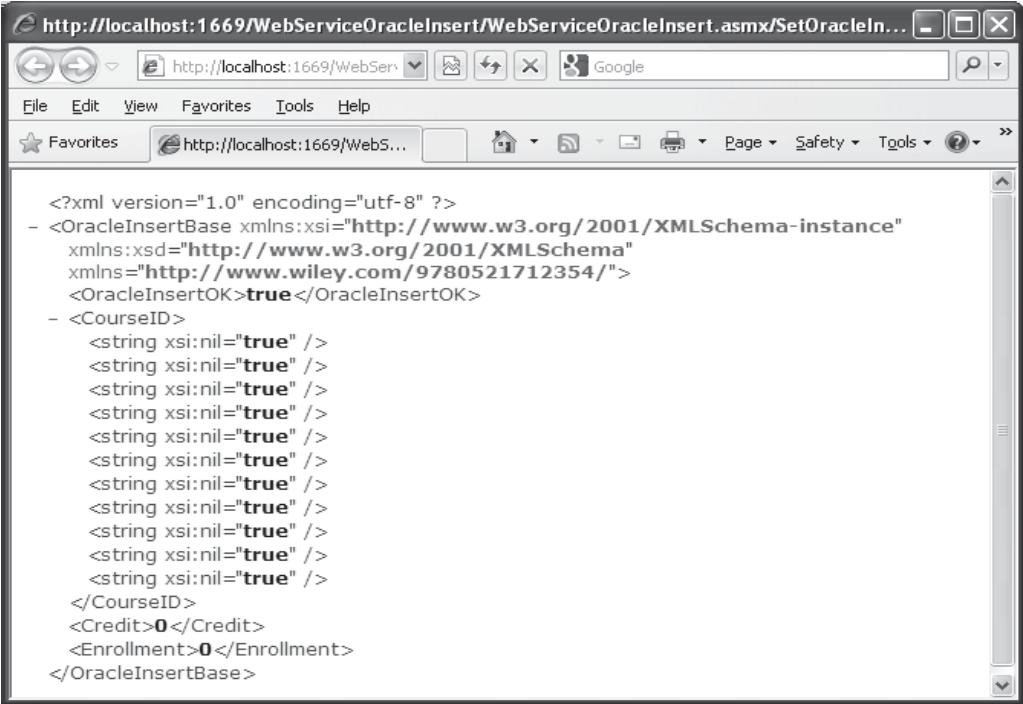


Figure 9.153. The running result of the Web method SetOracleInsertSP().

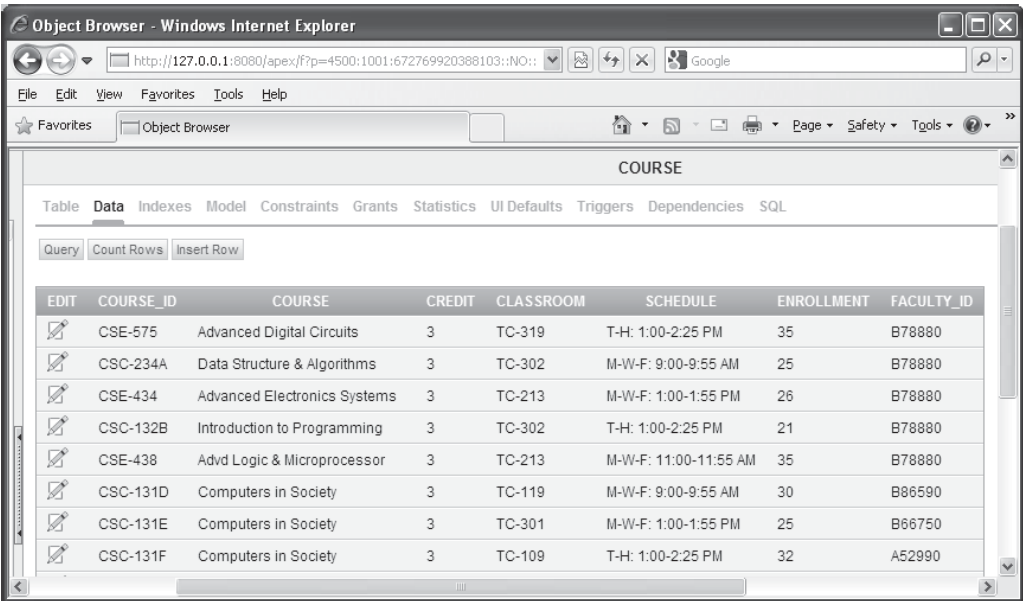


Figure 9.154. The opened Course table.

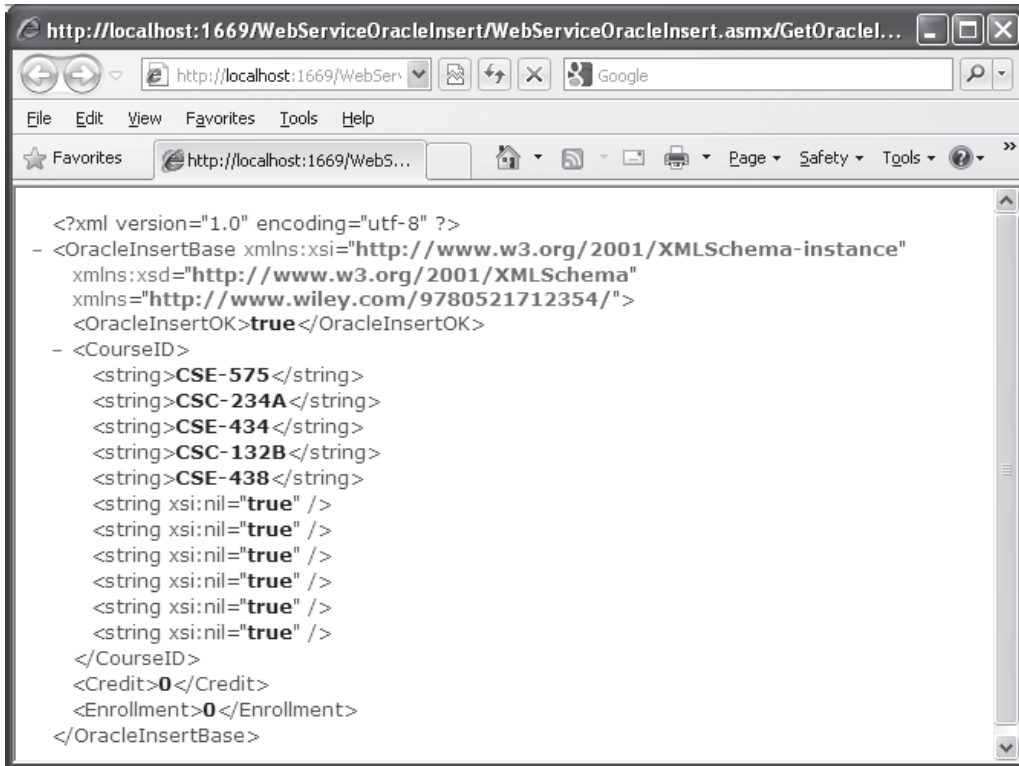


Figure 9.155. The running result of the Web method `GetOracleInsert()`.

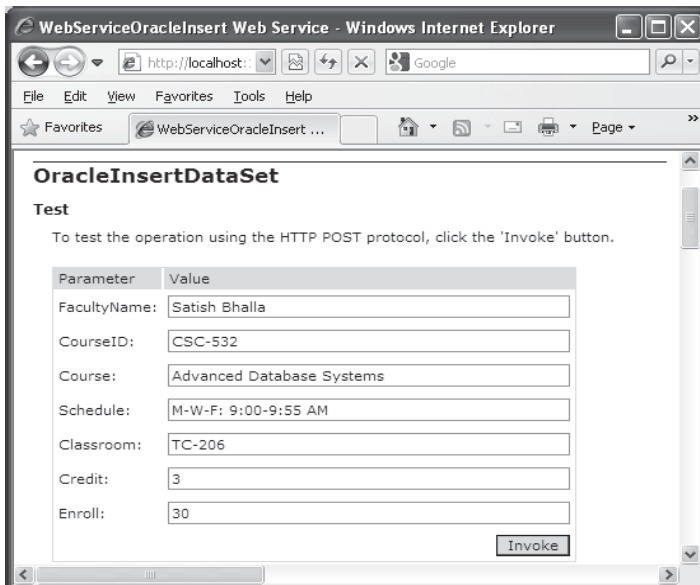


Figure 9.156. The parameter-input page.

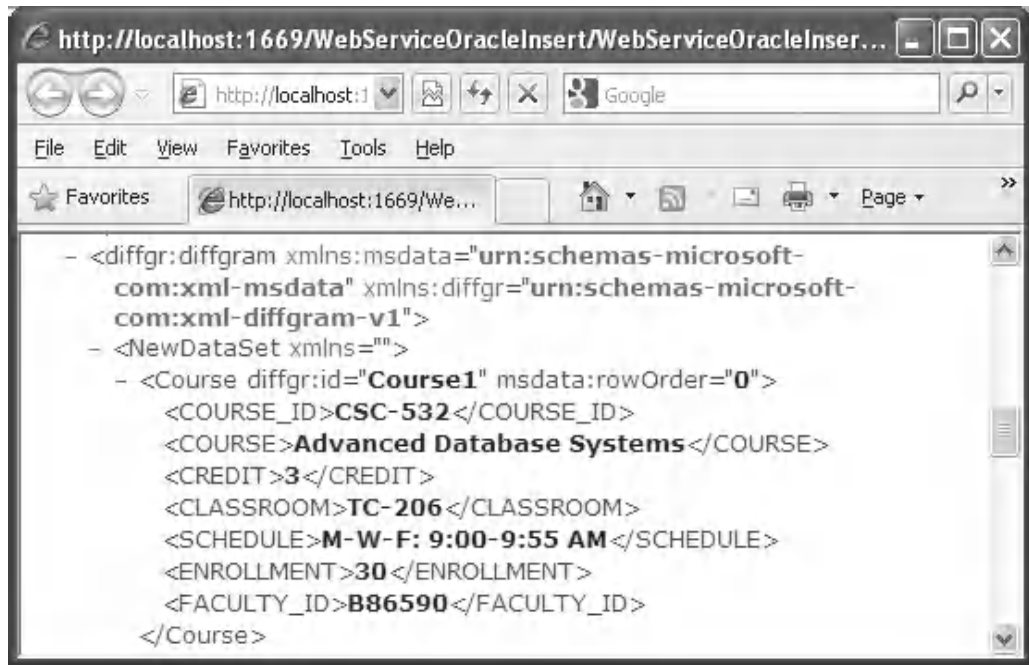


Figure 9.157. The running result of the Web method OracleInsertDataSet().

The running result of this Web method is shown in Figure 9.157.

It can be found in Figure 9.157 that the newly 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 arrow 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 9.158.

