

# Practical Database Programming with Visual Basic.NET



Ying Bai

CAMBRIDGE

CAMBRIDGE

[www.cambridge.org/9780521885188](http://www.cambridge.org/9780521885188)

This page intentionally left blank

## **Practical Database Programming with Visual Basic.NET**

This book teaches readers how to develop professional and practical database programs and apply auto-generated codes using Visual Basic.NET 2005 Design Tools and Wizards related to ADO.NET 2.0. The code can also be used with the newly released Visual Basic.NET 2008. Avoiding overly large blocks of code, the book shows a simple and easy way to create database programs and enables the reader to build professional and practical databases more efficiently. In addition to Design Tools and Wizards, the runtime object method is discussed and analyzed to allow users to design and implement more sophisticated data-driven applications with complicated coding techniques. Three popular database systems – Microsoft Access, Microsoft SQL Server 2005, and Oracle Database 10g Express Edition (XE) – are discussed in detail, with practical examples and sample projects that will help students, programmers, and software engineers alike.

Sample code and additional exercise questions for students as well as solutions and lecture slides for instructors are available via the Web ([www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354)).

Dr. Ying Bai is an associate professor in the Department of Computer Science and Engineering at Johnson C. Smith University in Charlotte, North Carolina. His special interests include computer architecture, software engineering, programming languages, fuzzy logic controls, automatic and robot controls, and robot calibrations. His industrial experience includes positions at Motorola MMS, Schlumberger ATE Technology, Immix TeleCom, and Lam Research. He is a senior member of the IEEE and a member of the ACM. Dr. Bai has published numerous research articles and five previous books on programming in the Windows environment and fuzzy logic controls.



# **PRACTICAL DATABASE PROGRAMMING WITH VISUAL BASIC.NET**

**Ying Bai**

Johnson C. Smith University

**Satish Bhalla (Chapter Contributor)**

Johnson C. Smith University



**CAMBRIDGE**  
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS  
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press  
The Edinburgh Building, Cambridge CB2 8RU, UK  
Published in the United States of America by Cambridge University Press, New York

[www.cambridge.org](http://www.cambridge.org)  
Information on this title: [www.cambridge.org/9780521885188](http://www.cambridge.org/9780521885188)

© Ying Bai 2009

This publication is in copyright. Subject to statutory exception and to the provision of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published in print format 2008

ISBN-13 978-0-511-45565-0 eBook (EBL)

ISBN-13 978-0-521-88518-8 hardback

ISBN-13 978-0-521-71235-4 paperback

Cambridge University Press has no responsibility for the persistence or accuracy of urls for external or third-party internet websites referred to in this publication, and does not guarantee that any content on such websites is, or will remain, accurate or appropriate.

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



# Contents

<i>Preface</i>	<i>page xxvii</i>
<i>Acknowledgments</i>	<i>xxix</i>

<b>1. Introduction</b>	1
1.1 Outstanding Features of This Book	2
1.2 Whom This Book Is For	2
1.3 What This Book Covers	3
1.4 How This Book Is Organized and How to Use This Book	5
1.5 How to Use the Source Code and Sample Databases	6
1.6 Instructor and Customer Support	7
1.7 Homework Solutions	8
<b>2. Introduction to Databases</b>	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 Developing a Database	13
2.3 Sample Database	14
2.3.1 Relational Data Model	14
2.3.2 Entity-Relationship (ER) Model	17
2.4 Identifying Keys	18
2.4.1 Primary Key and Entity Integrity	18
2.4.2 Candidate Key	19
2.4.3 Foreign Keys and Referential Integrity	19
2.5 Define Relationships	19
2.5.1 Connectivity	19
2.6 ER Notation	23
2.7 Data Normalization	23
2.7.1 First Normal Form	24
2.7.2 Second Normal Form	24
2.7.3 Third Normal Form	25

2.8	Database Components in Some Popular Databases	26
2.8.1	Microsoft Access Databases	28
2.8.1.1	Database File	29
2.8.1.2	Tables	29
2.8.1.3	Queries	29
2.8.2	SQL Server Databases	30
2.8.2.1	Data Files	30
2.8.2.2	Tables	30
2.8.2.3	Views	31
2.8.2.4	Stored Procedures	31
2.8.2.5	Keys and Relationships	31
2.8.2.6	Indexes	32
2.8.2.7	Transaction Log Files	32
2.8.3	Oracle Databases	33
2.8.3.1	Data files	33
2.8.3.2	Tables	33
2.8.3.3	Views	34
2.8.3.4	Stored Procedures	34
2.8.3.5	Indexes	35
2.8.3.6	Initialization Parameter Files	35
2.8.3.7	Control Files	35
2.8.3.8	Redo Log Files	36
2.8.3.9	Password Files	36
2.9	Create Microsoft Access Sample Database	36
2.9.1	Create the LogIn Table	36
2.9.2	Create the Faculty Table	37
2.9.3	Create the Other Tables	39
2.9.4	Create Relationships Among Tables	41
2.10	Create Microsoft SQL Server 2005 Sample Database	41
2.10.1	Create the LogIn Table	48
2.10.2	Create the Faculty Table	49
2.10.3	Create Other Tables	50
2.10.4	Create Relationships Among Tables	55
2.10.4.1	Create Relationship Between the LogIn and Faculty Tables	56
2.10.4.2	Create Relationship Between the LogIn and Student Tables	58
2.10.4.3	Create Relationship Between the Faculty and Course Tables	59
2.10.4.4	Create Relationship Between the Student and StudentCourse Tables	60
2.10.4.5	Create Relationship Between the Course and StudentCourse Tables	61
2.11	Create Oracle 10g XE Sample Database	62
2.11.1	Create the LogIn Table	64
2.11.2	Create the Faculty Table	68
2.11.3	Create Other Tables	72

2.11.4 Create the Foreign Keys for Tables	74
2.11.4.1 Create the Foreign Key Between the LogIn and Faculty Tables	74
2.11.4.2 Create the Foreign Key Between the LogIn and Student Tables	79
2.11.4.3 Create the Foreign Key Between the Course and Faculty Tables	80
2.11.4.4 Create the Foreign Key Between the StudentCourse and Student Tables	80
2.11.4.5 Create the Foreign Key Between the StudentCourse and Course Tables	82
2.12 Chapter Summary	84
2.13 Homework	84
<b>3. Introduction to ADO.NET</b>	87
3.1 ADO and ADO.NET	87
3.2 Overview of ADO.NET	88
3.3 The Architecture of ADO.NET	89
3.4 The Components of ADO.NET	91
3.4.1 The Data Provider	91
3.4.1.1 The ODBC Data Provider	92
3.4.1.2 The OLE DB Data Provider	93
3.4.1.3 The SQL Server Data Provider	93
3.4.1.4 The Oracle Data Provider	93
3.4.2 The Connection Class	94
3.4.2.1 The Open() Method of the Connection Class	96
3.4.2.2 The Close() Method of the Connection Class	97
3.4.2.3 The Dispose() Method of the Connection Class	97
3.4.3 The Command and Parameter Classes	98
3.4.3.1 The Properties of the Command Class	99
3.4.3.2 The Constructors and Properties of the Parameter Class	99
3.4.3.3 Parameter Mapping	100
3.4.3.4 The Methods of the ParameterCollection Class	102
3.4.3.5 The Constructor of the Command Class	104
3.4.3.6 The Methods of the Command Class	105
3.4.3.6.1 The ExecuteReader Method	105
3.4.3.6.2 The ExecuteScalar Method	106
3.4.3.6.3 The ExecuteNonQuery Method	106
3.4.4 The DataAdapter Class	107
3.4.4.1 The Constructor of the DataAdapter Class	108
3.4.4.2 The Properties of the DataAdapter Class	108
3.4.4.3 The Methods of the DataAdapter Class	108
3.4.4.4 The Events of the DataAdapter Class	109
3.4.5 The DataReader Class	110

3.4.6 The DataSet Component	113
3.4.6.1 The DataSet Constructor	115
3.4.6.2 The DataSet Properties	115
3.4.6.3 The DataSet Methods	115
3.4.6.4 The DataSet Events	115
3.4.7 The DataTable Component	118
3.4.7.1 The DataTable Constructor	119
3.4.7.2 The DataTable Properties	120
3.4.7.3 The DataTable Methods	120
3.4.7.4 The DataTable Events	122
3.5 Chapter Summary	124
3.6 Homework	125
<b>4. Data Selection Query with Visual Basic.NET</b>	129
PART I Data Query with Visual Basic.NET Design Tools and Wizards	130
4.1 A Completed Sample Database Application Example	130
4.2 Visual Basic.NET 2005 Design Tools and Wizards	133
4.2.1 Data Components in the Toolbox Window	133
4.2.1.1 DataSet	134
4.2.1.2 DataGridView	135
4.2.1.3 BindingSource	135
4.2.1.4 BindingNavigator	136
4.2.1.5 TableAdapter	136
4.2.2 Data Sources Window	137
4.2.2.1 Add New Data Source	137
4.2.2.2 Data Source Configuration Wizard	138
4.2.2.3 DataSet Designer	143
4.3 Build a Sample Database Project – SelectWizard	144
4.3.1 Application User Interface	145
4.3.1.1 The LogIn Form	145
4.3.1.2 The Selection Form	147
4.3.1.3 The Faculty Form	147
4.3.1.4 The Course Form	148
4.3.1.5 The Student Form	149
4.4 Add and Utilize Visual Basic Wizards and Design Tools	151
4.4.1 Add and Configure a New Data Source	151
4.5 Query and Display Data Using the DataGridView Control	155
4.5.1 View the Entire Table	155
4.5.2 View Each Record or the Specified Columns	157
4.6 Use DataSet Designer to Edit the Structure of the DataSet	159
4.7 Bind Data to Associated Controls in the LogIn Form	161
4.8 Develop Code to Query Data Using the Fill() Method	165
4.9 Use Return a Single Value to Query Data for the LogIn Form	168

4.10 Coding for the Selection Form	171
4.11 Bind Data to the Associated Controls in the Faculty Form	173
4.12 Develop Code to Query Data Using the Fill() Method	175
4.13 Display Pictures for the Faculty Form	178
4.13.1 Modify the Code for the Select Button Event Procedure	179
4.13.2 Create a Function to Select the Matched Faculty Image	180
4.14 Binding Data to Associated Controls in the Course Form	181
4.15 Develop Code to Query Data for the Course Form	185
 PART II Data Query with Runtime Objects	189
4.16 Introduction to Runtime Objects	189
4.16.1 Procedure of Building a Data-Driven Application Using Runtime Objects	191
4.17 Build a Microsoft Access Database Project – AccessSelectRTObject	192
4.17.1 Query Data Using Runtime Objects for the LogIn Form	192
4.17.1.1 Declare the Runtime Objects	193
4.17.1.2 Connect to the Data Source with the Runtime Object	193
4.17.1.3 Coding for Method 1: Using DataSet-TableAdapter to Query Data	194
4.17.1.4 Coding for Method 2: Using the DataReader to Query Data	196
4.17.1.5 Clean Up the Objects Used	197
4.17.2 Coding for the Selection Form	199
4.17.3 Query Data Using Runtime Objects for the Faculty Form	199
4.17.4 Query Data Using Runtime Objects for the Course Form	207
4.17.5 Query Data Using Runtime Objects for the Student Form	219
4.17.5.1 Coding for the Student Form_Load Event Procedure	219
4.17.5.2 Coding for the Student Select Button Event Procedure	220
4.18 Build an SQL Server Database Project – SQLSelectRTObject	226
4.18.1 Migrating from Access to SQL Server and Oracle Databases	226
4.18.2 Query Data Using Runtime Objects for the LogIn Form	229
4.18.2.1 Declare the Runtime Objects	230
4.18.2.2 Connect to the Data Source with the Runtime Object	231

4.18.2.3 Coding for Method 1: Using the TableAdapter to Query Data	232
4.18.2.4 Coding for Method 2: Using the DataReader to Query Data	233
4.18.3 Coding for the Selection Form	235
4.18.4 Query Data Using Runtime Objects for the Faculty Form	235
4.18.5 Query Data Using Runtime Objects for the Course Form	239
4.18.6 Retrieve Data from Multiple Tables Using Joined Tables	240
4.18.7 Query Data Using Runtime Objects for the Student Form	245
4.18.8 Query Data Using Stored Procedures	246
4.18.8.1 Create the Stored Procedure	247
4.18.8.2 Call the Stored Procedure	248
4.18.8.3 Query Data Using Stored Procedures for the Student Form	249
4.18.9 Query Data Using More Complicated Stored Procedures	258
4.18.10 Query Data Using Nested Stored Procedures	263
4.19 Build a Sample Oracle Database Project – OracleSelectRTOBJECT	266
4.19.1 Install Oracle Database 10g Express Edition	266
4.19.2 Configure the Oracle Database Connection String	267
4.19.3 Query Data Using Runtime Objects for the LogIn Form	268
4.19.3.1 Declare the Runtime Objects	269
4.19.3.2 Connect to the Data Source with the Runtime Object	269
4.19.3.3 Coding for Method 1: Using the TableAdapter to Query Data	270
4.19.3.4 Coding for Method 2: Using the DataReader to Query Data	272
4.19.4 Coding for the Selection Form	273
4.19.5 Query Data Using Runtime Objects for the Faculty Form	274
4.19.6 Query Data Using Runtime Objects for the Course Form	277
4.19.7 Stored Procedures in the Oracle Database Environment	280
4.19.7.1 The Syntax of Creating a Stored Procedure in Oracle	280
4.19.7.2 The Syntax for Creating a Package in Oracle	281
4.19.8 Create the Faculty_Course Package for the Course Form	283

4.19.9 Query Data Using the Oracle Package for the Course Form	286
4.20 Chapter Summary	294
4.21 Homework	295
<b>5. Data Insertion with Visual Basic.NET</b>	<b>301</b>
PART I Data Insertion with Visual Basic.NET Design Tools and Wizards	302
5.1 Insert New Data into a Database	302
5.1.1 Insert New Records into a Database Using the TableAdapter.Insert Method	303
5.1.2 Insert New Records into a Database Using the TableAdapter.Update Method	304
5.2 Insert Data into the Access Database Using a Sample Project – InsertWizard	304
5.2.1 Create a New Project Based on the SampleWizards Project	305
5.2.2 Application User Interfaces	305
5.2.3 Create the Insert Faculty Form Window	305
5.2.4 Validate Data Before the Data Insertion	308
5.2.4.1 Visual Basic Collection and .NET Framework Collection Classes	308
5.2.4.2 Validate Data Using the Generic Collection	309
5.2.5 Initialization and Termination Coding for the Data Insertion	312
5.2.6 Build the Insert Query	314
5.2.6.1 Configure the TableAdapter and Build the Data Insertion Query	314
5.2.7 Develop Code to Insert Data Using the TableAdapter.Insert Method	315
5.2.8 Develop Code to Insert Data Using the TableAdapter.Update Method	318
5.2.9 Validate Data After the Data Insertion	322
5.2.9.1 Develop Code to Retrieve the Newly Inserted Records	323
5.2.9.2 Data Binding for All Faculty-Information-Related TextBoxes	324
5.2.9.3 Develop Codes to Display the Newly Inserted Faculty Photo	327
5.3 Insert Data into the SQL Server Database Using a Sample Project – SQLInsertWizard	330
5.3.1 Modify the Existing Project to Create a New Data Insertion Project	331
5.3.2 Create a New Form Window to Insert Data for the Course Form	331
5.3.3 Project Initialization and Data Validation Before Data Insertion	332

5.3.4 Configure the TableAdapter and Build the Data Insertion Query	336
5.3.5 Develop the Code to Insert Data Using the TableAdapter.Insert Method	337
5.3.6 Perform Data Bindings for the Insert Course Form Window	341
5.3.7 Develop the Code to Insert Data Using the TableAdapter.Update Method	342
5.3.8 Data Validation for Newly Inserted Records	344
5.3.8.1 Develop Code to Perform the Data Validation	345
5.3.8.2 Use the Select Button in the Course Form to Perform the Data Validation	348
5.3.9 Insert Data into the Database Using Stored Procedures	349
5.3.9.1 Create the Stored Procedure Using the TableAdapter Query Configuration Wizard	350
5.3.9.2 Modify the Code to Perform the Data Insertion Using the Stored Procedure	350
5.4 Insert Data into the Oracle Database Using a Sample Project OracleInsertWizard	352
<b>PART II Data Insertion with Runtime Objects</b>	353
5.5 The Runtime Object Method	353
5.6 Insert Data into the SQL Server Database Using the Runtime Object Method	355
5.6.1 Insert Data into the Faculty Table for the SQL Server Database	355
5.6.1.1 Add a Data Insertion Form Window – Insert Faculty Form	355
5.6.1.2 Develop the Code to Insert Data into the Faculty Table	357
5.6.1.2.1 Validate Data Before the Data Insertion and Startup Coding	357
5.6.1.2.2 Insert Data into the Faculty Table	361
5.6.1.2.3 Validate Data After the Data Insertion	365
5.6.2 Insert a New Faculty Photo	369
5.6.3 Insert Data into the Microsoft Access Database Using Runtime Objects	372
5.6.3.1 Modify the Imports Commands	373
5.6.3.2 Modify the Database Connection String	373
5.6.3.3 Modify the LogIn Query Strings	374
5.6.3.4 Modify the Faculty Query String	375
5.6.3.5 Modifications to Other Forms	377
5.6.4 Insert Data into the Oracle Database Using Runtime Objects	379

5.6.4.1 Add the Reference and Modify the Imports Commands	380
5.6.4.2 Modify the Database Connection String	380
5.6.4.3 Modify the LogIn Query Strings	381
5.6.4.4 Modify the Faculty Query String	383
5.6.4.5 Modifications to Other Forms	384
5.7 Insert Data into the Database Using Stored Procedures	386
5.7.1 Insert Data into the SQL Server Database Using Stored Procedures	386
5.7.1.1 Add an Insert Data Form Window – Insert Course Form	386
5.7.1.2 Develop Stored Procedures for the SQL Server Database	388
5.7.1.3 Develop Code to Call Stored Procedures to Insert Data into the Course Table	390
5.7.1.3.1 Validate Data Before the Data Insertion and Startup Coding	390
5.7.1.3.2 Develop Code to Call Stored Procedures	393
5.7.1.3.3 Validate Data After the Data Insertion	396
5.7.2 Insert Data into the Oracle Database Using Stored Procedures	397
5.7.2.1 Develop Stored Procedures in the Oracle Database	398
5.7.2.2 Develop Code to Call Stored Procedures to Insert Data into the Course Table	401
5.7.2.2.1 Validate Data Before the Data Insertion and Startup Coding	401
5.7.2.2.2 Develop Code to Call Stored Procedures	401
5.7.2.2.3 Validate Data After Data Insertion	405
5.8 Chapter Summary	405
5.9 Homework	406
<b>6. Data Updating and Deleting with Visual Basic.NET</b>	411
PART I Data Updating and Deleting with Visual Basic.NET	
Design Tools and Wizards	412
6.1 Update or Delete Data from Databases	413
6.1.1 Updating and Deleting Data from Related Tables in a DataSet	413
6.1.2 Update or Delete Data from a Database by Using TableAdapter DBDirect Methods	414
6.1.3 Update or Delete Data from a Database by Using the TableAdapter.Update Method	414
6.2 Update and Delete Data from an Access Database by Using the Sample Project AccessUpdateDeleteWizard	415

6.2.1	Create a New Project Based on the InsertWizard Project	416
6.2.2	Application User Interfaces	416
6.2.2.1	Modify the Faculty Form Window	416
6.2.2.2	Bind Data for All Text Boxes of the Faculty Form Window	417
6.2.3	Validate Data Before Data Updating and Deleting	418
6.2.4	Build Update and Delete Queries	418
6.2.4.1	Configure TableAdapter and Build the Data Updating Query	418
6.2.4.2	Build the Data Deletion Query	419
6.2.5	Develop Code to Update Data by Using the TableAdapter DBDirect Method	420
6.2.5.1	Coding Modifications	420
6.2.5.2	Startup Coding	421
6.2.5.3	Update Coding	421
6.2.6	Develop Code to Update Data by Using the TableAdapter.Update Method	423
6.2.7	Develop Code to Delete Data by Using the TableAdapter DBDirect Method	425
6.2.8	Develop Code to Delete Data by Using the TableAdapter.Update Method	426
6.2.9	Validate the Data After Data Updating and Deleting	427
6.3	Update and Delete Data from a SQL Server Database by Using the Sample Project SQLUpdateDeleteWizard	430
6.4	Update and Delete Data from an Oracle Database by Using the Sample Project OracleUpdateDeleteWizard	433
<b>PART II Data Updating and Deleting with Runtime Objects</b>		434
6.5	The Runtime Object Method	434
6.6	Update and Delete Data from an SQL Server Database by Using Runtime Objects	436
6.6.1	Update Data in the Faculty Table for the SQL Server Database	436
6.6.1.1	Modify the Faculty Form Window	437
6.6.1.2	Modify the Original Coding in the Faculty Form	438
6.6.1.2.1	Modify the Coding for the Faculty Form	438
6.6.1.3	Develop Code to Update Data	441
6.6.1.4	Validate the Data Updating	443
6.6.2	Delete Data from the Faculty Table for the SQL Server Database	443
6.6.2.1	Develop Code to Delete Data	444
6.6.2.2	Validate the Data Updating and Deleting	445

6.7	Update and Delete Data for an Oracle Database by Using Runtime Objects	447
6.7.1	Add the Oracle Namespace Reference and Modify the Imports Command	449
6.7.2	Modify the Connection String and Query String for the LogIn Form	449
6.7.2.1	Modify the Connection String in the Form Load Event Procedure	449
6.7.2.2	Modify the SELECT Query String in the TabLogIn Button Event Procedure	449
6.7.2.3	Modify the SELECT Query String in the ReadLogIn Button Event Procedure	450
6.7.3	Modify the Query Strings for the Faculty Form	450
6.7.3.1	Modify the SELECT Query String for the Select Button Event Procedure	450
6.7.3.2	Modify the UPDATE Query String for the Update Button Event Procedure	451
6.7.3.3	Modify the DELETE Query String for the Delete Button Event Procedure	451
6.7.4	Modify the Query Strings for the Course Form	451
6.7.4.1	Modify the SELECT Query String for the Select Button Event Procedure	451
6.7.4.2	Modify the SELECT Query String for the CourseList Event Procedure	452
6.7.5	Modify the Query Strings for the Insert Faculty Form	452
6.7.6	Other Modifications	452
6.8	Update and Delete Data from a Database by Using Stored Procedures	454
6.8.1	Update Data in an Access Database by Using Stored Procedures	455
6.8.1.1	Modify the Existing Project	455
6.8.1.1.1	Modify the Imports Command and Connection String	456
6.8.1.1.2	Modify the Query Strings for the LogIn Button Event Procedures	456
6.8.1.1.3	Modify the Query Strings for the Select and Update Button Event Procedures	456
6.8.1.1.4	Other Modifications	457
6.8.1.2	Create Stored Procedures in the Access Database	457
6.8.1.3	Call the Stored Procedure to Update the Faculty Information	459
6.8.1.4	Confirm the Faculty Information Updating	460
6.8.2	Update Data for an SQL Server Database by Using Stored Procedures	462

6.8.2.1	Modify the Existing Project to Create a New Project	462
6.8.2.2	Develop the Stored Procedure in the SQL Server Database	465
6.8.2.3	Call the Stored Procedure to Perform the Data Updating and Validate the Updated Information	467
6.8.3	Update Data for an Oracle Database by Using Stored Procedures	469
6.8.3.1	Modify the Existing Project to Create a New Project	469
6.8.3.2	Develop the Stored Procedure in the Oracle Database	472
6.8.3.3	Call the Stored Procedure to Perform the Data Updating and Validation	475
6.8.4	Delete Data from the Oracle Database by Using Stored Procedures	476
6.8.4.1	Create the Stored Procedure in the Oracle Database	477
6.8.4.2	Develop the Code to Call the Stored Procedure to Delete Records	480
6.9	Chapter Summary	481
6.10	Homework	482
<b>7.</b>	<b>Accessing Data in ASP.NET</b>	487
7.1	What Is the .NET Framework?	488
7.2	What Is ASP.NET?	489
7.2.1	ASP.NET Web Application File Structure	491
7.2.2	ASP.NET Execution Model	491
7.2.3	What Really Happens When a Web Application Is Executed?	492
7.2.4	The Requirements to Test and Run the Web Project	493
7.3	Develop an ASP.NET Web Application to Select Data from SQL Server Databases	494
7.3.1	Create the User Interface – LogIn Form	495
7.3.2	Develop the Code to Access and Select Data from the Database	496
7.3.3	Validate the Data in the Client Side	501
7.3.4	Create the Second User Interface – Selection Page	502
7.3.5	Develop the Code to Open the Other Pages	503
7.3.6	Create the Third User Interface – Faculty Page	505
7.3.7	Develop the Code to Select the Desired Faculty Information	508
7.3.7.1	Develop the Code for the Page_Load Event Procedure	508
7.3.7.2	Develop the Code for the Select Button Event Procedure	509