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 have finished the testing for all Web methods we developed in this Web service project. A complete Web service project WebServiceOracleInsert can be found in the folder DBProjects\Chapter 9 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

9.11 BUILD WEB SERVICE CLIENT PROJECTS TO CONSUME THE WEB SERVICE

To consume 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 build-

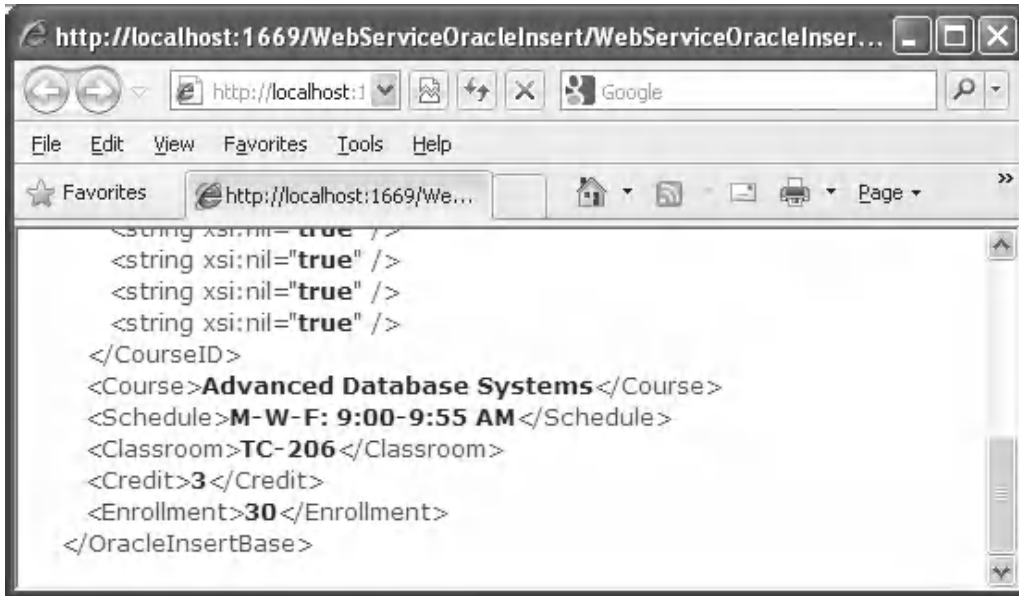


Figure 9.158. The running result of the method GetOracleInsertCourse().

ing a client project to consume a Web service to access the Oracle database and building a client project to consume a Web service to access the SQL Server database. For example, you can use any client project, such as either WinClientSQLInsert or WebClientSQLInsert, which we developed in the previous sections, to consume this Web service project WebServiceOracleInsert with small modifications. The main modification is to replace the Web Reference with a new Web Reference class, which is our newly developed Web service 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 wizard, 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 wizard 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 Windows-based, and WebClientOracleInsert, which is Web-based, can be found in the folder DBProjects\Chapter 9 that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

9.12 BUILD ASP.NET WEB SERVICE TO UPDATE AND DELETE DATA FOR THE ORACLE DATABASE

Basically, the procedure to build an ASP.NET Web service to update and delete data against the Oracle database is very similar to the procedure of building an ASP.NET Web service to update and delete data against the SQL Server 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 distinguished points exist between the procedures to build a Web service to update and delete data against two kinds of databases.

Based on the discussion and analysis we made in Section 9.8, as well as the similarity between the Oracle and SQL Server databases, we try to develop our Web service projects to update and delete data against the Oracle database by modifying some existing Web service project. In this section, we concentrate on the modifications to the Web service project `WebServiceSQLUpdateDelete` and make it as our new Web service project `WebServiceOracleUpdateDelete`.

9.12.1 Build a Web Service Project `WebServiceOracleUpdateDelete`

In this section, we try to modify an existing Web service project `WebServiceSQLUpdateDelete` to make it as our new Web service project `WebServiceOracleUpdateDelete`, and allow it to update and delete data against the Oracle database.

Open the Windows Explorer and create a new folder **Chapter 9** under the root directory if you have not done that. Then browse to our desired source Web service project `WebServiceSQLUpdateDelete` that can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). Copy and paste it into our new folder `C:\Chapter9`. Rename this project to `WebServiceOracleUpdateDelete`. In the Windows Explorer window, perform the following modifications to this project:

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.vb` to `OracleBase.vb`.
3. Open the `App_Code` folder and change the name of our code-behind page from `WebServiceSQLUpdateDelete.vb` to `WebServiceOracleUpdateDelete.vb`.

Now open Visual Studio.NET 2010 and our newly created Web service project `WebServiceOracleUpdateDelete` 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 `WebServiceOracleUpdateDelete.aspx`. Open this page by double-clicking on it from the Solution Explorer window and perform the following modifications:

- Change `CodeBehind` = "`~/App_Code/WebServiceSQLUpdateDelete.vb`" to `CodeBehind` = "`~/App_Code/WebServiceOracleUpdateDelete.vb`"
- Change `Class` = "`WebServiceSQLUpdateDelete`" to `Class` = "`WebServiceOracleUpdateDelete`"

Second, open the base class file `OracleBase.vb` and perform the following modifications:

- Change the class 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.

Next, let's begin to do the modifications listed above to our new Web service project. We start from the first step, modify the connection string in the Web configuration file `Web.config`.

9.12.2 Modify the Connection String

Double-click 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 installing the Oracle Database 11g XE in the local computer. In our case, we used `reback` as the password.

9.12.3 Add Oracle Database Reference and Modify the Namespace Directories

First, we need to add an Oracle Data Provider Reference to our Web service project. As you know, starting from .NET Framework 4.0, Microsoft no longer supports Oracle database-related operations. Therefore we need to use an Oracle database driver provided by a third-party vendor. As we discussed in Section 5.20.3 in Chapter 5, we utilized a third-party product, **dotConnect for Oracle 6.30 Express** developed by Devart™ Inc.

Now let's add some Oracle Data Provider references to our project. Perform the following operations to complete this addition operation:

1. Right-click on our project `Chapter 9\WebServiceOracleUpdateDelete` from the Solution Explorer window, and select the **Add Reference** item from the pop-up menu to open the Add Reference wizard.

2. With the .NET tab selected, scroll down the list until you find the items `Devart.Data` and `Devart.Data.Oracle`, click on both to select them, and click on the OK button to add these two references to our project.

Now double-click our code-behind page `WebServiceOracleUpdateDelete.vb` to open it. On the opened page, add two namespaces shown below to the top of this page:

```
Imports Devart.Data
Imports Devart.Data.Oracle
```

Also, change the name of our Web service class, which is located after the accessing mode `Public Class`, to `WebServiceOracleUpdateDelete`.

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 the Web Method `SQLUpdateSP` and Related Subroutines

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 subroutines `SQLConn()` and `ReportError()`.
6. The names of the dynamic parameters.

Let's perform those modifications step by step according to this sequence. Open this Web method and perform the modifications shown in Figure 9.159. All modified parts have been highlighted in bold.

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

- A. Rename this Web method to `OracleUpdateSP` and 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.
- 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

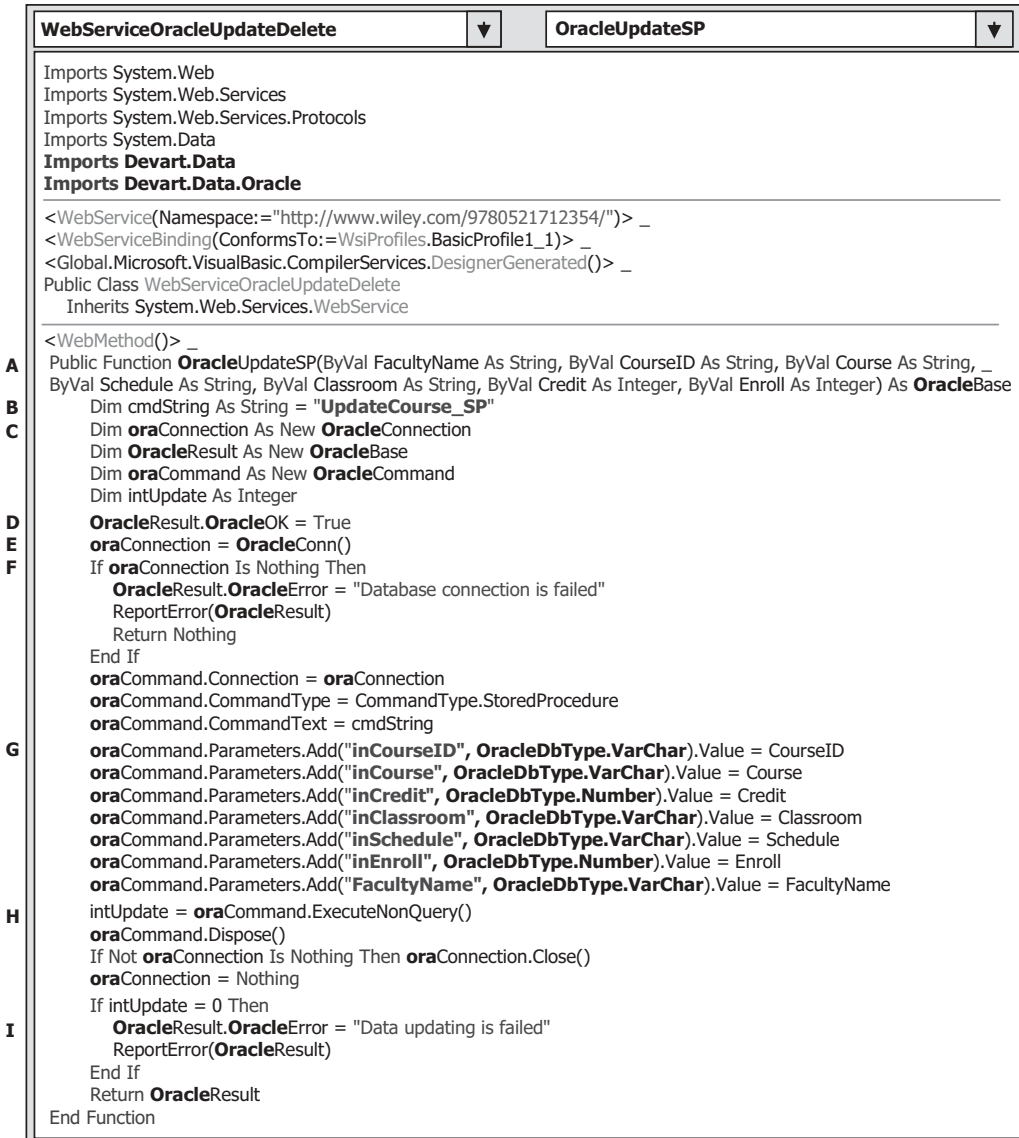


Figure 9.159. The modified Web method OracleUpdateSP().

five text input parameters from `SqlDbType.Text` to `OracleDbType.VarChar`. Change the data type for the two number input parameters from `SqlDbType.Text` to `OracleDbType.Number`.

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`.

Now let's perform the modifications to two related subroutines `SQLConn()` and `ReportError()`. Perform the following modifications to the subroutine `SQLConn()`:

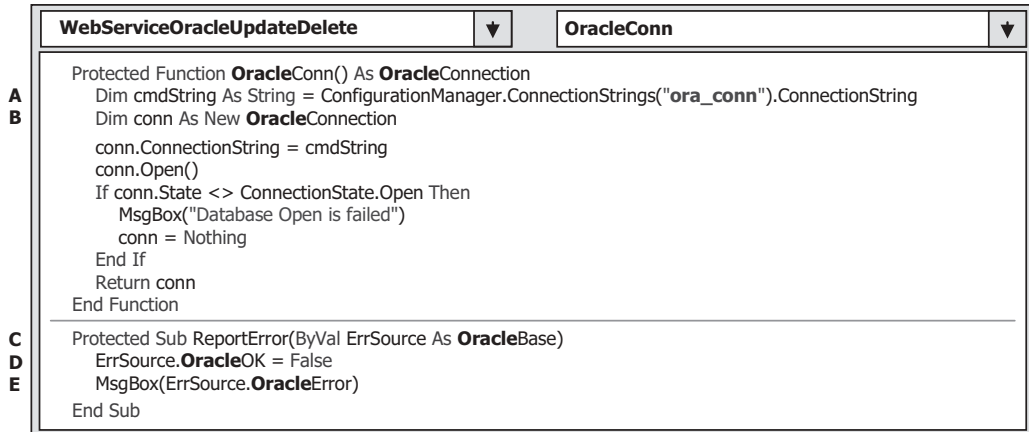


Figure 9.160. Modified subroutines OracleConn() and ReportError().

- A.** Change the name of this subroutine 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 subroutine ReportError():
- 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 subroutines OracleConn() and ReportError() should match the one that is shown in Figure 9.160. All modified parts have been highlighted in bold.

Next, let's develop the stored procedure UpdateCourseSP to perform the course updating function.

9.12.4.1 Develop the Stored Procedure UpdateCourse_SP

A very detailed discussion about creating and manipulating packages and stored procedures in the Oracle database is provided in Section 5.20.7 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 to update and delete data against the database, and no returned data is needed for these two data actions. Therefore, we only need to create stored procedures in the Oracle database, not packages, to perform the data updating and deleting functionalities.

As we discussed in Section 5.20.7.1 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 11g XE to create our stored procedures.

Open the Oracle Database 11g XE home page by going to the start!All Programs!Oracle Database 11g Express Edition!Get Started items. Perform the following operations to create this stored procedure:

1. Click on the **APEX** button to open the Login to APEX page.
2. Enter **SYSTEM** and **reback** into the Username and Password box to complete the login process for the APEX.
3. Since we have already created our sample database **CSE_DEPT** in Chapter 2, click on the **Already have an account? Login Here** button.
4. Enter **reback** into the Password box and click on the **Login** button.
5. Click on the **SQL Workshop** icon to open this workshop window.
6. Click on the **Object Browser** icon and click on the drop-down arrow on the **Create** button, and select the **Procedure** item to open the Create Procedure wizard.
7. Enter **UpdateCourse_SP** into the Procedure Name box and keep the **Include Arguments** checkbox checked, and click on the **Next** button to go to the next page.
8. The next wizard is used to allow us to enter all input parameters. For this stored procedure, we need to perform two queries, so 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_id** in the Course table based on the **faculty_id** 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 **FacultyName** is used by the first query, and the following six input parameters are used by the second query.
9. 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 2.11.5 in Chapter 2 to get a detailed list of data types used for those data columns in the Course data table.
10. For the **Input/Output** selection of the parameters, select **IN** for all seven parameters, since no output is needed for this data-updating query.

Your finished argument list should match one that is shown in Figure 9.161.

Click on the **Next** button to go to the procedure-defining page. Enter the codes that are shown in Figure 9.162 into this new procedure as the body of the procedure using the language of so-called PL-SQL. Then click on the **Next** and then the **Finish** buttons to confirm to create this procedure. Your finished stored procedure should match the one that is shown in Figure 9.163.

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 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**. The second query is to update a course record based on six input parameters in the Course table. A semi-colon 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 9.163, and this coding line has