7.3.7.3 Develop the Code for Other Procedures	510
7.3.8 Create the Fourth User Interface – Course Page	514
7.3.8.1 The AutoPostBack Property of the List Box Control	515
7.3.9 Develop the Code to Select the Desired Course Information	516
7.3.9.1 Coding for the Course Page Loading and Ending Event Procedures	517
7.3.9.2 Coding for the Select Button’s Click Event Procedure	519
7.3.9.3 Coding for the SelectedIndexChanged Event Procedure of the List Box Control	522
7.3.9.4 Coding for Other User-Defined Procedures	523
7.4 Develop an ASP.NET Web Application to Select Data from Oracle Databases	525
7.4.1 Modify the Connection String in the LogIn Page	526
7.4.2 Modify the Query String in the LogIn Page	527
7.4.3 Modify the Query String in the Faculty Page	528
7.4.4 Modify the Query String in the Course Page	530
7.5 Develop an ASP.NET Web Application to Insert Data into SQL Server Databases	534
7.5.1 Create a New Web Page – Insert.aspx	534
7.5.2 Develop the Code to Perform the Data Insertion Functionality	535
7.5.2.1 Develop the Code for the Page_Load and Back Button Event Procedures	536
7.5.2.2 Develop the Code for the Insert Button’s Click Event Procedure	537
7.5.2.3 Develop the Code for Other Procedures	539
7.5.3 Validate the Data Insertion	539
7.6 Develop an ASP.NET Web Application to Insert Data into Oracle Databases	544
7.6.1 Create the Insert Web Page and Develop the Code	545
7.6.1.1 Modifications to Imports Commands and Page_Load Event Procedure	546
7.6.1.2 Modifications to the Code of Subroutines and Procedures	546
7.6.2 Modify the Code for the Faculty Page	548
7.7 Develop Web Applications to Update and Delete Data in SQL Server Databases	551
7.7.1 Application User Interfaces	552
7.7.2 Modify the Code for the Faculty Page	552
7.7.3 Develop the Code for the Update Button Event Procedure	553
7.7.4 Develop the Code for the Delete Button Event Procedure	557

7.7.4.1 Relationships Between Five Tables in Our Sample Database	557
7.7.4.2 Data Deletion Sequence	558
7.7.4.3 Use the Cascade Deleting Option to Simplify the Data Deletion	559
7.7.4.4 Create the Stored Procedure to Perform the Data Deletion	561
7.7.4.5 Develop the Code to Call the Stored Procedure to Perform the Data Deletion	564
7.8 Develop an ASP.NET Web Application to Update and Delete Data in Oracle Databases	567
7.8.1 Modify the Project to Perform the Data Updating	567
7.8.1.1 Modifications to the Select Button's Click Event Procedure	567
7.8.1.2 Add the Code to the Update Button and UpdateParameters Procedures	568
7.8.2 Develop Stored Procedures to Perform the Data Deletion	570
7.8.2.1 Delete an Existing Record from the Faculty Table	571
7.8.2.2 Develop the Codes for the Delete Button's Click Event Procedure	572
7.8.2.3 Validate the Data Deleting Actions	574
7.8.2.4 The On Delete Cascade Constraint in the Data Table	575
7.9 Chapter Summary	577
7.10 Homework	578
<b>8. ASP.NET Web Services</b>	582
8.1 What Are Web Services and Their Components?	583
8.2 Procedures to Build a Web Service	585
8.2.1 The Structure of a Typical Web Service Project	585
8.2.2 The Real Considerations When Building a Web Service Project	586
8.2.3 Procedures to Build an ASP.NET Web Service	587
8.3 Build an ASP.NET Web Service Project to Access an SQL Server Database	588
8.3.1 Files and Items Created in the New Web Service Project	588
8.3.2 A Feeling of the HelloWorld Web Service Project as It Runs	590
8.3.3 Modify the Default Web Service Project	594
8.3.4 Create a Base Class to Handle Error Checking for Our Web Service	595
8.3.5 Create the Real Web Service Class	596
8.3.6 Add Web Methods into Our Web Service Class	597

8.3.7 Develop the Code for Web Methods to Perform the Web Services	598
8.3.7.1 Web Service Connection Strings	598
8.3.7.2 Modify the Existing Web Method	600
8.3.7.3 Develop the Code to Perform the Database Queries	602
8.3.7.4 Develop the Code for Subroutines Used in the Web Method	604
8.3.8 Develop a Stored Procedure to Perform the Data Query	607
8.3.8.1 Develop the Stored Procedure WebSelectFacultySP	607
8.3.8.2 Add Another Web Method to Call the Stored Procedure	608
8.3.9 Use DataSet as the Returning Object for the Web Method	609
8.3.10 Build Windows-Based Web Service Clients to Consume the Web Services	612
8.3.10.1 Create a Web Service Proxy Class	612
8.3.10.2 Develop the Graphical User Interface for the Windows-Based Client Project	614
8.3.10.3 Develop the Code to Consume the Web Service	616
8.3.10.3.1 Develop the Code for the Form_Load Event Procedure	617
8.3.10.3.2 Develop the Code for the Select Button's Click Event Procedure	618
8.3.10.3.3 Develop the Code for Other Subroutines	619
8.3.10.3.4 Develop a Subroutine ShowFaculty to Display the Faculty Image	621
8.3.11 Build Web-Based Web Service Clients to Consume the Web Service	623
8.3.11.1 Create a New Web Site Project and Add an Existing Web Page	624
8.3.11.2 Add a Web Service Reference and Modify the Web Form Window	624
8.3.11.3 Modify the Code for the Related Event Procedures	625
8.3.11.3.1 Modify the Code in the Page_Load Event Procedure	626
8.3.11.3.2 Modify the Code in the Select Button Event Procedure	626
8.3.11.3.3 Add Three User-Defined Subroutines	628
8.3.11.3.4 Modify the Code for the Back Button Event Procedure	629

8.3.12 Deploy the Completed Web Service to Production Servers	630
8.3.12.1 Copy Web Service Files to the Virtual Directory	631
8.3.12.2 Publish a Precompiled Web Service	632
8.4 Build an ASP.NET Web Service Project to Insert Data into an SQL Server Database	633
8.4.1 Modify an Existing Web Service Project	633
8.4.2 Web Service Project Development Procedure	635
8.4.3 Develop and Modify the Code for the Code-Behind Page	635
8.4.3.1 Develop and Modify the First Web Method – SetSQLInsertSP	636
8.4.3.2 Develop the Second Web Method – GetSQLInsert	640
8.4.3.3 Develop and Modify the Third Web Method – SQLInsertDataSet	642
8.4.3.4 Develop the Fourth Web Method – GetSQLInsertCourse	647
8.4.3.4.1 Create the Stored Procedure WebSelectCourseSP	648
8.4.3.4.2 Develop the Code to Call This Stored Procedure	649
8.4.4 Build Windows-Based Web Service Clients to Consume the Web Services	653
8.4.4.1 Create a Web Service Proxy Class	653
8.4.4.2 Develop the Graphical User Interface for the Client Project	654
8.4.4.3 Develop the Code to Consume the Web Service	657
8.4.4.3.1 Develop the Code to Initialize and Terminate the Client Project	657
8.4.4.3.2 Develop the Code to Insert a New Course Record into the Database	658
8.4.4.3.3 Develop the Code to Perform the Inserted Data Validation	662
8.4.4.3.4 Develop the Code to Get the Detailed Information for a Specific Course	666
8.4.5 Build Web-Based Web Service Clients to Consume the Web Services	668
8.4.5.1 Create a New Web Site Project and Add an Existing Web Page	669
8.4.5.2 Add a Web Service Reference and Modify the Web Form Window	670
8.4.5.3 Modify the Code for the Related Event Procedures	671

8.4.5.3.1	Modify the Code in the Page_Load Event Procedure	672
8.4.5.3.2	Develop Code for the Insert Button Event Procedure	673
8.4.5.3.3	Develop Code for the TextChanged Event Procedure of the CourseID Text Box	674
8.4.5.3.4	Modify the Code in the Select Button's Click Event Procedure	675
8.4.5.3.5	Modify the Code in the SelectedIndexChanged Event Procedure	678
8.4.5.3.6	Modify the Code in the Back Button's Click Event Procedure	680
8.5	Build an ASP.NET Web Service to Update and Delete Data in an SQL Server Database	682
8.5.1	Modify an Existing Web Service Project	682
8.5.2	Modify Related Web Methods	683
8.5.2.1	Modify the Web Method from SetSQLInsertSP to SQLUpdateSP	684
8.5.2.2	Modify the Web Method GetSQLInsert to GetSQLCourse	686
8.5.2.3	Modify the Web Method GetSQLInsertCourse to GetSQLCourseDetail	687
8.5.2.4	Add a New Web Method – SQLDeleteSP	689
8.5.3	Develop Two Stored Procedures – WebUpdateCourseSP and WebDeleteCourseSP	691
8.5.3.1	Develop the Stored Procedure WebUpdateCourseSP	691
8.5.3.2	Develop the Stored Procedure WebDeleteCourseSP	693
8.6	Build Windows-Based Web Service Clients to Consume the Web Services	702
8.6.1	Modifications to the File Folder and Project Files	702
8.6.2	Add a New Web Reference to Our Client Project	703
8.6.3	Modifications to the Graphical User Interface	704
8.6.4	Modifications to the Code for the Different Event Procedures	705
8.6.4.1	Modify the Code for the Form_Load Event Procedure and Form-Level Variables	705
8.6.4.2	Develop the Code for the Update Button Event Procedure	706
8.6.4.3	Develop the Code for the Delete Button Event Procedure	707
8.6.4.4	Modify the Code for the Select Button Event Procedure	708

8.6.4.5 Modify the Code for the SelectedIndexChanged Event Procedure	709
8.7 Build Web-Based Web Service Clients to Consume the Web Services	713
8.7.1 Create a New Web Site Project and Add an Existing Web Page	714
8.7.2 Add a Web Service Reference and Modify the Web Form Window	714
8.7.3 Modify the Code for the Related Event Procedures and Subroutines	716
8.7.3.1 Modify the Code for the Page_Load Event Procedure	716
8.7.3.2 Develop Code for the Update Button Event Procedure	716
8.7.3.3 Develop Code for the Delete Button's Click Event Procedure	717
8.7.3.4 Modify Code for the Select Button Event Procedure and Related Subroutines	719
8.7.3.5 Modify Code for the SelectedIndexChanged Event Procedure of the Course List Box Control and Related Subroutines	720
8.8 Build an ASP.NET Web Service Project to Access an Oracle Database	724
8.8.1 Build a Web Service Project – WebServiceOracleSelect	725
8.8.2 Modify the Connection String	726
8.8.3 Modify the Namespace Directories	726
8.8.4 Modify the Web Method GetSQLSelect and Related Subroutines	726
8.8.5 Modify the Web Method GetSQLSelectSP and Related Subroutines	728
8.8.5.1 Modifications to the Stored Procedure WebSelectFacultySP	729
8.8.5.2 Modifications to the Code for the Web Method GetSQLSelectSP	733
8.8.6 Modify the Web Method GetSQLSelectDataSet	734
8.9 Build Web Service Clients to Consume the Web Service WebServiceOracleSelect	739
8.10 Build an ASP.NET Web Service Project to Insert Data into an Oracle Database	740
8.10.1 Build a Web Service Project – WebServiceOracleInsert	740
8.10.2 Modify the Connection String	741
8.10.3 Modify the Namespace Directories	741
8.10.4 Modify the Web Method SetSQLInsertSP and Related Subroutines	742

8.10.5 Modify the Web Method GetSQLInsert and Related Subroutines	744
8.10.6 Modify the Web Method SQLInsertDataSet	746
8.10.7 Modify the Web Method GetSQLInsertCourse and Related Subroutines	748
8.11 Build Web Service Clients to Consume the Web Service WebServiceOracleInsert	757
8.12 Build an ASP.NET Web Service to Update and Delete Data in an Oracle Database	758
8.12.1 Build a Web Service Project – WebServiceOracleUpdateDelete	758
8.12.2 Modify the Connection String	759
8.12.3 Modify the Namespace Directories	759
8.12.4 Modify the Web Method SQLUpdateSP and Related Subroutines	760
8.12.4.1 Develop the Stored Procedure UpdateCourse_SP	762
8.12.5 Modify the Web Method GetSQLCourse and Related Subroutines	765
8.12.6 Modify the Web Method GetSQLCourseDetail and Related Subroutines	766
8.12.7 Modify the Web Method SQLDeleteSP	768
8.12.7.1 Develop the Stored Procedure WebDeleteCourseSP	770
8.13 Build Web Service Clients to Consume the Web Service WebServiceOracleUpdateDelete	774
8.14 Chapter Summary	775
8.15 Homework	776
<i>Index</i>	781



# 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 on the basis of teenagers' music preferences. Grocery stores display more popular items at eye level, and reorders are based on the inventories taken at regular intervals. Other examples include patients' records in hospitals, bank customers' account information, book orders by libraries, club memberships, auto part orders, winter clothing stock in department stores, and many more.