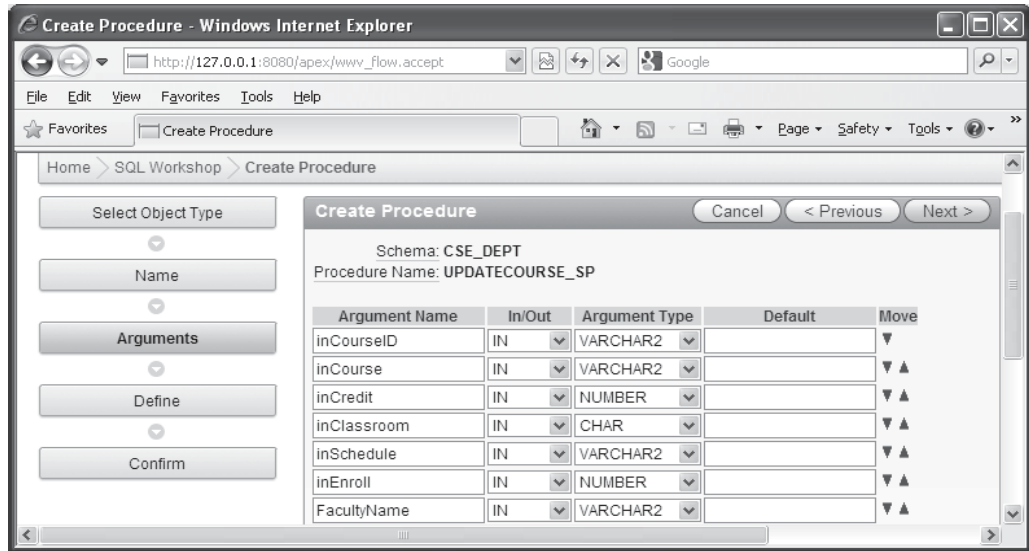


Figure 9.161. The finished argument list wizard.

```
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 9.162. The stored procedure body.

been highlighted with the black color. Type this coding line to add this local variable. This local variable is used to hold the returned `faculty_id` from executing the first query.

Another important issue to arrange the input parameters 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 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 **Save & 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 11g Express Edition by clicking the **Close** button that is located at the upper-right corner of this wizard.

Next, let's return to the Visual Studio.NET environment and open our Web service project `WebServiceOracleUpdateDelete` to build the codes to call this stored procedure to perform the data updating actions against the `Course` table in our sample database.

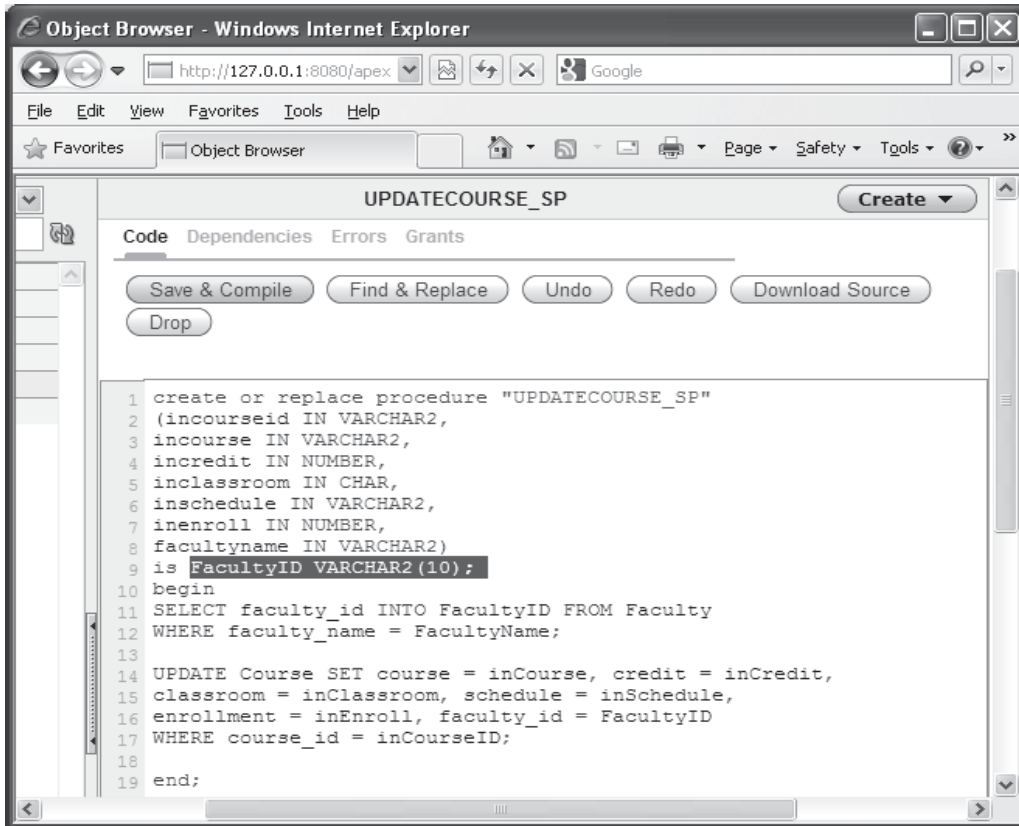


Figure 9.163. The completed stored procedure.

9.12.5 Modify the Web Method GetSQLCourse and Related Subroutines

The function of this Web method is to retrieve all `course_id`, which includes the original and the newly 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 list box control in the client project.

Recall that in Section 5.19.6 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 only 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 perform a query from the Faculty table to obtain it. In this situation, a join query is a desired method to complete this function.

Open this Web method and perform the modifications that are shown in Figure 9.164 to this method. All modified parts have been highlighted in bold.

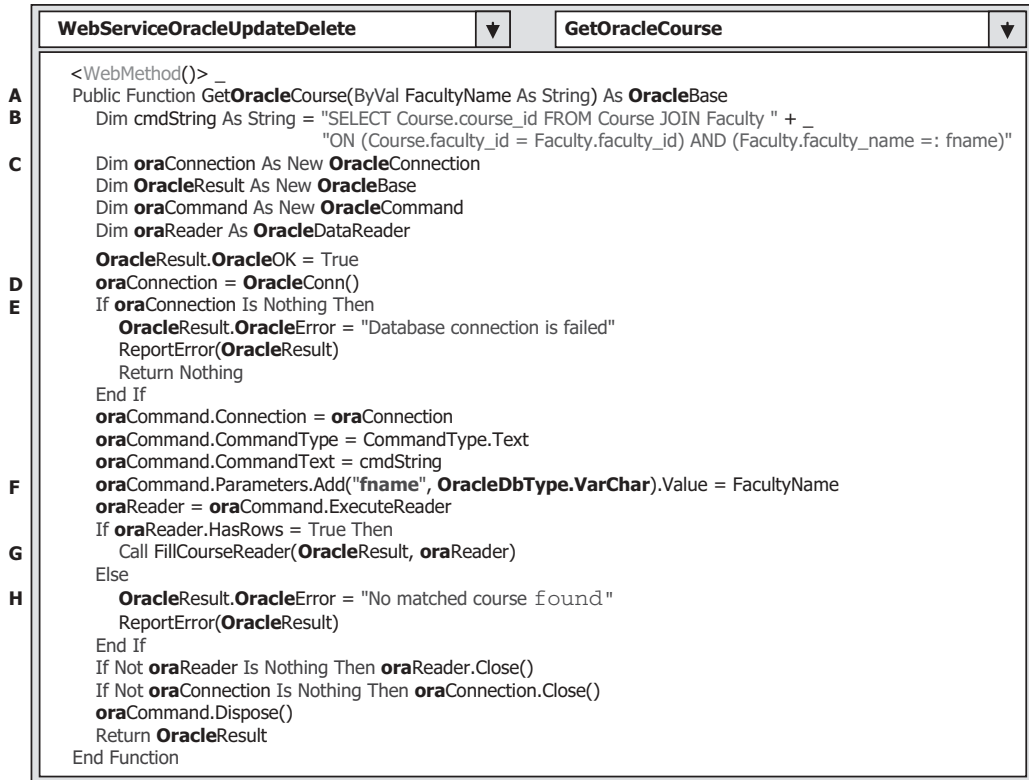


Figure 9.164. The modified Web method GetOracleCourse().

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 Oracle database query style, which is to replace the SQL comparison operator LIKE@ with the Oracle comparator =: for the second clause, and the = to replace LIKE in the first clause in the ON statement.
- 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 subroutine from SQLConn to OracleConn. Change the second member data from SQLError to OracleError.
- E.** Change the prefix from sql to ora for all data objects. 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 OracleDbType.VarChar.
- G.** Change the names of two arguments passed to the subroutine FillCourseReader() from SQLResult to OracleResult, and from sqlReader to oraReader.

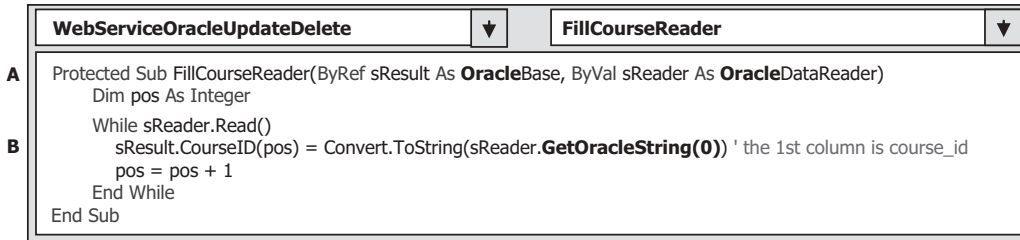


Figure 9.165. The modified subroutine FillCourseReader().

- 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 from **sql** to **ora** for all data objects.

The modifications to the related subroutine FillCourseReader() are relatively simple. Perform the following modifications to this subroutine:

- 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 subroutine FillCourseReader() should match the one that is shown in Figure 9.165.

9.12.6 Modify the Web Method GetSQLCourseDetail and Related Subroutines

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 subroutine FillCourseDetail().
3. Modify the content of the query string and make it equal to the name of an Oracle Package, **WebSelectCourseSP**, we developed in Section 9.10.8.

Open this Web method and perform the modifications that are shown in Figure 9.166 to this Web method. All modified parts have been highlighted in bold.

Let's have a closer look at these modifications to see how they work.

- A.** Change the name of this Web method to **GetOracleCourseDetail**. Also, change the name of the returned base class from **SQLBase** to **OracleBase**.
- B.** Modify the content of the query string and make it equal to the name of the Package we developed in Section 9.10.8. Change this query string from **dbo.WebSelectCourseSP** to **WebSelectCourseSP.SelectCourse**. The **WebSelectCourseSP** is an Oracle Package, and the **SelectCourse** is an Oracle stored procedure.

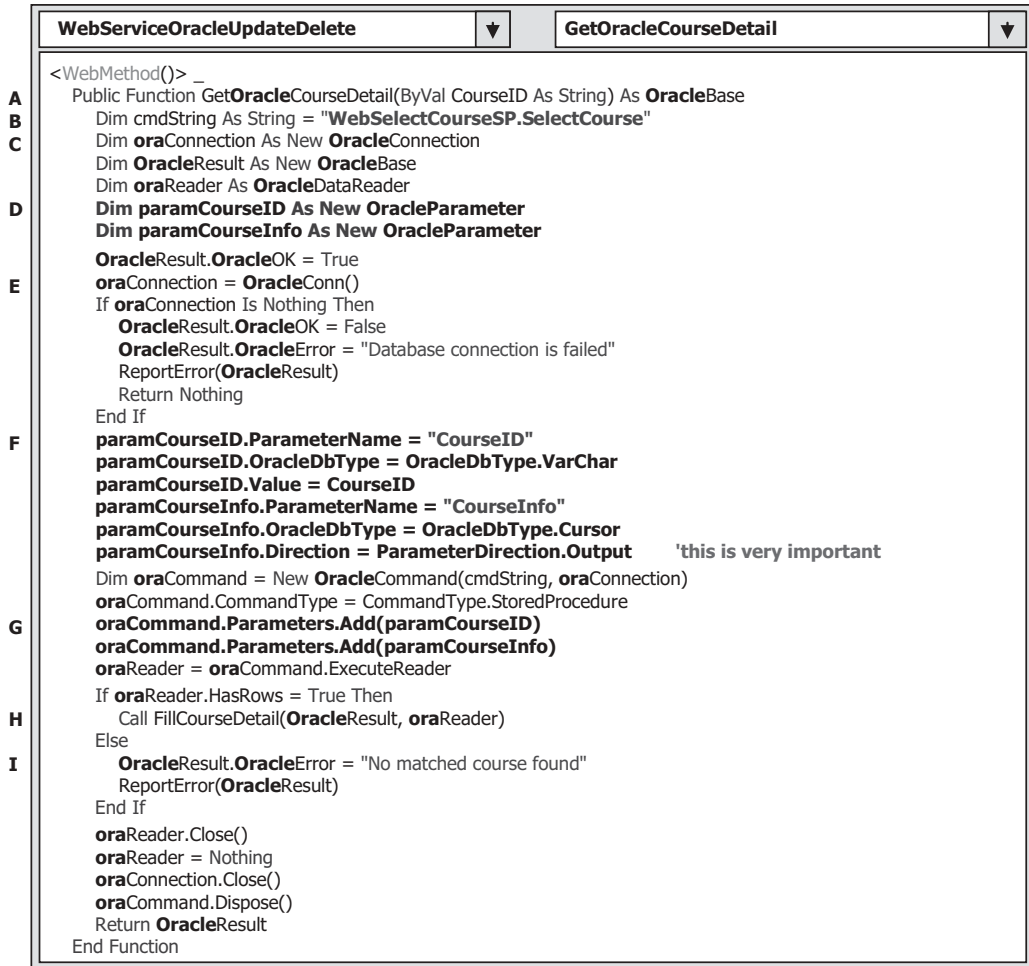


Figure 9.166. The modified Web method GetOracleCourseDetail().

- 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. Add two Oracle Parameter objects **paramCourseID** and **paramCourseInfo**. Because of some differences exist between the SQL Server and Oracle databases, we need to use a different way to assign parameters to the Parameters collection of the Command object later.
- E. Change the name of the subroutine from **SQLConn** to **OracleConn**. Change the second member data from **SQLError** 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 may be encountered when the project runs.

- G.** Use two statements to add two `OracleParameter` objects to the `Command` object.
- H.** Change the names of two arguments passed to the subroutine `FillCourseDetail()` 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 subroutine `FillCourseDetail()` are simple. The only modification is 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 the 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 Figure 2.5 in Section 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. To save time and space, as well for efficiency, we place these two queries into a stored procedure named `WebDeleteCourseSP()` that will be developed in the following section. A single input parameter `course_id` is passed into this stored procedure as the primary key.

Open this Web method and perform the modifications that are shown in Figure 9.167 to this Web method. All modified parts have been highlighted in bold.

Let's take a closer look at this piece of modified codes 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 we will develop soon. 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`.

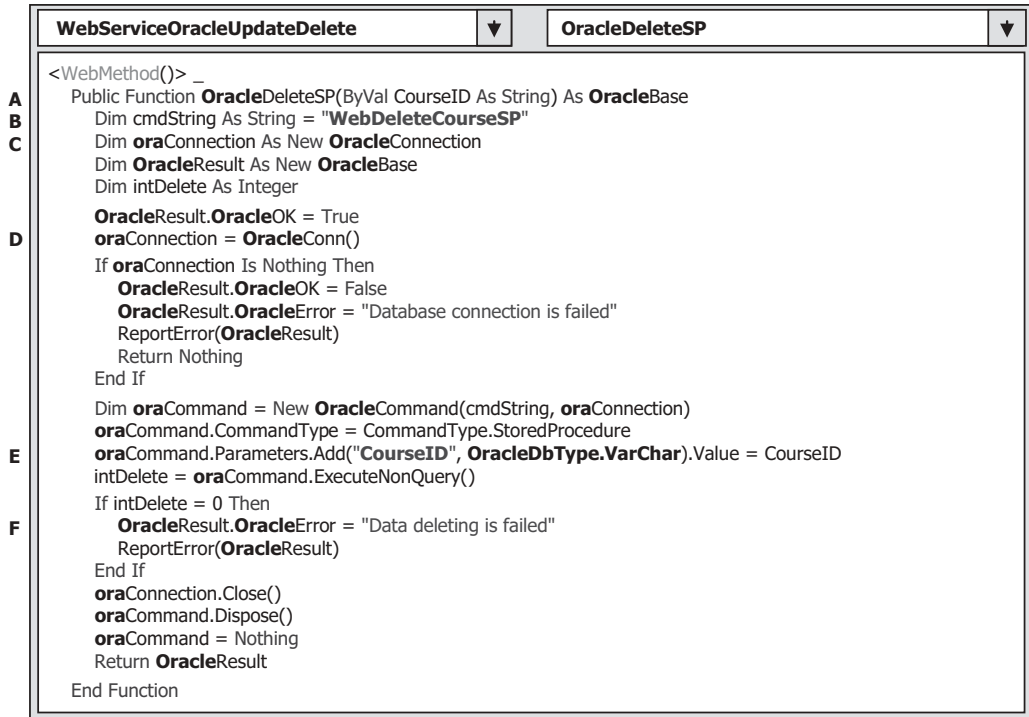


Figure 9.167. The modified Web method OracleDeleteSP().

- D.** Change the name of the subroutine 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 **OracleDbType.VarChar**.
- F.** Change the name of the returned instance from **SQLResult** to **OracleResult**, change the second member data from **SQLError** to **OracleError**, and change the prefix from **sql** to **ora** for all data objects.

Next, let's build the Oracle stored procedure **WebDeleteCourseSP()**.

9.12.7.1 Develop the Stored Procedure **WebDeleteCourseSP**

The topic we are discussing in this section is to update and delete data against the database, so no returned data is needed for these two data actions. Therefore, we only need to create an Oracle stored procedure, not an Oracle package, to perform the data deleting function.

As we discussed in Section 5.20.7.1 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 11g XE to create our stored procedures.

Open the Oracle Database 11g XE home page by going to **start|All Programs|Oracle Database 11g Express Edition|Get Started** items. Perform the following operations to create this stored procedure:

1. Click on the **APEX** button to open the Login to APEX page.
2. Enter **SYSTEM** and reback into the Username and Password box to complete the login process for the APEX.
3. Since we have already created our sample database **CSE_DEPT** in Chapter 2, click on the **Already have an account? Login Here** button.
4. Enter reback into the Password box and click on the Login button.
5. Click on the **SQL Workshop** icon to open this workshop window.
6. Click on the **Object Browser** icon and click on the drop-down arrow on the **Create** button, and select the **Procedure** item to open the Create Procedure wizard.
7. Enter **WebDeleteCourseSP** into the Procedure Name box and keep the **Include Arguments** checkbox checked, and click on the **Next** button to go to the next page.
8. The next wizard is used to allow us to enter input parameters. For this stored procedure, we need to perform two queries: first, we need to delete any related course records from the child table—**StudentCourse** table—and, secondly, we can delete the target course record from the parent table—**Course** table—based on the input **course_id**. Only one input parameter **CourseID** is needed for this stored procedure.
9. 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** in the **Course** table. Refer to Section 2.11.5 in Chapter 2 to get a detailed list of data types used for those data columns in the **Course** data table.
10. For the **Input/Output** selection of the parameters, select **IN** for this input parameter, since no output is needed for this data deleting query.

Your finished argument list should match one that is shown in Figure 9.168.

Click on the **Next** button to go to the procedure-defining page. Enter the codes that are shown in Figure 9.169 into this new procedure as the body of the procedure using the

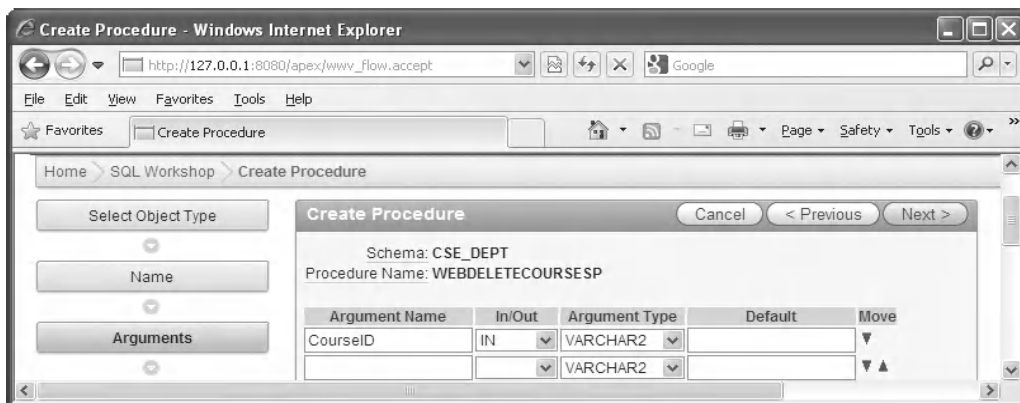


Figure 9.168. The argument list wizard.