In addition to database management systems, in order to effectively apply and implement databases in real industrial or commercial systems, a good graphical 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 provide this GUI functionality. Unlike other programming languages, Visual Basic.NET is easy to learn and easy to understand, with a low learning curve. Beginning with 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 the .NET Framework that makes Internet and Web programming easy and simple. Any language integrated in to 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 the .NET Framework. The former provides all the components, including the Data Providers, DataSet, and DataTable, needed to access and manipulate data from different databases. The latter provides support to develop Web applications and Web services 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 2005 and relational databases such as Microsoft Access, Microsoft SQL Server 2005, and Oracle Database 10g Express Edition (XE). The book provides a detailed description of the practical considerations and applications in database

programming with Visual Basic 2005, along with authentic examples and detailed explanations. More important, a new writing style implemented in this book, combined with real examples, provides readers with a clear picture of how to handle database programming issues in the Visual Basic.NET 2005 environment.

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

1. A unique 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 huge blocks of code, as is common in traditional database programming books.
2. A real, completed sample database, CSE\_DEPT, with three versions (Microsoft Access, SQL Server 2005, and Oracle Database 10g XE), is provided and used for the entire book. Step-by-step, detailed illustrations and descriptions about how to design and build a practical relational database are provided.
3. Both fundamental and advanced database programming techniques are covered for the convenience of both beginning students and experienced programmers.
4. Three types of popular databases are covered and discussed in detail with practical sample examples: Microsoft Access, SQL Server 2005, and Oracle Database 10g XE.
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 OLE DB to access the SQL Server or Oracle databases, real SQL Server and Oracle data providers are utilized to connect to Visual Basic.NET 2005 directly to perform data operations.
6. It is a good textbook for college students and a good reference book for programmers, software engineers, and academic researchers.

I sincerely hope that this book will help readers or users develop and build professional and practical database applications.

# Acknowledgments

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

I would also like to thank Dr. Satish Bhalla, who is the chapter contributor for this book. He is a specialist in database programming and management, especially in SQL Server, Oracle, and DB2. Dr. Bhalla spent a lot of time preparing materials for Chapter 2, and he deserves thanks for this.

Many thanks to my editor, Heather Bergman, for helping to make this book a reality. You would not have found this book in the market without her deep perspective and hard work. Thanks are also extended to the editing team of this book. Without their contributions, it would have been impossible for this book to get published.

Thanks should also be extended to the following book reviewers for their invaluable opinions on this book:

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

Last but not least, I thank all the people who supported me in finishing this book.



## **Practical Database Programming with Visual Basic.NET**



---

**1**

---

## Introduction

For years, during my teaching of database programming and Visual Basic.NET programming at the college level, I found it so difficult to find a good textbook for these topics that I had to combine a few different professional books together as references to teach my course. Most of those books are specially designed for programmers or software engineers, cover a lot of programming strategies, and include huge blocks of code, which cause terrible headaches for college students or beginning programmers who are new to Visual Basic.NET and database programming. I had to prepare my class presentations and figure out all the homework and exercises for my students. I dreamed that one day I would find a good textbook that would be suitable for college students or beginning programmers to help them learn and master database programming with Visual Basic.NET easily and conveniently. Finally I decided that I needed to do something about this dream myself after a long period of waiting.

Another reason I had the idea for this book is the job market. As you know, most industrial and commercial companies in the United States are database application businesses, such as manufacturers, banks, hospitals, and retail stores. A 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 a book like this one.

Unlike most of the database programming books in the current market, which discuss and present database programming techniques with huge blocks of programming code from the first page to the last page, this book uses a new writing style to show readers, especially college students, how to develop professional and practical database programs in Visual Basic.NET by using Visual Basic.NET 2005 Design Tools and Wizards related to ADO.NET 2.0 and how to apply code that is auto-generated by using Wizards. With this new style, the headache caused by those huge blocks of programming code is eliminated. Instead, a simple and easy way to create database programs using the Design Tools can be developed to attract students'

interest and enable students to build professional and practical databases in more efficient and interesting ways.

There are so many different database programming books available in the market, but one can rarely find a book like this one, which implements a new writing style to hold students' interest. To meet the needs of experienced or advanced students and software engineers, the book contains two programming methods: the fundamental database programming method using Visual Basic.NET 2005 Design Tools and Wizards and the advanced database programming method using the runtime object method. In the second method, all database-related objects are created and applied while the project is running by utilizing a few blocks of code.

## **1.1 OUTSTANDING FEATURES OF THIS BOOK**

1. A new writing style is adopted to increase students' or beginning programmers' interest in learning and developing practical database programs and to avoid the headache caused by the huge blocks of code found in traditional database programming books.
2. A real, completed sample database, CSE\_DEPT, with three versions (Microsoft Access, SQL Server 2005, and Oracle Database 10g XE), is provided and used throughout the book. A detailed illustration and description of how to design and build a practical relational database is provided step by step.
3. The book covers both fundamental and advanced database programming techniques for the convenience of both beginning students and experienced programmers.
4. Three types of popular databases are covered and discussed in detail, with practical examples: Microsoft Access, Microsoft SQL Server 2005, and Oracle Database 10g Express Edition (XE).
5. Various actual data providers are discussed and implemented in the sample projects, such as SQL Server and Oracle data providers. Instead of using OLE DB to access the SQL Server or Oracle databases, real SQL Server and Oracle data providers are utilized to connect to Visual Basic.NET 2005 directly to perform data operations.
6. The book provides homework and exercises, which allow users to practice what they have learned by doing some exercises themselves.
7. It is a good textbook for college students and a good reference book for programmers, software engineers, and academic researchers.

## **1.2 WHOM 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 Access, SQL Server 2005, and Oracle Database 10g XE. A fundamental knowledge and understanding of Visual Basic.NET and Visual Studio.NET IDE is assumed.

### 1.3 WHAT THIS BOOK COVERS

Eight chapters are included in this book. The contents of each chapter can be summarized as follows:

- Chapter 1 provides an introduction to and summary of the whole book.
- Chapter 2 provides a detailed discussion and analysis of the structure and components of relational databases. Some key technologies in developing and designing a database are also discussed in this part. The procedure and components used to develop a practical relational database with three database versions, namely, Access, SQL Server 2005, and Oracle Database 10g XE, are analyzed in detail with real data tables in the sample database CSE\_DEPT.
- Chapter 3 provides an introduction to ADO.NET that includes the architecture, organization, and components of ADO.NET. It also includes detailed discussions and descriptions to give readers both fundamental and practical ideas and pictures of how to use components in ADO.NET to develop professional data-driven applications. Two ADO.NET architectures are discussed to enable users to design and build their preferred projects on the basis of the different organizations of ADO.NET. Four popular data providers, OLE DB, ODBC, SQL Server, and Oracle, are discussed in detail. The basic ideas of DataTable and DataSet are also analyzed and described with some real coding examples.
- Starting with Chapter 4, actual database programming techniques of Visual Basic.NET 2005, such as data selection queries, are provided and discussed. This chapter comprises two parts: Part I contains detailed descriptions of how to develop professional data-driven applications with the help of Visual Basic.NET design tools and wizards and includes two real projects as examples. This part contains a lot of hidden code that is created by Visual Basic.NET automatically when using the design tools and wizards. Therefore the coding in this part is very simple and easy. Part II covers an advanced technique, the runtime object method, for developing and building professional data-driven applications. Detailed discussions and descriptions of how to build professional and practical database applications using the runtime method are provided, along with four real projects.
- Chapter 5 provides detailed discussions and analyses of three popular data insertion methods with three different databases: Access, SQL Server 2005, and Oracle.
  1. Using TableAdapter's DBDirect and TableAdapter.Insert() methods
  2. Using the TableAdapter's Update() method to insert new records that have already been added to the DataTable in the DataSet
  3. Using the Command object's ExecuteNonQuery() method

This chapter is divided into two parts. Methods 1 and 2 are related to Visual Basic.NET design tools and wizards and therefore are covered in Part I. The third method is related to runtime objects and therefore is covered in Part II.

Nine real projects are used to illustrate how to perform data insertion in three different databases: Access, SQL Server 2005, and Oracle Database 10g XE. Some professional and practical data validation methods are also discussed in this chapter to confirm the data insertion.

- Chapter 6 provides discussions and analyses of three popular data updating and deleting methods:

1. Using TableAdapter DBDirect methods, such as TableAdapter.Update() and TableAdapter.Delete(), to update and delete data directly from the database
2. Using the TableAdapter.Update() method to update and execute the associated TableAdapter' properties, such as UpdateCommand and DeleteCommand, to save changes made in the table in the DataSet to the table in the database
3. Using the runtime object method to develop and execute the Command class's ExecuteNonQuery() method to update or delete data from the database directly

This chapter is also divided into two parts. Methods 1 and 2 are related to Visual Basic.NET design tools and wizards and therefore are covered in Part I. The third method is related to runtime objects and is covered in Part II. Seven real projects are used to illustrate how to perform data updating and deleting in three different databases: Access, SQL Server 2005, and Oracle Database 10g XE. Some professional and practical data validation methods are also discussed in this chapter to confirm the data updating and deleting actions. The key points to remember when updating and deleting data in a relational database, such as the order in which to execute data updating and deleting in parent and child tables, are also discussed and analyzed.

- Chapter 7 provides introductions to and discussions about the development and implementation of ASP.NET Web applications in the Visual Basic.NET 2005 environment. At the beginning of Chapter 7, detailed and complete descriptions ASP.NET and Microsoft .NET Framework are provided. This part is especially useful and important to students or programmers who do not have any knowledge of or background in Web application development and implementation. Following the introduction section, a detailed discussion of how to install and configure the environment to develop ASP.NET Web applications is provided. Some essential tools, such as the Web server, Internet Information Services (IISs), and FrontPage Server Extension 2000 or 2002, and the installation process for these tools are introduced and discussed in detail. Starting with Section 7.3, the detailed development and building processes 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 these development and building processes.
- Chapter 8 provides introductions to and discussions about the development and implementation of ASP.NET Web services in the Visual Basic.NET 2005 environment. A detailed discussion and analysis of the structure and

components of 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:

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. In all, eighteen projects, including the Web service projects and the associated Web service client projects, are developed in this chapter. All projects have been debugged and tested and can be run in Windows-compatible operating systems such as Windows 95, 98, 2000, XP, and Vista.

## **1.4 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 have experience with this topic.

Chapters 2 and 3 provide the fundamentals of database structures and components of ADO.NET. Chapters 4, 5, and 6 are each divided into two parts, a fundamental part and an advanced part. The data-driven applications developed with design tools and wizards provided by Visual Basic.NET in the fundamental part have less coding, and therefore are more suitable for students or programmers who are new to database programming with Visual Basic.NET. Part II contains the runtime object method and covers the development of code to perform different data actions on the database. This method is more flexible and convenient for experienced programmers because a lot of coding is required.

Chapters 7 and 8 give a full discussion and analysis the development and implementation of ASP.NET Web applications and Web services. These technologies are necessary for students and programmers who want to develop and build Web applications and Web services to access and manipulate data through the Internet.

This book is organized so that it can be used at two levels, which are shown in Figure 1.1.

It is highly recommended that undergraduate college students and beginning programmers learn and understand the contents of Chapters 2 and 3 and Part I of Chapters 4, 5, and 6 since those provide fundamental knowledge and techniques of database programming with Visual Basic.NET 2005. Chapters 7 and 8 are optional for instructors and depend on the time and schedule.

Step by step, a detailed description is given to illustrate how to design and set up relationships between parent and child tables using the primary key and foreign

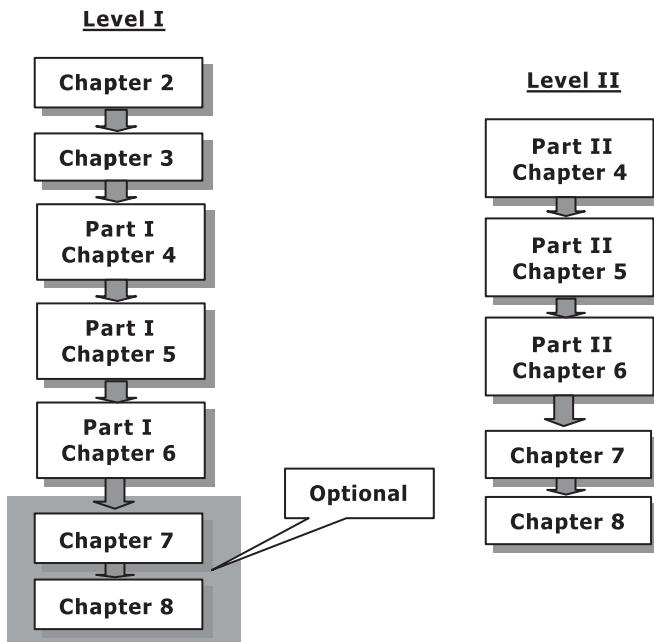


Figure 1.1. Two levels in this book.

keys for Access, SQL Server 2005, and Oracle Database 10g XE databases. In Part I of Chapters 4–6, Visual Basic.NET design tools and wizards are discussed and analyzed in detail to show readers how to use them to easily and conveniently design and build professional database programs with Visual Basic.NET.