```
DELETE FROM StudentCourse WHERE course_id = CourseID;  
DELETE FROM Course WHERE course_id = CourseID;
```

Figure 9.169. The coding body of the stored procedure.

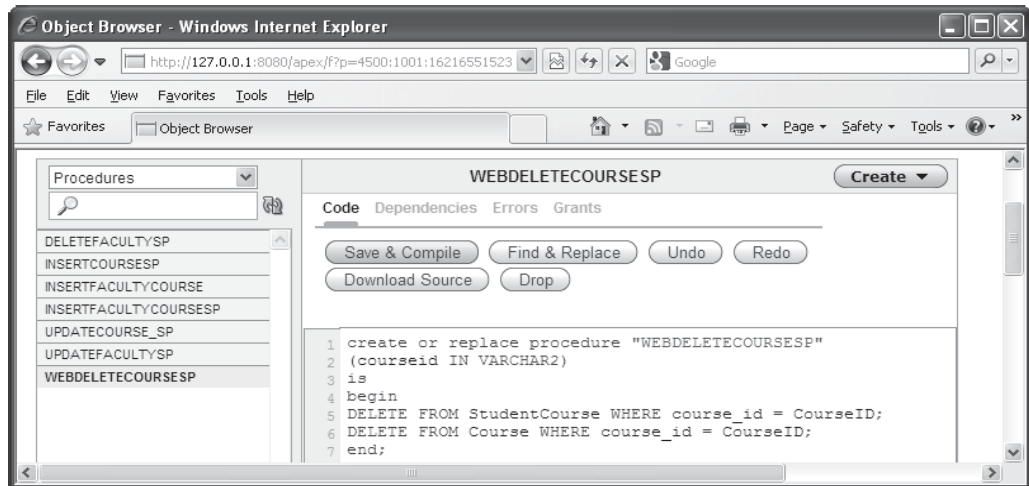


Figure 9.170. The completed coding body of the stored procedure.

language of so-called Procedural Language Extension for SQL or PL-SQL. Then click on the Next and then the Finish buttons to confirm creating this procedure. Your finished stored procedure body should match one that is shown in Figure 9.170.

Two queries are included in this procedure. The first query is used to delete the related course records from the child table (StudentCourse table), and the second query is to delete the target course record from the parent table (Course table). A semicolon must be attached after each PL-SQL statement.

To make sure that this procedure works properly, we need to compile it first. Click on the **Save & 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 11g Express Edition by clicking 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 functionalities.

Click on the **Start Debugging** button to run the project. First, let's test the Web method **OracleUpdateSP()** to update a course record **CSE-575** 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-575** 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 9.171.

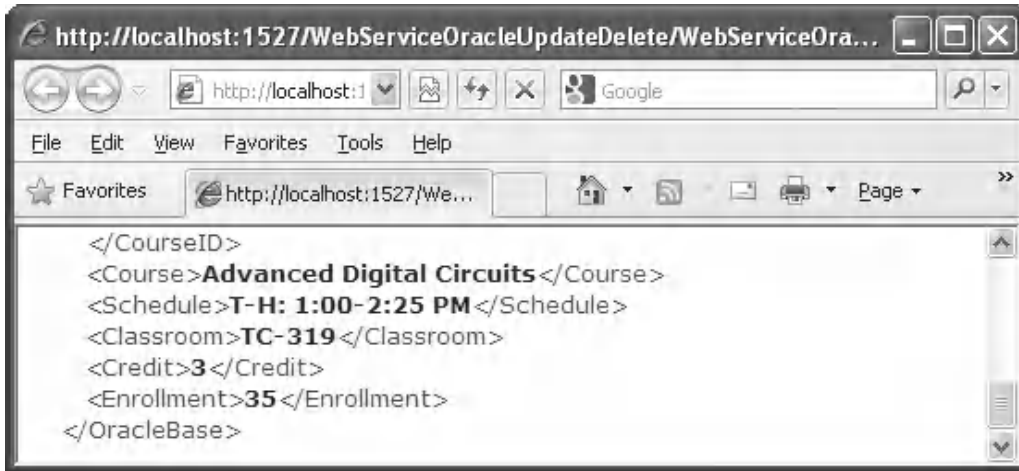


Figure 9.171. The running result of the Web method GetOracleCourseDetail().

Try to remember 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 arrow on the top of this page to return to the initial Web page. Click on the Web method OracleUpdateSP to open its parameter-input page. Enter the updating course information that is shown in Figure 9.172 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 running result wizard.

To confirm this course updating, click on the Back arrow to return to the initial page and click on the Web method GetOracleCourseDetail to try to get back the detailed information for this updated course to validate this data updating. Enter CSE-575 into the CourseID box, and click on the Invoke button to run this method. The running result of this method is shown in Figure 9.173.

Compare this running result with the one that is shown in Figure 9.171; it can be found that this course has been updated.

Close the current running result wizard and click the Back arrow to return to the initial page. Next, let's test the Web method OracleDeleteSP().

Click on this method and enter a course_id that you want to delete, such as CSE-575, 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 running result wizard.

To confirm this data deleting, let's run the Web method GetOracleCourse() to retrieve all courses taught by the selected faculty. Recall that the course CSE-575 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 into the FacultyName box, and click on the Invoke button to run this Web method. The running result is shown in Figure 9.174.

From this running result, it can be found that the course CSE-575 has been deleted from the Course table successfully.

The screenshot shows a web browser window titled "WebServiceOracleUpdateDelete Web Service - Windows Internet Explo...". The address bar shows "http://localhost:1". The browser has a menu bar with "File", "Edit", "View", "Favorites", "Tools", and "Help". Below the menu bar is a "Favorites" section with a link to "WebServiceOracleUpdate...". The main content area is titled "OracleUpdateSP" and contains a "Test" section. The "Test" section has a paragraph: "To test the operation using the HTTP POST protocol, click the 'Invoke' button." Below this is a form with a table of parameters and their values. The parameters are: FacultyName (Ying Bai), CourseID (CSE-575), Course (Advanced Discrete Signal Processing), Schedule (M-W-F: 9:00-9:55 AM), Classroom (TC-315), Credit (3), and Enroll (29). An "Invoke" button is located at the bottom right of the form.

Parameter	Value
FacultyName:	Ying Bai
CourseID:	CSE-575
Course:	Advanced Discrete Signal Processing
Schedule:	M-W-F: 9:00-9:55 AM
Classroom:	TC-315
Credit:	3
Enroll:	29

Invoke

Figure 9.172. The parameter-input page.

The screenshot shows a web browser window titled "http://localhost:1527/WebServiceOracleUpdateDelete/WebServiceOra...". The address bar shows "http://localhost:1527/We...". The browser has a menu bar with "File", "Edit", "View", "Favorites", "Tools", and "Help". Below the menu bar is a "Favorites" section with a link to "http://localhost:1527/We...". The main content area displays the XML response from the Web method GetOracleCourseDetail(). The XML is as follows:

```

</CourseID>
<Course>Advanced Discrete Signal Processing</Course>
<Schedule>M-W-F: 9:00-9:55 AM</Schedule>
<Classroom>TC-315</Classroom>
<Credit>3</Credit>
<Enrollment>29</Enrollment>
</OracleBase>

```

Figure 9.173. The running result of the Web method GetOracleCourseDetail().

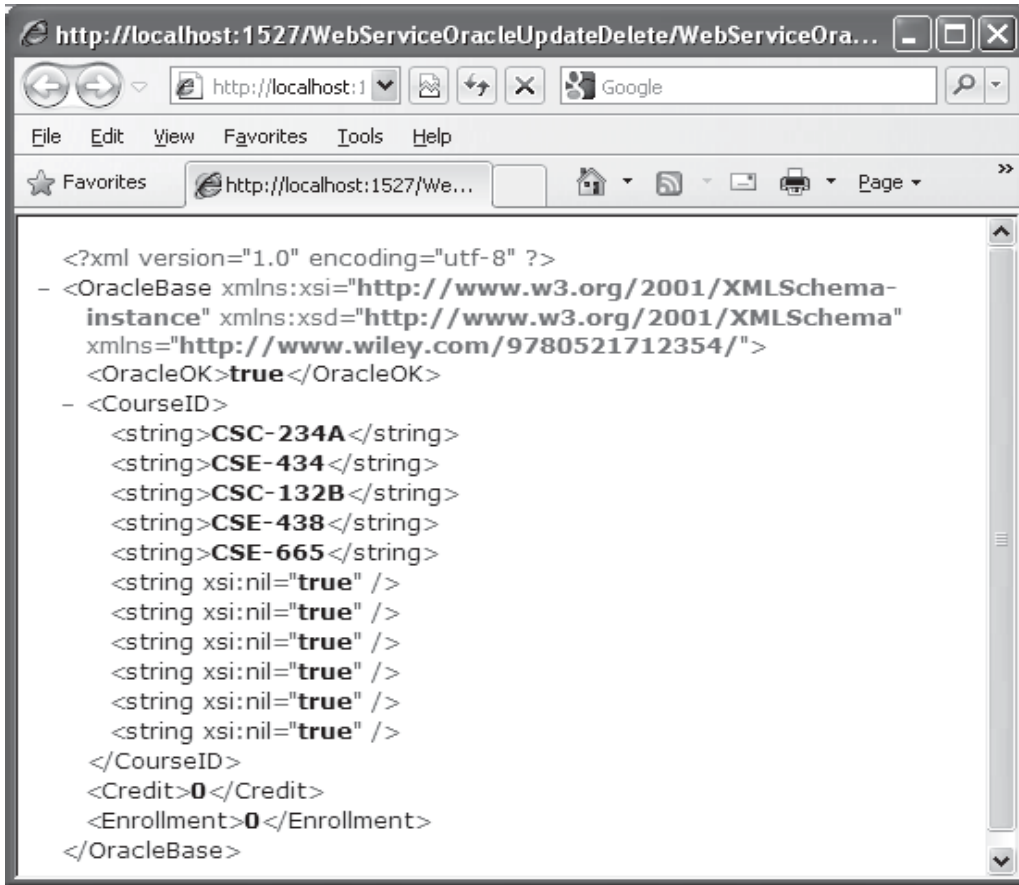


Figure 9.174. The running result of the Web method GetOracleCourse().

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 in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

9.13 BUILD WEB SERVICE CLIENT PROJECTS TO CONSUME THE WEB SERVICE

To consume 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 consume a Web service to access the Oracle database and building a client project to consume a Web service to access the SQL Server database. For example, you can use any client projects, such as either `WinClientSQLUpdateDelete` or `WebClientSQLUpdateDelete`, which we developed in the previous sections, to consume this Web service project `WebServiceOracleUpdateDelete` with small modifications. The

main modification is to replace the **Web Reference** with a new Web Reference class that is our newly 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 wizard. 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 wizard in the client project.
3. Change the Web reference name for all data components used in the client project.
4. Remove the namespace for the SQL Server Data Provider from the top of each page.
5. Change the name of the base class located in the Web reference.
6. 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 in the folder **DBProjects\Chapter 9** that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).

9.14 CHAPTER SUMMARY

A detailed discussion and analysis about the structure and components of the Web services is 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 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 consume 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 operating systems, such as Windows 2000, Windows XP, Windows Vista, Windows 7, and Windows 8.

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 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 to list themselves on the Internet, and 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 build a Web service are located in 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, but you must deploy it to a production server if you want other users to access your Web service from 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. The 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 a 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 consume 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).
 - a. XML
 - b. HTTP
 - c. HTML
 - d. Java scripts

III. Exercises

1. Write a paragraph to answer and explain the following questions:
 - a. What is ASP.NET Web service?
 - b. What are main components of the ASP.NET Web service?
 - c. How an ASP.NET Web service is executed?
2. Suppose we have a Web service project and the main service page contains the following statement:

```
<%@ WebService Language = "vb" CodeBehind = "~/App_Code/testWeb.vb" Class = "testWeb" %>
```

Answer the following questions:

- a. What is the name of this Web service?
 - b. What are the name and the location of the code-behind page of this Web service?
 - c. Is the content of this page related to the WSDL file of this Web service?
3. 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.
4. Add the Web method `GetSQLStudent()` in question 3 into our Web service project `WebServiceSQLSelect` and develop the codes to that method to perform the data query for the Student via our sample SQL Server database `CSE_DEPT`. The project file can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1).
5. 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 project `WinClientSQLSelect` that can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). In fact, you can add some controls to the main form window and the codes to the associated event procedures in that project to complete this job.
6. Develop a Web-based Web service client project `WebClientSQLStudent` to consume the Web service developed in question 4, exactly to consume the new Web method `GetSQLStudent()`.
Hints: Refer to project `WinClientSQLSelect` that can be found in the folder `DBProjects\Chapter 9` that is located at the Wiley ftp site (refer to Figure 1.2 in Chapter 1). In fact, you can add some controls to the main form window and the codes to the associated event procedures in that project to complete this job.

Index

.NET Framework Collection Classes, 422
.NET Framework Data Provider, 253
<Extension()> Keyword, 231

A

AcceptCondition, 126
AcceptButtonproperty, 260
AcceptChanges(), 195
AccessSelectRTOObject, 314–315, 329–330, 345, 349, 353, 379, 388, 395, 417
Active ServiceMethod file, 660
ActiveServer Page.NET, 560
ActiveX Data Object (ADO), 96
AddColumn, 68, 76
AddConnectiondialog, 190–191, 201, 307, 373
AddExistingItem, 750, 794
AddNewStoredProcedure, 375, 380
AddProceduredialog, 380
Add Query, 196
Add Reference, 462
Add Table, 296
Add Web Reference, 693–694, 705, 734, 735, 751, 785, 795, 821, 858
Add() Method, 106–108, 145, 218, 319, 322, 423–424, 449, 491, 502, 600, 645
Address Bar, 694, 704, 734, 750, 785
AddWithValue() Methods, 107
ADO.NET 3.5 Framework, 92
ADO.NET Common Language Runtime, 92
AllCells, 269
ALTERTABLE Command, 87
Alternate Keys, 17
ANSI 92 Standard, 364, 826
ANSI89 Standard, 722
Applicationstate, 560, 570, 577, 582, 584, 592, 599, 605–606, 617, 624, 630–633, 635–636, 638, 644–645, 647, 655, 706, 755–757
Application User Interfaces, 421, 499
Application.Exit(), 286, 323, 359, 392
AsEnumerable, 153–154, 179, 184, 186, 187–188, 194–196, 238, 326, 348–349, 457
AsEnumerable(), 154, 186–188, 195–196, 238, 326, 348–349, 457

asmx, 562, 661, 664–665, 669–672, 675, 686, 693, 712, 716, 722, 724, 731, 733, 764, 778–781, 783, 805–806, 818, 822, 836, 839–840, 859
ASP.NET, 560
ASP.NET Runtime, 672, 685
ASP.NET Web Application, 560–563, 566, 567, 570, 584, 653, 655–656, 703, 749
ASP.NET Web Forms, 565
ASP.NET Web Service, 666, 705, 845
ASP+, 660
Assembly Information, 421, 785
Attribute (XName), 221
Attributes, 16, 21, 23, 28, 30, 133, 216, 220–222, 230, 564, 660, 664–665, 669, 722
AutoPostBack Property, 560, 583, 590, 591, 752, 755
AutoSizeColumns Mode, 269
AutoSizeRows Mode, 269

B

BinFigure, 222, 224, 502
Bind ListBox, 298
Binding Navigator, 119, 246–247, 249, 259, 269, 311, 415
BandingSource, 119, 149, 242, 246–247, 249–250, 259, 269, 273–274, 276–278, 287, 299, 311, 414–416, 420, 498
Bitmap Indexes, 35
BorderStyle Property, 263
B-Tree, 29, 33, 35
BuildCommand(), 342, 344–345, 378, 383, 460, 469, 470, 521, 525
Built-in Web Interface, 680, 685, 687, 692, 695, 705, 718, 722–723, 728, 733, 735, 778, 785, 817, 819, 834, 837, 853

C

C# Web Services, 659
Caching, 94, 166, 213
Candidate Key, 17
Cardinality, 16, 35
Cascade Deleting, 611
Cascade Option, 611–612
Cast(), 172

Cast(Of TResult)(IQueryable), 153
 CDATA, 217
 CenterScreen, 137, 259–260, 263, 736
 ChangeData Source, 267, 307
 Child Class, 199, 661, 670, 676–677, 685, 687, 699, 707, 716
 Class Libraries, 560
 Clear Method, 120, 122
 ClearBeforeFill Property, 290
 ClientServer Databases, 28
 Close() Method, 101, 321, 577, 697, 709, 739
 Clustered Index, 29
 Clustered Table, 33
 Code Illustrations, 562
 Code Window, 96, 144, 170, 173, 176, 182, 184, 187, 191, 198, 203, 225, 270, 285, 289, 293, 300, 316, 317, 323, 341, 354, 355, 383, 384, 388, 389, 437, 454, 459, 462, 463, 474, 486, 522, 526, 551, 569, 592, 621, 629, 697, 700, 718, 783
 Code-behind Page, 562, 568–569, 660, 669, 674, 679–680, 682, 688, 690, 718, 721, 731, 765, 771, 807, 823, 859, 861
 Column Scollection, 124–125, 128
 Column Collections, 123
 ComboMethod, 753
 ComboBox Control, 243, 245
 Command Class, 103, 105, 107, 109, 110, 148, 346, 378, 407, 415, 447, 476, 493, 514–515, 518, 556, 571, 594, 617, 717, 732
 Command Console, 561
 Command Execution, 94, 495
 Command Object, 4, 93, 95, 102–104, 106–110, 113, 116, 123, 145, 147–149, 185, 186, 193, 314, 319–320, 325, 327, 333, 342, 356, 364, 371–372, 377–378, 383, 384, 390, 395, 407–410, 414, 418–419, 443, 475–476, 492, 495, 513, 515, 518, 531, 540–541, 557–558, 571, 584, 596, 599, 616–617, 638, 645, 647, 681–682, 689, 691, 718, 722–728, 731–732, 767, 770, 772, 816, 831, 848–849
 CommandText Property, 104, 122, 145, 371, 476, 495, 558, 616, 726, 728
 CommandType Property, 104, 371, 383, 413, 495, 531, 540, 541, 557, 617, 689, 732
 CommandType.StoredProcedure, 371, 377, 378, 406, 476, 487, 495, 531, 541, 557, 558, 617, 647, 689, 718, 731, 765, 769, 771, 830, 848, 850
 Common Language Runtime, 654
 Compile Directive, 669–670
 Component Object Model, 91
 Composite Key, 16
 Configuration Files, 562
 Configuration Manager class, 679
 Connection Class, 99, 101–102
 Connection Instance, 99–100, 316, 351, 354, 359, 592
 Connection Object, 93
 Connection Property, 104, 122, 617

Connection String, 99–101, 133, 140, 144–147, 185, 187, 189, 193, 203, 254, 268, 308–309, 317, 325, 350–355, 364, 367, 386–389, 441, 454, 463, 521–522, 550, 570, 594, 621, 629–630, 678–681, 804–806, 808, 821–822, 824, 839–840
 ConnectionState.Open, 101, 317, 324, 332, 360, 362, 368, 392, 396, 455, 456, 459, 464, 466, 469, 570, 577, 582, 592, 630, 632, 634, 656, 679, 681, 824, 842
 Connectivity, 94
 Console.ReadLine(), 203
 Console.WriteLine(), 170, 172–173, 177, 192, 194, 196, 205
 Constraint Name, 650
 Constraint Property, 650
 Constraint Type, 99, 650
 ControlToValidate, 573, 574
 Count, 73, 83, 164, 171–172, 180, 184, 279, 290, 301, 319, 325, 333, 335, 336, 344, 378, 390, 407, 410, 414, 433, 545, 597, 650, 835
 Count Property, 290, 301, 319, 333, 344, 378, 390, 407, 410
 Createor Replace, 404, 485, 545, 812, 814, 832
 Create Package, 400–401, 810–812
 Create Procedure, 482–483, 843, 851
 Creation Wizard, 87, 92
 CURSOR_TYPE, 399, 402, 403, 811–812, 814–815, 832
 Custom Stored Procedures, 369

D

Data Access Objects, 91
 Data Actions, 4, 6, 114, 198, 201, 216, 263, 440–443, 494–495, 499, 512–513, 550, 576, 582, 621, 626, 738, 785, 842
 DataBindings Property, 275, 277, 286, 299
 Data Connections, 99, 312
 Data Consistency, 14
 Data Deleting Query, 500
 Data Files, 14, 32, 31, 36
 Data Independence, 14
 Data Insertion Methods, 425
 Data Model, 10, 12, 16, 129, 131, 168
 Data Provider Classes, 468
 DataProvider-Dependent Objects, 359
 Data Query, 95, 103, 107, 110, 112, 119, 147–149, 162, 234, 242, 255, 257, 260, 263, 272, 288, 294, 302, 304, 310, 315, 318–319, 321–325, 332, 333, 337, 342, 344, 354, 356, 357–358, 360, 368, 379, 384, 389, 390, 392, 393, 396, 397, 411–418, 444, 450, 471, 494, 496, 499, 514, 557, 559, 583, 591, 592, 613, 626, 678, 682, 684, 685, 687–689, 691, 692, 695–698, 701, 703, 706, 709, 721, 722, 731, 767–770, 804, 809, 815, 821, 839, 845
 Data Redundancy, 14
 Data Selection, 203, 620
 Data Sharing, 14

- DataSource Configuration, 250, 252, 253, 266, 267, 268, 309
 - DataSource Parameter, 387
 - DataSource Wizard, 253
 - Data Tool, 71
 - Data Type, 105
 - Data Updating Query, 205
 - Data Validation, 421, 423, 425, 429, 444–446, 461, 474, 480, 491, 499, 516, 573, 574, 606, 641, 726, 736, 739, 746, 749, 763, 786
 - Data Adapter Class, 112–114, 186, 691
 - DataApapter.Update(), 513
 - Database Alias, 352, 386
 - Database Connectivity, 96, 562
 - Database Design, 10, 16, 19, 21
 - Database Development Process, 16, 93
 - Database Management Programs, 10
 - Database Management Systems, 12
 - Database Parameters, 100
 - Database Processing, 16
 - Database Programs, 1, 2, 12
 - Database Technology, 10, 386
 - Database Versions, 4
 - DataBindings Property, 276
 - DataColumn, 118, 123–125, 127, 128, 148, 195, 196, 327, 338, 346, 347, 415
 - DataColumn Objects, 118
 - DataColumn Collection, 123
 - Data Connector, 250
 - DataContext Class, 198–199, 203, 240, 550, 620, 621, 624–626
 - Data-driven Projects, 384
 - Data GridView, 119, 242, 246, 247, 249, 259, 269, 270, 272, 413
 - DataReader, 126
 - DataReader Method, 315, 321, 325, 335, 337, 364, 389, 407, 410, 417
 - DataReader Object, 94, 95, 115, 118, 147, 314, 320, 325, 357, 371, 383, 390, 571, 584, 586, 594, 597, 683, 684, 722
 - Data Relation, 117, 118, 148, 247
 - DataRow, 113, 118, 121, 123–128, 148, 179, 180, 192–197, 205, 207, 327, 338, 346, 347, 415, 420, 430–433, 496, 497, 505–507, 699, 700, 745, 758
 - DataRow Object, 124–125, 431–432, 505
 - DataRow Collection, 125, 148, 415, 495, 604
 - DataRow Extensions, 166, 192, 196
 - DataRow State, 126
 - DataSet Class, 94, 114, 118–121, 247, 707
 - DataSet Component, 117
 - DataSet Designer, 119, 189, 190, 256, 257, 270, 272, 281, 288, 296, 301, 420, 498