For experienced college students or software programmers who already have some knowledge of database programming, it is recommended to learn and understand the contents of Part II of Chapters 4–6 as well as Chapters 7 and 8 since the runtime data object method and some sophisticated database programming techniques, such as joined-table query, nested stored procedures, and Oracle Package, are discussed and illustrated with real examples. Also, ASP.NET Web applications and ASP.NET Web services are discussed and analyzed with twenty-four real database program examples for SQL Server 2005 and Oracle Database 10g XE.

## 1.5 HOW TO USE THE SOURCE CODE AND SAMPLE DATABASES

All source code for each real project developed in this book is available. All projects are categorized by chapter in the folder **DBProjects**, which is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). (See Figure 1.2.)

You can copy or download the code onto your computer and run each project as you like. To successfully run these projects on your computer, the following conditions must be met:

- Visual Studio.NET 2005 or later must be installed on your computer.

- Three databases management systems, Access (Microsoft Office), SQL Server 2005 Management Studio Express, and Oracle Database 10g XE, must be installed on your computer.
- Three versions of the sample database, CSE\_DEPT.mdb, CSE\_DEPT.mdf, and the Oracle version of CSE\_DEPT, must be installed on your computer in appropriate folders.
- To run the projects developed in Chapters 7 and 8, in addition to the conditions listed above, Internet Information Services (IIS) such as FrontPage Server Extension 2000 or 2002 must be installed on your computer and must work as a pseudo server for those projects.

The following appendixes provide useful references and practical knowledge to install database management systems and develop actual database application projects:

- **Appendix A:** Provides a complete SQL commands reference
- **Appendix B:** Provides detailed descriptions of downloading and installing Microsoft SQL Server 2005 Express
- **Appendix C:** Provides detailed descriptions of downloading and installing Oracle Database 10g XE
- **Appendix D:** Provides detailed discussions of how to create a user database in Oracle Database 10g XE and how to duplicate the Oracle 10g user database using Unload and Load methods
- **Appendix E:** Provides detailed discussions of how to add and connect the Oracle 10g XE database into Visual Basic.NET applications using Visual Basic Design Tools and Wizards
- **Appendix F:** Provides detailed discussions of how to use the three sample databases: CSE\_DEPT.mdb, CSE\_DEPT.mdf, and the Oracle version of CSE\_DEPT

All these appendixes can be found in the **appendix** folder available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

Three sample databases, CSE\_DEPT.mdb, CSE\_DEPT.mdf, and the Oracle version of CSE\_DEPT, are located in the **database** folder available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). To use these databases for your applications or sample projects, refer to Appendix F.

## 1.6 INSTRUCTOR AND CUSTOMER SUPPORT

The teaching materials for all chapters are available in a sequence of Microsoft PowerPoint files, one file for each chapter. Interested instructors can find these teaching materials in the folder **Teaching-PPT** [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

E-mail support is available to readers of this book. When you send an e-mail message, please provide the following information:

- A detailed description of your problem, including the error message and debug message as well as the error or debug number if it is provided
- Your name, job title, and company name

Please send all questions to the e-mail address [bai\\_db\\_book@bellsouth.net](mailto:bai_db_book@bellsouth.net). Questions will be answered as quickly as possible.

## **1.7 HOMEWORK SOLUTIONS**

Selected home work solutions are available for instructors at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

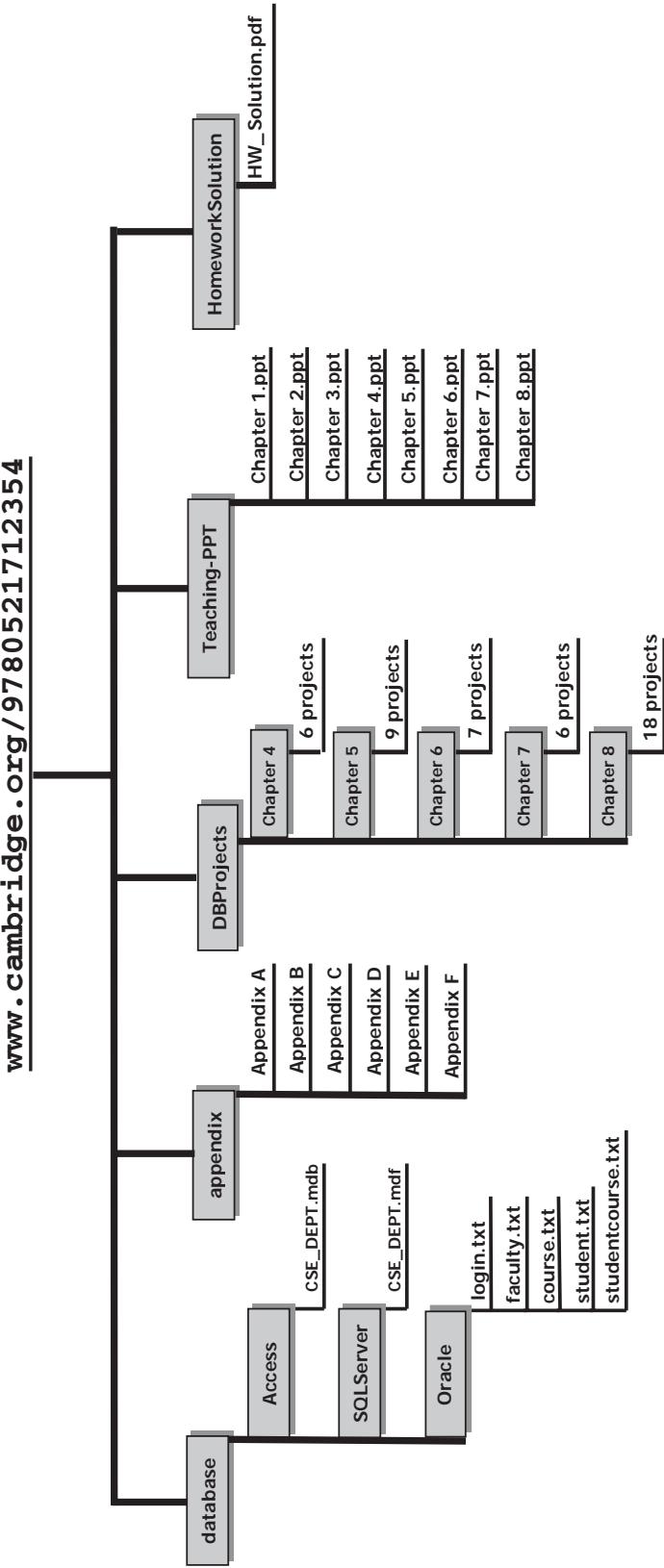


Figure 1.2. Book-Related Materials on the Web Site.

---

**2**

---

## Introduction to Databases

Databases have become an integral part of our modern-day life. We are an information-driven society. We generate large amounts of data that are 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 Institutes of Health. Many countries in the world participated in this venture for ten years. The project was a tremendous success. It was completed in 2003 and resulted in huge amounts of genome data, currently stored in databases around the world. Scientists will be analyzing this data in the years to come.

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

Database management programs have been in existence since the sixties. However, it was not until the seventies, when E. F. Codd proposed the then revolutionary relational data model, that database technology really took off. In the early eighties it received a further boost with the arrival of personal computers (PCs) and microcomputer-based data management programs like dBase II (later followed by dBase III and dBase IV). Today, we have a plethora of vastly improved programs for PCs and mainframe computers, including Microsoft Access, IBM DB2, Oracle, SQL Server, MySQL, 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 the following:

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

## 2.1 WHAT ARE DATABASES AND DATABASE PROGRAMS?

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

### 2.1.1 A file processing system

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

A file processing system may be advantageous under certain circumstances. For example, if the data is static and a simple application will solve the problem, a more expensive DBMS is not needed. For example, in a small business environment you want to keep track of the inventory of 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 hire a knowledgeable database administrator. Before DBMSs became popular, data was kept in files and application programs were developed to delete, insert, or modify records in the files. Since specific application programs were developed for specific data, these programs lasted for months or years before modifications were necessitated by business needs.

### **2.1.2 Integrated Databases**

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

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

1. **Data sharing:** The data in the database is available to a large numbers of users who can access the data simultaneously and create reports, as well as manipulate the data given proper authorization and rights.
2. **Minimizing data redundancy:** Since all the related data exists in a single database, there is a minimal need for data duplication. The duplication is needed to maintain relationships between various data items.
3. **Data consistency and data integrity:** Reducing data redundancy leads to data consistency. Since data is stored in a single database, enforcing data integrity becomes much easier. Furthermore, the inherent functions of the DBMS can be used to enforce the integrity with minimum programming.
4. **Enforcing standards:** DBAs are charged with enforcing standards in an organization. They take into account the needs of various departments and balance them against the overall needs of the organization. They define various rules such as documentation standards, naming conventions, and update and recovery procedures. It is relatively easy to enforce these rules in a database management 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, using data encryption, and providing access to restricted views of the database.
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 is mainly the exorbitant costs associated with it. The hardware, software, and maintenance are expensive. Providing security, concurrency, integrity, and recovery may add further to this cost. Furthermore, since a DBMS consists of a complex set of programs, trained personnel are needed to maintain it.

## 2.2 DEVELOPING A DATABASE

The database development process may follow a classical Systems Development Life Cycle.

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

An alternative approach to developing a database is through a phased process that includes designing a conceptual model of the system that imitates the real-world operation. It should be flexible and change when the information in the database changes. Furthermore, it should not depend on the physical implementation. This process includes the following phases:

1. **Planning and Analysis:** This phase is roughly equivalent to the first three steps mentioned above in the Systems Development Life Cycle: identify requirements and specifications, evaluate alternatives, and determine 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 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, the attributes that will represent fields in a table, and each entity-attribute relationship. Entity-relationship diagrams provide a good representation of the conceptual design.

**Table 2.1.** LogIn Table

<b>user_name</b>	<b>pass_word</b>	<b>faculty_id</b>	<b>student_id</b>
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	

3. **Logical Design:** Transform the conceptual design into a logical design by creating a roadmap of how the database will look before actually creating the database. Identify a data model; usually it is the relational model. Define the tables (entities) and fields (attributes). Identify primary and foreign keys for each table. Define relationships between the tables.
4. **Physical Design:** Develop physical data structures, and specify file organization, data storage, and so forth. 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 it 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 is stored in five tables: LogIn, Faculty, Course, Student, and StudentCourse. A table consists of rows and columns (Figure 2.1.). A row represents a record, and a column represents a field. A row is called a tuple, and a column an attribute. For example, the Student table (Table 2.4) has seven columns or fields: student\_id, name, gpa, credits, major, schoolYear, and email. It has five records or rows.

### 2.3.1 Relational Data Model

A data model is like a blueprint for developing a database. It describes the structure of the database and various data relationships and constraints on the data. This information is used in building tables and keys and in defining relationships. A relational model implies that a user perceives the database as made up of relations, which is database jargon for tables. It is imperative that all data elements

**Table 2.2. Faculty Table**

faculty_id	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	Assistant Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sballa@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	Jenny King	MTC-324	750-378-1230	East Florida University	Professor	jkking@college.edu

**Table 2.3. Course Table**

<b>course_id</b>	<b>course</b>	<b>credit</b>	<b>classroom</b>	<b>schedule</b>	<b>enrollment</b>	<b>faculty_id</b>
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 & Algorithm	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	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

in the tables be represented correctly. To achieve this goal, designers use various tools. The most commonly used tool is the entity-relationship (ER) model. A well-planned model will give consistent results and allow changes if needed later on. The following section further elaborates on the ER model.