 - DataSet Events, 121
 - DataSet Object, 124
 - DataSet-DataAdapter, 313, 443, 513
 - DataSet Name, 120
 - DataSet-TableAdapter, 315
 - DataSource, 249
 - DataTable Class, 118, 123–127, 148, 184, 196, 324, 327, 699
 - DataTable Components, 93
 - DataTable Object, 94–95, 123–125, 147, 148, 193, 196, 314, 327, 336, 413, 414
 - DataTable Extensions, 166, 192, 196
 - DataView, 123–124
 - dBase, 10
 - DBDirect Method, 425–427, 496, 498, 499, 503, 506, 556
 - DBML File, 199
 - DbType Property, 104–105, 829
 - Delete Rule Item, 655
 - Delete Command, 4, 94, 103, 112, 146, 314, 419, 420, 443, 497, 513, 555
 - Design View, 36, 52, 78
 - DialogResult, 506–507, 517–518, 531, 541
 - Direction Property, 407
 - Disconnected Working Mode, 418
 - Discovery Map File, 703, 749
 - DisplayMember Property, 298
 - Dispose() Method, 102
 - Disposed Event, 121
 - Domain Indexes, 35
 - Drop Column, 69
 - DropDownList Control, 752
 - DropDownStyle, 260, 262, 264
 - Dynamic Parameters, 106–107, 274, 358, 389–391, 413, 630, 804, 821, 823, 839, 841
 - Dynamic Query, 272, 274
- E**
- Edit DataSet With Designer, 272, 281, 296
 - Edit Relationships, 43
 - ElementAt, 153, 158
 - ElementAtDefault(Of TSource), 158
 - Endpoint, 658–659, 662–663, 665, 859–860
 - Enforce Referential Integrity, 17
 - Enterprise Edition, 385–386
 - Enterprise Manager Wizard, 370
 - Entity Data Model, 137–138, 140, 143, 168
 - Entity Framework, 92, 128–133, 137–138, 210–211, 213–214
 - Entity Integrity Rule, 16, 88
 - Entity-Relationship Diagrams, 16
 - Entity-Relationship Model, 12
 - ER Diagram, 16
 - ER Notation, 20
 - Error Message, 8, 45, 301, 318, 324, 325, 332–333, 335–336, 342, 344, 356, 372, 378, 384, 389–390, 407, 518, 684, 691, 707, 741, 748, 757, 760, 767–768, 770, 772, 800
 - ExecuteNonQuery, 4, 95, 110, 111, 123, 146, 314, 371, 418–419, 443, 447, 449, 458, 468, 475–476, 487,

492–493, 497, 505, 513–518, 531, 540–541, 555–556, 558, 599, 606–607, 617, 637–638, 645, 647, 648, 717–718, 726–728, 765, 767, 771, 772, 828, 841, 850
 ExecuteReader() Method, 95, 97, 110, 145, 148, 314, 320, 358, 367, 371, 384, 391, 395, 407, 410, 571, 584, 594, 596, 682, 722, 732, 768, 770
 ExecuteScalar Method, 111
 ExitSub, 101, 204, 279, 283, 290, 301, 317, 324, 332, 360, 362, 368, 377, 396, 424, 428, 431, 433, 446, 447, 455, 458, 464, 468, 475, 476, 487, 503, 504, 506, 507, 516, 517, 531, 540, 541, 570, 599, 606, 617, 630, 637, 645, 647, 743, 748, 756, 759, 789, 798
 Extension Methods, 196, 225, 227, 229, 238, 239

F

Field Name, 38, 36
 Field Properties, 38, 40
 Field Size, 40
 Field(), 189, 192, 193
 FieldCount Property, 117
 File Processing System, 12
 File Server Databases, 28
 File System, 175
 FileNotFound Exception, 175
 Fill() Method, 122, 186, 187, 193, 250, 255, 270, 274, 292, 296, 297, 302, 319, 320, 325, 333, 344, 364, 371, 372, 378, 384, 395, 407, 410, 414, 415, 493, 556, 691
 FillFacultyReader, 328, 362, 394, 395, 586, 684
 FillSchema Method, 113
 First Normal Form, 21
 FirstOrDefault, 159
 Fixed Single, 263
 Foreign Key, 134
 Form_Load() Event Procedure, 247, 270, 284, 341, 359, 469, 502, 550, 569, 570
 FrontPage 2002 Server Extensions, 565
 FrontPage Server Extension 2000, 6, 565, 653
 FullTable View, 269

G

GetData() Methods, 270
 GetEnumerator, 156–158, 222, 234
 GetFileText(), 177
 GetOracleString, 826, 847
 GetSQLInsertCourse, 731, 748, 759, 769, 829
 GetString, 116, 594, 684
 GetTypes(), 178
 Global.asaxfile, 562, 655
 GroupBy, 281
 GroupJoin, 155, 186

H

HasChanges, 121
 HasErrors, 120
 HasRows Property, 117, 320, 325, 335, 584, 594, 596, 683, 684, 722, 732

Hide() Method, 280
 HTTPPOST, 673, 686, 720, 724, 729, 778, 780, 834, 836, 855
 HTTP Protocol, 658
 HttpApplication, 562
 HttpApplicationState Class, 570
 HttpContext Class, 570
 HttpModules, 562
 Hypertext Transfer Protocol, 658

I

IBM DB2, 10
 IEnumerable, 4, 150–162, 164, 165, 169, 170–174, 176–177, 179, 181, 183, 184, 186–188, 194–197, 217, 220, 222–224, 229, 233, 235–239, 348, 491
 IEnumerable(Of *int*), 150
 IIS Manager, 712
 IIS Virtual Directory, 664
 Image Properties, 376
 Image.FromFile, 433, 451, 701
 ImageUrl, 585, 601, 623, 640
 Implementation Phase, 18
 ImportRow, 126
 Imports System.Data, 144, 185, 188, 192, 194–195, 203–204, 316–317, 324, 332, 341, 355, 362, 368, 388, 396, 454, 455, 522, 550, 569, 570, 576, 582, 591, 592, 620, 621, 629, 630, 632, 634, 679, 683, 706, 739, 753, 786, 796, 841
 Index Organized Tables, 33
 IndexOutOfRangeException, 118
 Initialization Parameter Files, 35
 Initialized Event, 121
 Inner Join Method, 364
 InsertCommand, 94, 103, 146, 314, 420, 443, 447, 458, 468, 495, 513, 557, 727, 728, 828
 InsertOnSubmit(), 206
 Integrated Security, 101
 Integrated Database Approach, 12, 14, 16, 93, 94
 Integrated Databases, 12
 Internet Information Services, 6, 565, 661
 Invalid Operation Error, 318
 Invoke Button, 671, 675, 680, 685, 689, 692, 723, 729, 777, 779, 781, 817–818, 832, 834, 854
 IQueryable, 4, 150–153, 157–158, 161–162, 184, 197, 212–214, 233, 236–239, 491
 IS Operator, 832
 IsClosed, 116
 IsDBNull, 116
 IsDigit(), 172
 IsPostBack, 656
 Item Property, 586

J

Java Web Services, 659
 Javascript Alert() Method, 571
 Join, 30, 43, 153–155, 180, 182, 186–188, 235, 363

Joint Engine, 28

Just-In-Time Compiler, 560

K

Key Parameter, 424

Keys Folder, 54, 59, 61, 65, 611

KeyValuePair Structure, 423

L

LIKE Operator, 470

ListBox, 525

LoadDataRow, 126

Local File System, 667

Local Host, 565

Logical Design, 16

LogIn Form, 243, 259–260, 280, 283, 307, 315, 353, 522, 567

LogIn Page, 577, 630

LoginTableAdapter, 243

Login Web Form, 569, 573, 574

M

Many-To-Many Relationship, 19–20, 90

MapFacultyTable(), 327–329, 584, 586, 709

Max, 164, 180, 182, 184

Me.Close(), 291, 395, 553

Merge Methods, 120

Merge Failed Event, 121

MessageBox, 280

Microsoft Access, 2, 4, 10, 12, 28, 30, 43, 50, 52, 71, 74, 93, 97, 98, 102, 118, 146, 161, 185, 187, 190, 191, 193, 246, 247, 252–254, 266, 315, 317, 349, 356, 359, 389, 413, 453, 455–456, 458–459, 461–462, 492, 512, 556, 565

Microsoft Intermediate Language, 560, 655

Microsoft ODBC, 97, 253–254, 307

Microsoft SQL Server, 566

Microsoft.Jet.OLEDB.4.0, 98

MissingSchemaAction, 113, 123

MsgBox, 679–681, 698–699, 709, 740, 743, 744, 748, 754, 756, 759–770, 782, 787–790, 824, 842

N

Named Parameter Mapping, 107

Namespace Attribute, 670

Nested Stored Procedures, 6

Network Identification Tab, 356

newConnection, 314, 443, 513

New Database Dialog, 46

NewRow, 123, 126–128

Nonclustered Indexes, 29

Normal Forms, 21

NotNull, 68

NVARCHAR, 48, 50, 52, 143, 532

O

ObjectBrowser, 64, 66, 68, 84, 400, 487, 541, 544, 545, 549, 649, 651, 814, 831, 832, 835, 843, 850, 851

Object Explorer, 46, 50, 52, 48, 52, 59, 61, 65

Object Relational Designer, 168, 197, 200, 203, 550

Object Tool, 71

Object Query, 210–214

ODBC Data Provider, 96–97, 105, 146

ODBC Databases, 100

ODBC.NET, 96–97

OdbcCommand, 103, 110, 112

OdbcDataAdapter, 112

OdbcDataReader, 110, 115

OfType, 150, 153, 156, 170

OLE DB.NET Data Provider, 98

OleDb, 100

OLEDB Connection, 101

OleDb Namespace, 97, 453

OLEDB.NET, 96–98

OleDbCommand, 97, 103, 110, 112, 188, 194–195, 318, 320, 326, 337, 345, 413, 456, 457, 458, 459

OleDb Connection, 454

OleDb Connection Classes, 316

OleDb DataAdapter, 97, 112, 185, 188, 194–195, 318, 326, 337, 413, 456–458

OleDb DataReader, 320

OleDbException, 101–102, 317

OleDbType, 105–106, 318, 320, 326, 337, 459, 494, 557

ON clause, 469, 594

OnDeleteCascade, 650

One-To-Many Relationship, 19

One-To-One Relationship, 475, 597

Open Data Base Connectivity, 96

Open() Method, 101–102, 145, 185, 187, 193, 570

Oracle Database, 31, 67, 69, 71, 74, 93–95, 98, 147, 304, 306–307, 310, 350, 352–353, 388–393, 397, 400, 407, 416, 441, 453, 461–462, 464–466, 470–471, 480, 489, 493, 494, 496, 511, 520, 538, 541, 556–558, 565–566, 628–630, 633, 637, 641, 643–647, 649, 652, 666, 805, 807, 810, 819, 821, 829, 838, 839, 840, 842, 846

Oracle Client, 352

Oracle Data Provider, 350, 464

Oracle Database Connection, 100, 386

Oracle Database Environment, 397, 480

OracleParameter Objects, 831

OracleClient, 98

OracleCommand, 103, 110, 112, 391, 394, 395, 406, 409, 413, 465, 468, 487, 524, 525, 540–541, 631, 634, 637–638, 644–645, 647, 808, 825, 828, 830, 846, 848, 850

OracleCommand Object, 395

OracleConnection, 99, 101, 112, 352, 388, 413, 463–464, 522, 630, 808, 824–825, 828, 830, 841–842, 846, 848, 850

OracleConnection Class, 388
 Oracle DataAdapter, 112, 394, 406, 409, 413, 465, 468, 828
 OracleDataReader, 110, 115, 391, 394, 395, 406, 408–410, 413, 466, 470, 525, 631, 632, 634, 636, 644, 809, 825–826, 830–831, 846–849
 OracleParameter Objects, 390–391, 831, 848–849
 OracleSelectRTOBJECT, 314, 387–388, 417
 OracleType.Char, 495, 558
 OrderBy, 153, 156–157, 183, 222, 235
 Outer join, 155
 Output, 156

P

Package, 252, 397–405, 407, 417–478, 480, 541, 677, 810–812, 814–815, 831–832, 850
 PageEvents, 66, 569
 Parameter Classes, 103
 Parameter Collection, 103
 Parameter Mapping, 105
 Parameter Object, 102, 104–107, 147–148, 371, 584
 ParameterCollection Class, 107
 Parameter Name, 107–108
 Parameters Property, 104–105, 107, 148, 395
 Partitioned Table, 31
 Passing-By-Reference, 346
 Passing-By-Value, 684
 Password File, 31, 36
 PasswordChar Property, 278
 Perl Web Services, 659
 Physical Design, 18, 93
 PictureBox, 342, 245, 293
 Port Type, 658
 Positional Parameter Mapping, 105, 107
 Precision, 79–80, 213
 Primary Data Files, 32
 Primary Key, 17, 25–26, 29, 37, 43, 48, 52, 69, 75–76, 84, 87, 93, 123–124, 206, 261, 363, 419, 454, 471, 495–496, 517, 537, 604, 651–652, 740, 765, 770, 790, 800, 849
 Procedural Language Extension for SQL (PL-SQL), 403
 ProjectAssembly Files, 564
 ProjectDataSource, 287, 299
 ProjectAdd Windows Form, 260
 Property Page Wizard, 636
 Provider Parameter, 351

Q

Query, 717
 Query Analyzer, 370
 Query Applications, 147
 QueryBuilder, 273, 281–282, 288, 296–298, 311, 426–427, 437–438, 497, 499, 500–501
 QUERY DATA, 281

Query Method, 261–264, 295, 302–303, 331, 339–340, 342, 350, 368, 385, 411, 441, 478, 480, 489, 491, 502, 508–509, 526, 538, 548, 696, 702, 706, 710, 737, 747, 749

R

Read() Method, 329, 586
 ReadAllText(), 177
 ReadXml, 124, 126
 Redirect() Method, 577, 587, 593
 Redolog Files, 31, 36
 Reference Table Column List, 85–86, 90–91, 651
 Reference Table Name, 85–91, 651
 Referential Integrity, 94
 Referential Integrity Rules, 17
 Regular Table, 31
 RejectChanges, 126
 Relational Data Model, 10
 Relational Databases, 2, 4, 17, 94, 161, 169, 227, 604
 Relation Scollection, 118
 Remote Data Objects (RDOs), 96
 Remote Procedure Call (RPC), 658
 RemoveNodes(), 220
 Rename Column, 69
 Repeating Groups, 21
 ReplaceNodes(), 219
 Request Object, 571
 RequiredFieldValidator, 573
 Response Object, 571, 573, 577, 584, 587, 593, 707–709, 754
 RowChanged, 127
 RowDeleted, 127
 Rows Collection, 118, 124, 420
 Rows.Count, 318, 326, 337, 357, 371, 377, 394, 406, 409, 456–457, 465
 Run Stored Procedure Wizard, 473, 535, 613, 775
 Run Without Debugging, 713
 RunQuery(), 173
 Runtime Object Method, 6, 413, 442, 493, 497, 514

S

Scale, 68, 69
 Second Normal Form, 23
 Secondary Data Files, 32
 SelectCase Structure, 329
 SELECT Statement, 105, 110–111, 113–114, 116, 122, 149, 270, 273–275, 281, 288, 292, 297, 302, 348–349, 389–390, 397–399, 414–415, 429, 436, 478, 500, 584
 SelectCommand, 94, 103, 112, 114, 122, 146–147, 154, 185–188, 193–195, 314, 318–319, 326–327, 333, 336–337, 342, 344, 356–357, 371, 377–378, 390, 394, 406–407, 409–410, 443, 456–457, 465, 513, 690–691, 727–728, 828

- SelectedIndexChanged, 337, 340, 409, 459, 469–470, 521, 524–525, 590–591, 595, 597, 628, 633, 635, 750, 758–759, 786, 788–789, 794, 798
 - SelectionForm Class, 278
 - SelectMany, 157, 222
 - SelectWizardOracle Project, 307
 - Sequence Object, 75, 93
 - Server Explorer, 370
 - ServerName, 679
 - Service.asmx, 660
 - Set As StartPage, 621, 709, 761
 - SET Command, 812
 - SetAttribute(), 220
 - SetAttributeValue(), 221–222
 - SetElement(), 219–221
 - SetField(), 189, 192–194, 196
 - ShowFaculty(), 325, 329, 360, 395, 450, 451, 551, 584, 601, 623, 636, 639–640, 700–701, 707
 - Single(), 159, 194–195, 208, 553, 626
 - SingleOrDefault, 159, 208
 - SizeMode, 293, 377, 433
 - SOAP 1.2, 671
 - SOAP1.1, 671, 673, 686, 720
 - SOAP Interfaces, 693
 - SOAP Message, 658, 661, 859
 - SOAP Namespaces, 657
 - SOURCE CODE, 6
 - Source File, 567
 - SQL Commands Page, 397, 400
 - SQL Connection Object, 680
 - SQL Objects, 364
 - SQL Query, 105, 150, 152, 193, 197, 273, 497, 514, 555, 560, 571, 621
 - SQL Server Database, 31, 353, 414–415, 529, 630, 805
 - SQL Server Data Provider, 96, 104–106, 108, 110, 112, 121, 146, 354, 359, 367, 569, 594–595, 858
 - SQL Server Enterprise Manager, 370
 - SQL Server Management Studio Express, 397, 611, 615
 - SQL Statements, 281, 370, 398
 - SQL Stored Procedure, 414
 - SqlCommand Class, 114
 - SqlCommand Objects, 109
 - SqlConnection, 99, 101, 112, 114, 122, 145, 350–351, 413, 569
 - SqlConnection Objects, 112
 - SqlDataAdapter, 112
 - SqlDataAdapter Class, 114
 - SqlDataReader Class, 115
 - SqlDataReader Object, 112, 114, 122, 413, 447
 - sqlDataSet, 122
 - SqlDbType, 105, 494, 516, 557, 600, 683, 727, 769
 - SQLEXPRESS, 100, 101, 351
 - SQLMetal, 168, 197–198, 238, 492
 - SQL OLEDB Data Provider, 98
 - SqlParameter, 107
 - SqlParameter Object, 105
 - SQLSelectRTOBJECT, 353–354, 382, 417, 444
 - SQLWebSelect, 587, 628, 704
 - Standard Query Operator, 152–153, 161, 169, 179–180, 182, 197, 235
 - StartPosition, 137, 259–260, 263, 736
 - StartPosition Property, 137, 259–260
 - Static Parameter File, 35
 - StepInto Stored Procedure, 375
 - Stored Procedures, 28, 372, 374–376, 396–398, 414, 476, 480, 529, 541, 547, 555–556, 842, 850
 - Stretch Image, 377, 433
 - String.Empty, 541
 - Styleproperty, 578
 - StyleSheet, 568
 - SubmitChanges(), 205–208, 226, 491, 553, 624, 626
 - System Stored Procedures, 369
 - System.Collections.Generic, 150, 164, 183, 192, 204, 422, 423
 - System.Data.dll, 93
 - System.Data.Odbc, 96
 - System.Data.OleDb, 96–97, 185
 - System.Data.OracleClient, 96
 - System.Data.SqlClient, 96, 396
 - System.Drawing(), 586
 - System.Drawing.Image.FromFile, 377
 - System.Linq, 144, 150, 161, 164–165, 171, 178, 183, 185, 195, 204, 491
 - System.Web.Services, 660, 669–670, 679, 683, 717, 808, 841
 - System.Web.UI.WebControls.Image, 586
 - System.Xml.dll, 93
 - Systems Development Life Cycle, 16
- T**
- TableAdapter Configuration Wizard, 272
 - TableAdapter DBDirect Method, 419
 - TableAdapter Method, 289, 294, 300, 303, 324, 326, 337, 360, 362, 394–396, 406, 426, 438, 455, 502
 - TableAdapter Query Configuration Wizard, 296, 418, 425–426, 435–436, 438, 442, 494, 499, 556
 - TableAdapter.Delete(), 4, 419, 497, 494, 496, 505–506, 555–556
 - TableAdapter.Insert(), 4, 419–420, 427, 493, 496
 - TableAdapter.Update(), 419
 - Tables and Columns Dialog, 56, 61, 63
 - Table And Columns Specification, 57
 - Target Location, 218
 - TextChanged Event Procedure, 430, 450, 636, 750, 794
 - Third Normal Form (3NF), 21
 - tnsnames.ora File, 352, 386
 - ToArray(), 160
 - ToArray (Of TSource), 160, 164
 - Toolbox Window, 241, 246

ToString method, 113, 700
 ToUpper(), 232
 Transaction Log Files, 32
 Transaction Logs, 31
 Try. ... Catch Block, 102
 tuples, 30
 Typed DataSet, 119, 189, 191, 256, 700

U

UDDI, 659
 UDDI File, 693
 Untyped DataSet, 119
 Update() Method, 4, 418–420, 425–426, 428, 430, 432, 443, 492–494, 497, 504–507, 513, 555–557
 Update Command, 4, 94, 103, 112, 146, 419–420, 443, 497, 513, 555
 Updating Query String, 606, 645
 User-Defined Query, 272

V

Validate Data, 421–422, 445, 478, 489, 499, 737
 Validate(), 504
 Validation Controls, 573
 Validation Tab, 573
 Value Property, 319
 ViewCode, 285, 289, 293, 315, 323, 331, 388, 454, 462, 550, 576, 738
 ViewDesigner, 144, 274, 278, 285, 289, 297, 300, 302, 318, 320–321, 336, 357, 359, 389–390, 427, 430, 568, 576
 Views, 14, 28, 32, 28, 62, 134, 136, 247, 255, 269, 400
 Visual Basic Programmers, 96
 VisualStudio 2005, 241
 Visual Web Developer, 660

W

Web Application, 564
 Web Configuration File, 806
 Web Services Description Language (WSDL), 657
 Web Forms, 561–563, 654
 Web Interfaces, 672
 Web Methods, 4, 657–658, 661, 666, 671, 674, 677, 680, 690–692, 696, 701–702, 706, 710, 717–718, 726, 734, 738–740, 742, 747, 752, 754, 756, 759, 763, 765, 776–777, 807, 821, 823, 837, 841, 856, 858, 859
 Web Pages, 6, 560, 565, 620, 629, 654–655, 712
 Web Reference, 692
 Web Server, 4, 562, 564–565, 583, 653–654, 657–658, 667, 710, 712–713, 755, 858–859

Web Service Binding Attribute, 670
 Web Service Class, 666, 674, 678, 690, 718, 801
 Web Service Deployment, 712
 Web Service Namespace, 672
 Web Service Project, 670
 Web Service Proxy Class, 692, 695, 705, 734, 736
 Web Services, 657, 670
 Web.config, 562, 568–569, 580, 665–667, 669, 671, 678–679, 713, 804, 806, 821–822, 839–840
 Web-Based Applications, 578, 590, 661, 692, 734
 Web-Based Web Service Client Project, 703, 749, 804
 WebSelectFacultySP, 809
 Web Service Attributes, 670
 Web Service Binding, 660, 669, 679, 717, 808
 WebServiceOracleSelect, 4, 805–808, 818–819, 821, 858
 WebServiceSQLSelect, 805
 WebServiceSQLUpdateDelete, 801
 Where(), 183, 229, 235
 WinClientSQLInsert, 735
 window.close, 573, 709, 760
 Windows Components Wizard, 565
 Windows Application, 133
 Windows Authentication, 372, 663, 679
 Windows Forms, 137, 190, 199, 247, 259, 561–562
 Windows NT Security Authentication, 100
 Windows-Based Form, 568
 Windows-Based Web Service, 695, 700, 703, 708, 736, 749–750, 793, 803, 861
 Wizards, 147
 Write() Method, 163, 172, 183, 571, 577, 709, 754
 WSDL File, 693
 WSDL Terminology, 859

X–Z

XAttribute, 216–217, 220–221, 226
 XCDATA, 216
 XComment, 216–217, 223
 XDocument, 216
 XDocumentCreateNavigator(), 225
 XDocumentType, 216
 XElement, 215–221, 223–224
 XML Documents, 32, 150, 161
 XML Path Language (XPath), 561
 XML Schema (XSD), 119
 XMLWeb Service Links, 562
 XMLWeb Service, 560
 XNode, 216
 XPath, 215, 224, 21

About the Author

Dr. YING BAI is a Professor in the Department of Computer Science and Engineering at Johnson C. Smith University. His special interests include: intelligent controls, mix-language programming, fuzzy logic controls, robotic controls, robots calibrations, and fuzzy multi-criteria decision making. 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. In recent years, Dr. Bai has published more than 30 academic research papers in IEEE Transactions Journals and international conferences. He also published nine books with publishers such as Prentice Hall, CRC Press LLC, Springer, Cambridge University Press, and Wiley IEEE Press in recent years. The Russian translation of his first book, *Applications Interface Programming Using Multiple Languages*, was published by Prentice Hall in 2005. The Chinese translation of his eighth book, *Practical Database Programming with Visual C#.NET*, was translated and published by Tsinghua University Press in China at the end of 2011. Most of his books are about software programming, and serial port and database programming, as well as fuzzy logic controls in industrial applications.