### 2.3.2 Entity-Relationship (ER) Model

The ER model was first proposed and developed by Peter Chen in 1976. Since then, Charles Bachman and James Martin have added some refinements. The model was designed to communicate the database design in the form of a conceptual schema. The ER model is based on the perception that the real world is made up of entities, their attributes, and relationships. The model is graphically depicted in

**Table 2.5.** StudentCourse Table

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

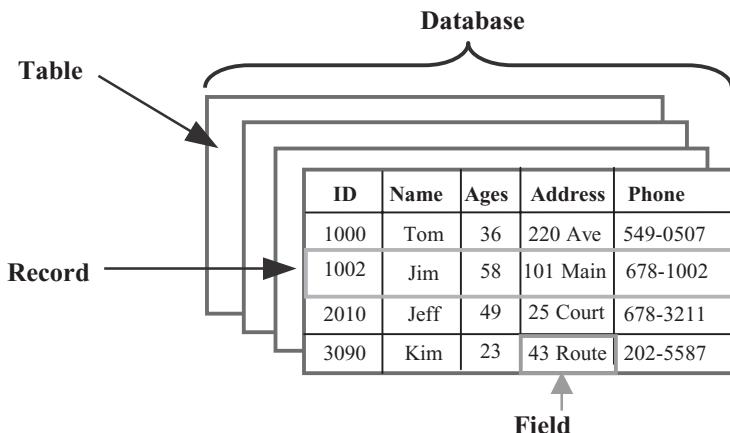


Figure 2.1. Records and fields in a table.

entity-relationship diagrams (ERDs). ERDs are a major modeling tool; they graphically describe the logical structure of the database. ERDs can be used with ease to construct relational tables and are a good vehicle for communicating the database design to the end user or a developer. The three major components of an ERD are entities, relationships, and 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, place, or thing. An entity in an ERD translates into a table and 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 of these entities is translated into a table with the same name. An instance of the Faculty entity is Alice Brown and her attributes.

**Relationships:** A database is made up of related entities. There are natural associations between the entities that are referred to as *relationships*. For example,

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

The number of occurrences of one entity associated with a 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 Systems is one attribute of an entity called Courses in the CSE-DEPT database. The domain of an attribute is a set of all possible values for the attribute.

## 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, in

which case it is referred to as a *composite key*. The *entity integrity rule* states that no attribute that is a member of the primary (composite) key may accept a null value.

A Faculty ID may serve as a primary key for the Faculty entity, assuming that all faculty members have been assigned a unique Faculty ID. 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 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 in the primary key field will fail.

### 2.4.2 Candidate Key

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

### 2.4.3 Foreign Keys and Referential Integrity

Foreign keys are used to create relationships between tables. A foreign key is an attribute in one table whose values are required to match those of a 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. Microsoft Access also provides for cascade update and cascade delete, which imply that changes made in one of two related tables will be reflected in the other.

Consider the two tables Course and Faculty in the sample database, CSE\_DEPT. The Course table has a foreign key titled faculty\_id, which is a primary key in the Faculty table. The two tables are logically related through the faculty\_id link. The referential integrity rule implies that we may not add a record to the Course table with a faculty\_id that is not listed in the Faculty table. In other words, there must be a logical link between the two related tables. Secondly, if we change or delete a faculty\_id in the Faculty table, it must be reflected 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*, or *many-to-many*. In ERDs these are indicated by

**Table 2.6. Faculty Table**

<b>faculty_id</b>	<b>name</b>	<b>office</b>	<b>phone</b>	<b>college</b>	<b>title</b>	<b>email</b>
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	Assistant Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenny King	MTC-324	750-378-1230	East Florida University	Professor	jkking@college.edu

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

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

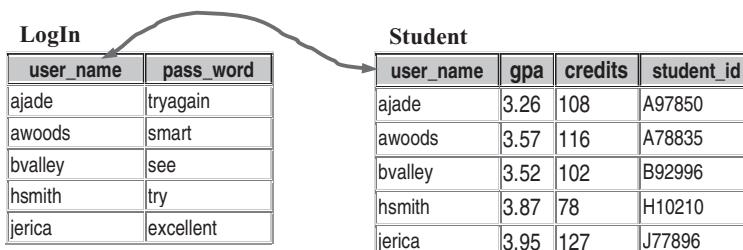
Faculty (Partial data shown)

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

placing 1, M, or N at one of the two ends of the relationship diagram. Figures 2.2–2.4 illustrates the various types of connectivity.

- 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 student\_id in the Student table (Figure 2.2).
- A *one-to-many* (1:M) relationship occurs when one instance of entity A is associated with zero, one, or many instances of entity B. However, entity B is associated with only one instance of entity A. For example, one department can have many faculty members; each faculty member is assigned to only one department. In the CSE\_DEPT database, a one-to-many relationship is represented by faculty\_id in the Faculty table and course\_id in the Course table (Figure 2.3).
- A *many-to-many* (M:N) relationship occurs when one instance of entity A is associated with zero, one, or many instances of entity B and one instance of entity B is associated with zero, one, or many instances of entity A. For example, a student may take many courses and a course may be taken by more than one student (Figure 2.4).

In the CSE\_DEPT database, a many-to-many relationship can be realized by using the third table. For example, in this case, the StudentCourse that works as the

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

**Faculty**

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

**Course**

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

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

third table sets a many-to-many relationship between the Student and the Course tables (Figure 2.4).

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 member. If one wants to develop a Course table that contains courses taught by all faculty in more than one semester, a third table, say, FacultyCourse, should be created to set up a many-to-many relationship between the Faculty and Course tables, since one course may be taught by different faculty members in different semesters.

The relationships in the CSE\_DEPT database are summarized in Figure 2.5.

The five entities are LogIn, Faculty, Course, Student, and StudentCourse.

**Student**

student_id	name	gpa	credits
A78835	Andrew Woods	3.26	108
A97850	Ashly Jade	3.57	116
B92996	Blue Valley	3.52	102
H10210	Holes Smith	3.87	78
J77896	Erica Johnson	3.95	127

**Course**

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

**StudentCourse**

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

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

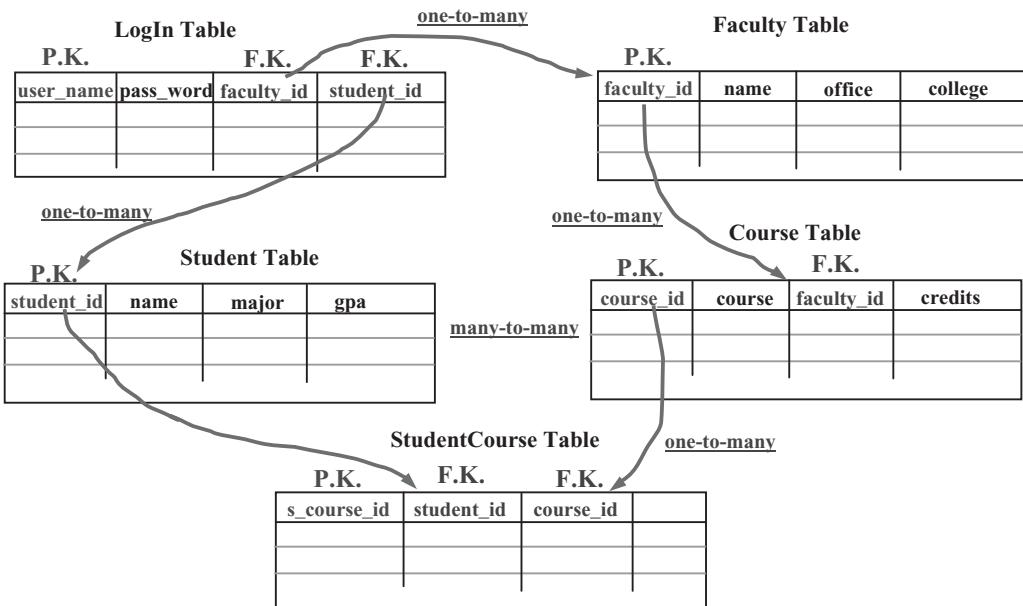


Figure 2.5. Relationships in the CSE\_DEPT database. P.K. and F.K. represent the primary key and the foreign key, respectively.

Figure 2.6 displays the Microsoft Access relationship diagram among the various tables in the CSE\_DEPT database. The one-to-many relationships are indicated by placing 1 at one end of the link and  $\infty$  at the other. The many-to-many relationship between the Student and the Course table was broken down into two one-to-many relationships by creating the 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. A number of drawing tools are available to draw ERDs. These include ER Assistant, Microsoft Visio, and SmartDraw, among others. The commonly used notation is 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, inefficiencies, and update and deletion anomalies that usually occur in 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

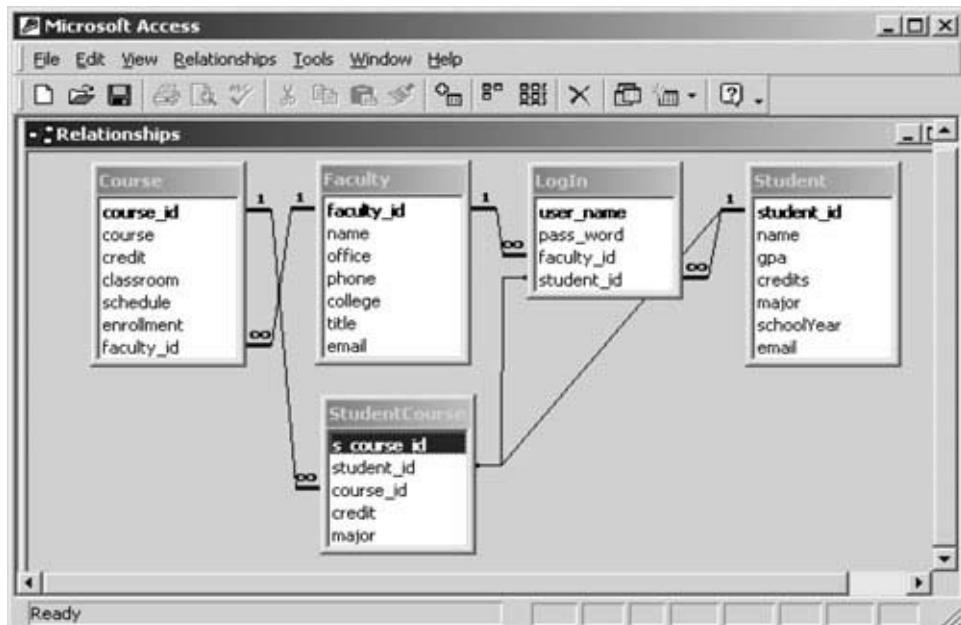


Figure 2.6. Relationships in the CSE\_DEPT database illustrated using Microsoft Access.

improves the database and makes it more efficient. In other words a database that is in second normal form is better than one in first normal form, and one in third normal form is better than one in second normal form. To be in third normal form, a database has to be in first and second normal forms. There are fourth and fifth normal forms, but for most practical purposes a database meeting the criteria of the third normal form is considered to be of good design.

### 2.7.1 First Normal Form

A table is in first normal form if the values in each column are atomic, that is, there are no repeating groups of data.

The Faculty table in 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.

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

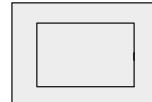
### 2.7.2 Second Normal Form

A table is in second normal form (2NF) if it is already in 1NF and every non-key column is fully dependent on the primary key.

An entity is represented by a rectangular box.



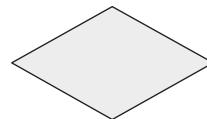
A weak entity is represented by a double rectangular box.



An attribute is represented by an oval.



A 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.

This implies that if the primary key consists of a single column, then a table in 1NF is automatically in 2NF. The second part of the definition implies that if the key is composite, then none of the non-key columns will depend on 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 into two new tables, Faculty and Office (Figure 2.8), 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 non-key columns in both tables now depend only on the primary keys only.

### 2.7.3 Third Normal Form

A table is in third normal form (3NF) if it is already in 2NF and every non-key column is nontransitively dependent on the primary key. In other words, all non-key

**Table 2.8.** Unnormalized Faculty Table with Repeating Groups

<b>faculty_id</b>	<b>name</b>	<b>office</b>	<b>phone</b>
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

columns are mutually independent, but at the same time they are fully dependent on the primary key only.

Another way of stating this is that in order to achieve 3NF, no column should depend on any non-key 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 the only determinants it contains are candidate keys.

In the Faculty table in Figure 2.8, all non-key columns depend on the primary key, *faculty\_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.

## 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

**Table 2.9.** Normalized Faculty Table

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

Old Faculty table in 1NF

faculty_id	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	750-330-1230



New Faculty table

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

New Office table

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

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

**Table 2.10.** The Old Course Table

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

**Table 2.11.** The New Course Table

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

components, such as views and stored procedures, are collectively called database objects. In this section we will discuss various objects that make up Microsoft Access, SQL Server, and Oracle databases.

There are two major types of databases, *file server* and *client server*.

In a file server database, data is stored in a file and each user of the database retrieves the data, displays the data, or modifies the data directly from or to the file. In a client server database, the data is also stored in a file; however, all these operations are mediated through a master program called a server. Microsoft Access is a file server database, whereas Microsoft SQL Server and Oracle are client server databases. Client server databases have several advantages over 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 with file server databases.

### 2.8.1 Microsoft Access Databases

The Microsoft Access database engine is a collection of information stored in a systematic way that forms the underlying component of a database. Also called Jet, (Joint Engine Technology), it allows the manipulation of relational databases. It offers a single interface that other software may use to access Microsoft databases. The supporting software is developed to provide security, integrity, indexing, record locking, and so forth. By executing the Microsoft Access program MSACCESS.EXE, you can see the database engine at work and the user interface

**Table 2.12.** The New Instructor Table

faculty_id	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

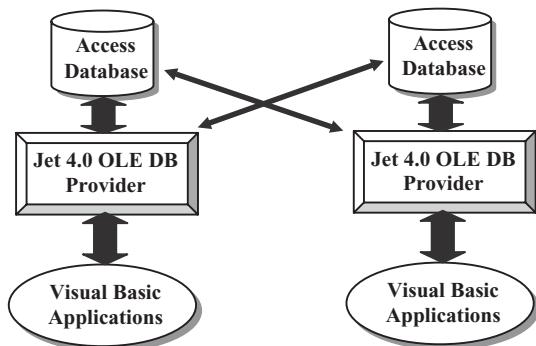


Figure 2.9. Microsoft Access database illustration.

it provides. Figure 2.9 shows how a Visual Basic application accesses the Microsoft Access database via Jet 4.0 OLE database provider.

### 2.8.1.1 Database File

An Access database is made up of a number of components called objects, which are stored in a single file referred to as a *database file*. Access database files have an .mdb (Microsoft DataBase) extension. As new objects are created or more data is added to the database, this file gets bigger. This is a complex file that stores objects like tables, queries, forms, reports, macros, and modules. Some of these objects help the user 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 so 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 graphical user interface (GUI) to create queries. This bypasses writing SQL statements and makes it appealing to beginning and not-so-savvy users, who can use wizards or the 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 the desired information, perform comparisons, or perform calculations and obtain the results. In essence, operators, functions, and expressions are the building blocks for the 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 Web sites 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 the 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, Standard, Workgroup, Developer, and Express. The most common editions are Enterprise, Standard, and Workgroup. It is noteworthy that the database engine is virtually the same in all these editions.

An SQL Server database can be stored on a disk using three types of files: primary data files, secondary data files, and transaction log files. Primary data files are created first and contain user-defined objects, such as tables and views, and system objects. These files have an .mdf extension. If the database grows too big for a disk, it can be stored as secondary files with an .ndf extension. The SQL Server still treats these files as if they were 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 Visual Basic 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 including tables, keys, views, stored procedures, and others. All these objects are necessary for the efficient operation of the database.

### **2.8.2.2 Tables**

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

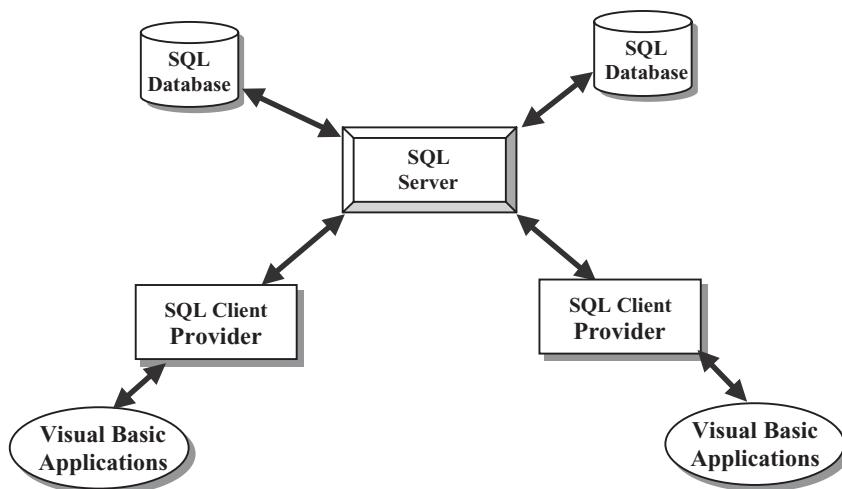


Figure 2.10. SQL Server database structure.

### 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. Views can provide database security. Sensitive information in a database can be concealed by including nonsensitive information in a view and providing user access to the view instead of to all tables in a database. Views can also hide the complexities of a database. A user can use 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 as against 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. These advantages have to do with network performance. Usually users use the same queries over and over again; frequently, different users try 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. Therefore no two records can have the same primary key. A primary a globally unique identifier. Moreover, a primary key may not have a 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 column can be a primary key, or columns can be combined to create 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 stored in different tables from corruption.

### **2.8.2.6 Indexes**

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.

A *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. One can go directly to a specific alphabet letter and from there search sequentially for the specific record. The clustered index is like an inverted tree. The index structure is called a B-tree (binary tree). You start with the root page at the top and find the location of other pages further down at the secondary level, continuing to the tertiary level and so on until you find the desired record. The very bottom pages are the leaf pages and contain the actual data. There can be only one clustered index per table because clustered indexes physically rearrange the data.

Nonclustered indexes do not physically rearrange the data. Like clustered indexes, they 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.

Indexes can be unique, that is, allow duplicate keys, or not unique, not allow duplicate keys. Any column can be used to generate an index. Usually the primary and 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. Then it applies the transaction and creates another entry to generate an after picture of the data in the table. This keeps track of all the operations performed on the 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. An Oracle database contains more files than an 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 a large number of transactions in a multiuser environment. The Standard edition is also used for high-level multiuser installations. It lacks some of the utilities available in the Enterprise edition. The Personal edition is used in a single-user environment for developing database applications. The database engine components are virtually the same for all three editions.

Oracle architecture comprises several components, including the Oracle server, Oracle instance, and 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. The 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 table spaces, 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. The Oracle server uses other files that are not part of the database. These include the *parameter file*, which defines the characteristics of an Oracle instance; the *password file*, used for authentication; and *archived redo log files*, which are copies of the redo log files necessary for recovery from failure.

#### 2.8.3.2 Tables

Users can store data in a regular table, a partitioned table, an index-organized table, or a 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 an 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* is a group of tables that 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.

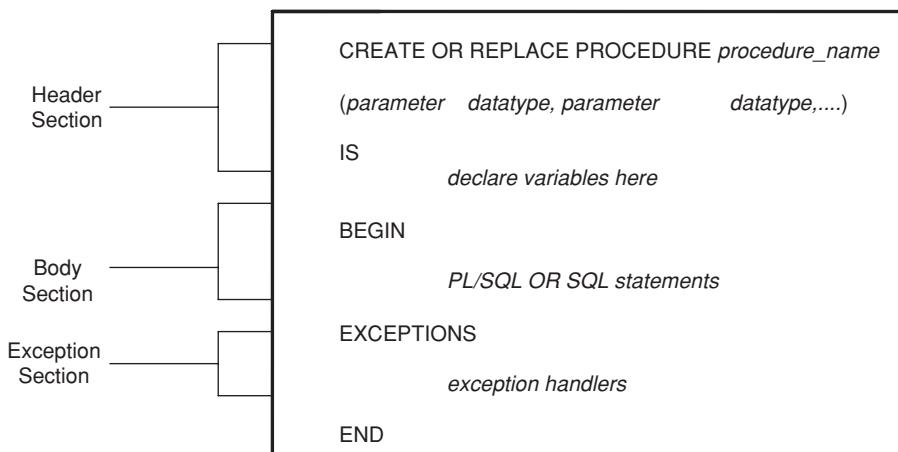
### **2.8.3.3 Views**

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

### **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 SQL Server. By storing procedures on the server, individual SQL statements do not have to be transmitted over the network, thus reducing 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 users from writing 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. A stored procedure 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 the keyword BEGIN and consists of SQL statements. The exception section of the procedure begins with the



**Figure 2.11.** Syntax for creating a stored procedure in Oracle.

keyword EXCEPTION and contains exception handlers that are designed to handle the occurrence of conditions that change the normal flow of execution.

### 2.8.3.5 Indexes

Indexes are created to provide direct access to rows. An index is a tree structure. Indexes can be classified based on their logical design or their physical implementation. Logical classification is based on an 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 spread an index over multiple table spaces, thus decreasing contention for index lookup and increasing manageability. An index may consist of a single column or multiple columns; it may be unique or nonunique. Some types of indexes are outlined below.

**Function-based indexes** precompute the value of a function or an expression of one or more columns and store it in an index. The index can be created as a B-tree or as a bitmap. 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 application. Single-column indexes can be built on text, spatial, scalar, object, or LOB data types.

**B-tree indexes** store a list of row IDs for each key. The structure of a B-tree index is similar to the ones in the SQL Server described above. The leaf nodes contain indexes that point to rows in a table. The leaf blocks allow scanning of the index in either ascending or descending order. The 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, such as Y/N for marital status or M/F for gender. A bitmap is organized like a B-tree, where the leaf nodes store a bitmap instead of row IDs. When changes are made to the key columns, bitmaps must be modified.

### 2.8.3.6 Initialization Parameter Files

The Oracle server must read the initialization parameter file before starting an Oracle database instance. There are two types of initialization parameter files: static parameter files and persistent parameter files. An initialization parameter file contains a list of instance parameters, 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, the 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 the database name and identifier, a time stamp of database creation, tablespace names, names and location of data files and redo log files, current log file sequence numbers, 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, 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 on whether the database is administered locally or remotely, one can choose either operating system or password file authentication to authenticate database administrators. Oracle provides a password utility to create 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.mdb. As we mentioned in the previous sections, Access is a file-based database system, which means that the database is composed of a set of data tables that are represented in the form of files.

Open Microsoft Access and select New from the File menu item to open a new Access database, and name this database CSE\_DEPT.mdb by entering this name into the box for the file name. Then click the Create button to create this new database.

### **2.9.1 Create the LogIn Table**

After the new database is created, select and double-click the default item – Create table in the Design view – to open the Design view window, which is shown in Figure 2.12.

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 in the first Field Name box. The data type for user\_name should be Text, so click the drop-down arrow of the Data Type box and select Text. You can enter comments in the Description box to indicate the purpose of this data. In this case, just enter Primary key for the LogIn table, since this column will be the primary key for this table.

In a similar way, enter 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

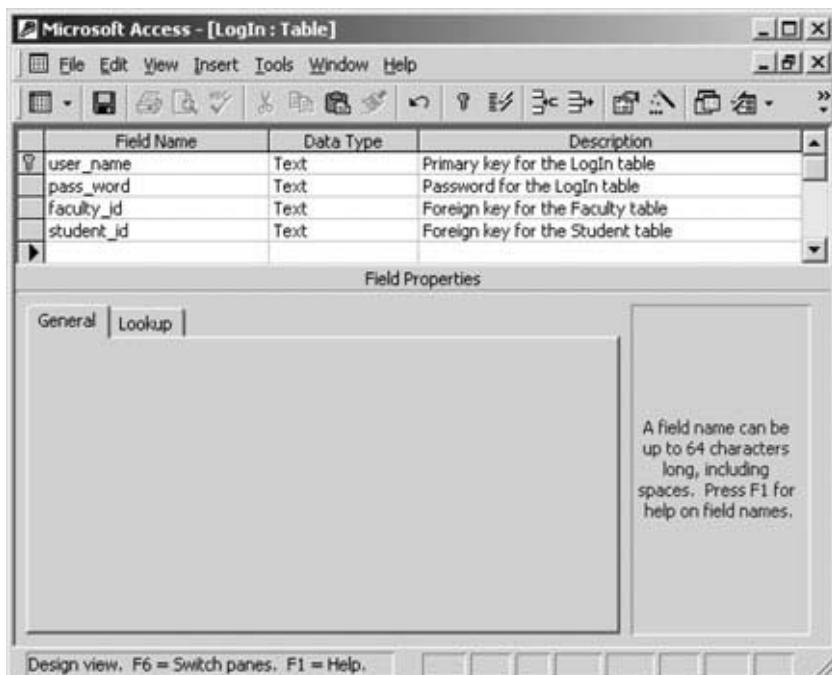


Figure 2.12. The Design view of the new table.

need to assign the user\_name column as the primary key for this table. Click and select the first row, user\_name, from the table and then go to the Toolbar and select the Primary Key tool, which looks like a key. Immediately a primary key symbol is added to the user\_name row, which is shown in Figure 2.12.

Go to the File menu and select Save to save this table. Enter the table name LogIn into the Table Name box, then click the OK button.

The finished Design view of the LogIn table is shown in Figure 2.12.

Next, you need to insert data into this LogIn table. To do that, you need to open the DataSheet view of the table. You can open this view either by clicking the drop-down arrow next to the View tool, which is the first tool on the Toolbar, and selecting DataSheet view or by going to the View menu and choosing DataSheet View.

Four data columns, user\_name, pass\_word, faculty\_id, and student\_id, are displayed when the DataSheet view of this LogIn table is opened. Enter the data shown in Table 2.13 into this table. The finished LogIn table is shown in Figure 2.13.

Your finished LogIn table should match the one in Figure 2.13. Go to the File menu and select Save to save this table. Then click the Close button that is located on the upper right corner of the table to close this table.

## 2.9.2 Create the Faculty Table

Now let's create the second table, Faculty. Again, select and double-click the default item – Create table in the Design view – from the Access database main window to

**Table 2.13.** The Data in the LogIn Table

<b>user_name</b>	<b>pass_word</b>	<b>faculty_id</b>	<b>student_id</b>
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	

open the design view for this new table. This table will have seven columns: faculty\_id, 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 the table. The default length for each text string is 50.

Now you need to assign the primary key for this table. As in the LogIn table, the first column, in this case faculty\_id, will be the primary key. Click and select the row faculty\_id and then go to the Toolbar and select the Primary key tool. Save this table as Faculty, as is shown in Figure 2.14.

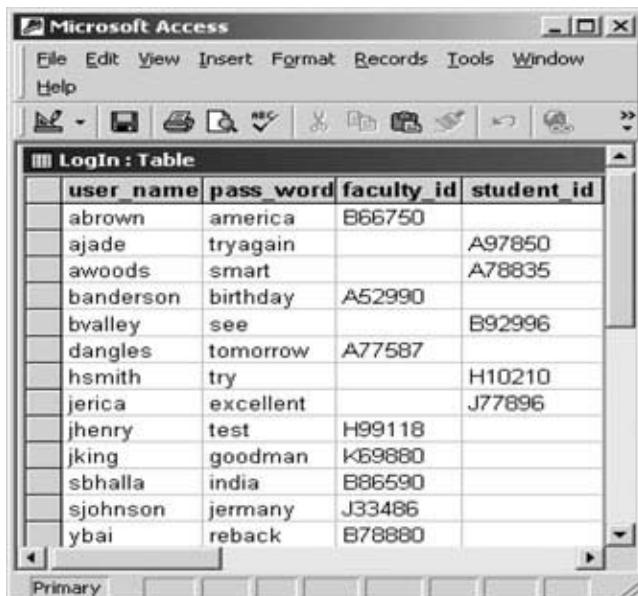


Figure 2.13. The completed LogIn table.

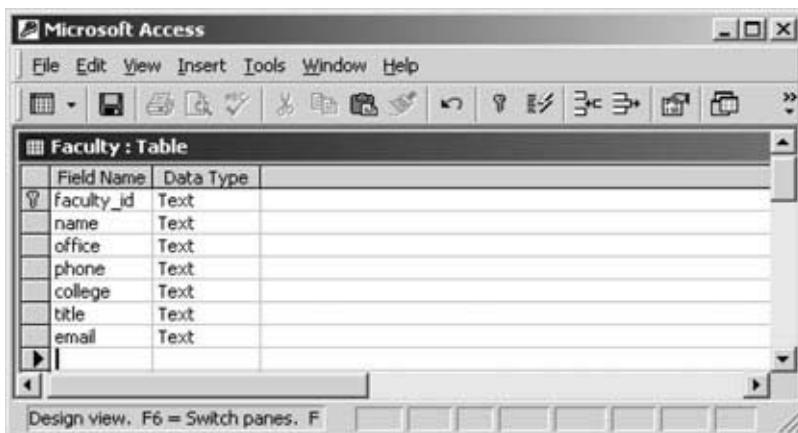


Figure 2.14. The Design view of the Faculty table.

Now open the DataSheet view of the Faculty table and enter the data that is shown in Table 2.14 into this open 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 keys for the Course, Student, and StudentCourse tables, respectively (refer to Tables 2.15, 2.16, and 2.17). For the data type selections, follow the directions given below.

The data type selections for the Course table are:

- course\_id – Text
- credit – Number
- enrollment – Number
- All other columns – Text

The data type selections for the Student table are:

- student\_id – Text
- credits – Number
- All other columns – Text

The data type selections for the StudentCourse table are:

- s\_course\_id – Number
- credit – Number
- All other columns – Text

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

The finished Course, Student, and StudentCourse tables are shown in Figures 2.16, 2.17, and 2.18.

**Table 2.14.** The Data in the Faculty Table

faculty_id	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	Assistant Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jkking@college.edu

The screenshot shows a Microsoft Access window titled "Faculty : Table". The table has eight columns: faculty\_id, name, office, phone, college, title, and email. The data is as follows:

faculty_id	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	Assistant Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

Figure 2.15. The completed Faculty table.

#### 2.9.4 Create Relationships Among Tables

Now that all five tables are completed, we need to set up the relationships between these tables by using the primary and foreign keys. Go to the Tools|Relationships menu item to open the Show Table dialog box. Select all five tables and click the Add button, then click the Close button to close this dialog box. All five tables are displayed in the Relationships dialog box. The relationships we want to add are shown in Figure 2.19.

P.K and F.K. in Figure 2.19 designate the primary and foreign keys, respectively. For example, faculty\_id in the Faculty table is a primary key and can be connected with faculty\_id in the LogIn table, which is a foreign key. The relationship between these two tables is one-to-many since the unique primary key faculty\_id in the Faculty table can be connected to multiple foreign keys, that is, faculty\_id located in the LogIn table.

To set this relationship between these two tables, click an faculty\_id from the Faculty table and drag to 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 check box and click the Create button to create this relationship. In a similar way, you can create all other relationships between these five tables.

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

A completed Microsoft Access database file, CSE\_DEPT.mdb, can be found the folder **database\Access** that is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). Refer to Appendix F if you want to use this sample database in your Visual Basic applications.

### 2.10 CREATE MICROSOFT SQL SERVER 2005 SAMPLE DATABASE

After you have finished the installation of SQL Server Management Studio Express (refer to Appendix B), you can use it to connect to the server and build your database. To start, go to Start>All Programs|Microsoft SQL Server 2005 and select

**Table 2.15. The Data in the Course Table**

<b>course_id</b>	<b>course</b>	<b>credit</b>	<b>classroom</b>	<b>schedule</b>	<b>enrollment</b>	<b>faculty_id</b>
CSC-131A	Computers in Society	3	TC-109	MWF: 9:00-9:55 AM	28	A52990
CSC-131B	Computers in Society	3	TC-114	MWF: 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	MWF: 9:00-9:55 AM	30	B86590
CSC-131E	Computers in Society	3	TC-301	MWF: 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	MWF: 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	MWF: 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	MWF: 9:00-9:55 AM	18	H99118
CSC-233B	Introduction to Algorithms	3	TC-302	MWF: 11:00-11:55 AM	19	K69880
CSC-234A	Data Structure & Algorithms	3	TC-302	MWF: 9:00-9:55 AM	15	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	MWF: 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	MWF: 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	MWF: 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	MWF: 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	MWF: 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	MWF: 1:00-1:55 PM	26	B78880
CSE-436	Automatic Control and Design	3	TC-305	MWF: 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	Adv Logic & Microprocessor	3	TC-213	MWF: 11:00-11:55 AM	35	B78880
CSE-439	Special Topics in CSE	3	TC-206	MWF: 10:00-10:55 AM	22	J33486

**Table 2.16.** The Data in the Student Table

<b>student_id</b>	<b>name</b>	<b>gpa</b>	<b>credits</b>	<b>major</b>	<b>schoolYear</b>	<b>email</b>
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

<b>s_course_id</b>	<b>student_id</b>	<b>course_id</b>	<b>credit</b>	<b>major</b>
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	CSIS
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 screenshot shows the Microsoft Access application window with the title bar "Microsoft Access". Below the title bar is the menu bar with options: File, Edit, View, Insert, Format, Records, Tools, Window, Help. The main area displays a table named "Course : Table". The table has columns: course\_id, course, credit, classroom, schedule, enrolin, and faculty\_id. The data in the table consists of 36 rows, each representing a course with its details like name, credit hours, room number, and scheduling information. The last row is currently selected. At the bottom of the table, there are navigation buttons for records (Record: 1 of 36) and a "Datasheet View" button.

course_id	course	credit	classroom	schedule	enrolin	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	B66690
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	B66690
CSC-439	Database Systems	3	TC-206	M-W-F: 1:00-1:55 PM	18	B66690
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	Fundations 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	B66690
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

Figure 2.16. The completed Course table.

SQL Server Management Studio Express. A connection dialog is opened, as shown in Figure 2.22.

Your computer name followed by your server name should be displayed in the Server name box. In this case, it is SUSAN\SQLEXPRESS. 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

The screenshot shows the Microsoft Access application window with the title bar "Microsoft Access". Below the title bar is the menu bar with options: File, Edit, View, Insert, Format, Records, Tools, Window, Help. The main area displays a table named "Student : Table". The table has columns: student\_id, name, gpa, credits, major, schoolYear, and email. The data in the table consists of 6 rows, each representing a student with their name, GPA, credits, major, year, and email address. The last row is currently selected. At the bottom of the table, there are navigation buttons for records (Record: 1 of 6) and a "Datasheet View" button.

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

Figure 2.17. The completed Student table.

The screenshot shows the Microsoft Access application window with the title bar "Microsoft Access". Below the title bar is a menu bar with options: File, Edit, View, Insert, Format, Records, Tools, Window, Help. Under the "File" menu, there are icons for Open, Save, Print, and Exit. The main area displays a table named "StudentCourse : Table". The table has columns: s\_course\_id, student\_id, course\_id, credit, and major. The data in the table consists of 28 rows, each representing a student's enrollment in a course. The "student\_id" column contains values like H10210, B92996, J77896, etc. The "course\_id" column contains values like CSC-131D, CSC-132A, CSC-335, etc. The "credit" column shows values such as 3 CE, 3 CS/IS, 3 CS/IS, etc. The "major" column shows values like CSC-234B, CSC-234A, CSC-233A, etc. At the bottom of the table, there is a status bar with "Datasheet View" and "CAPS".

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	CSE-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 2.18. The completed StudentCourse table.

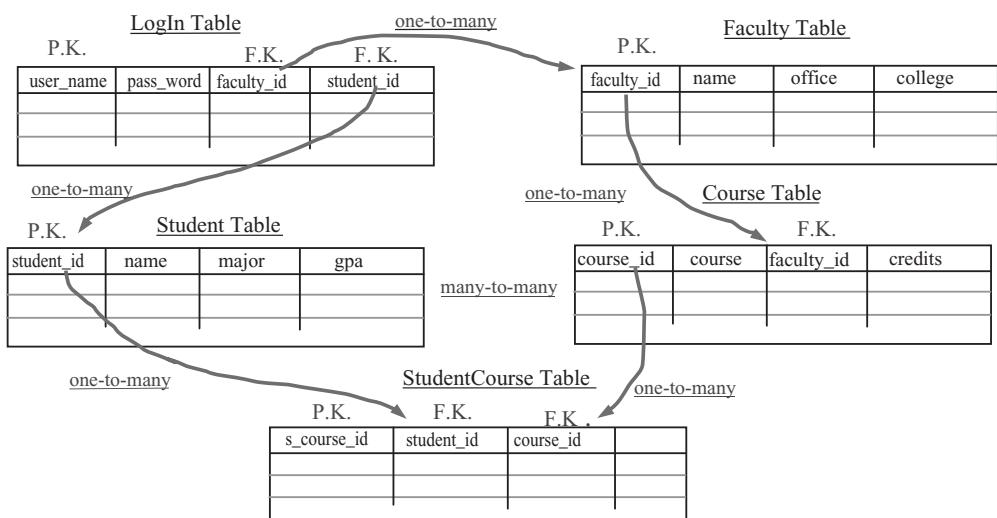


Figure 2.19. Relationships between tables.

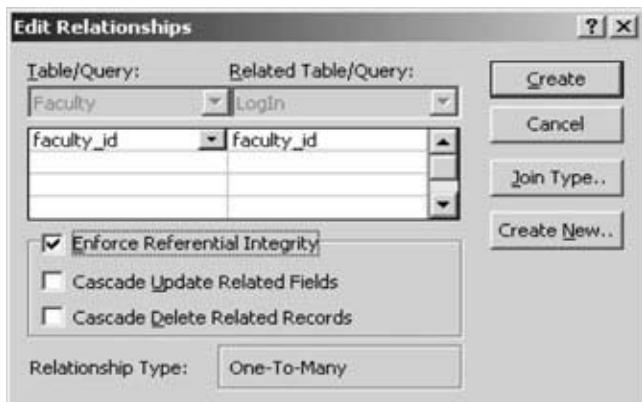


Figure 2.20. The Edit Relationships dialog box.

when you registered your computer. Click the Connect button to connect your client to your server.

The server management studio opens when this connection is completed, which is shown in Figure 2.23.

To create a new database, right-click the Databases folder from the Object Explorer window, and select the New Database item from the popup menu. Enter CSE\_DEPT into the Database name box in the New Database dialog box; keep

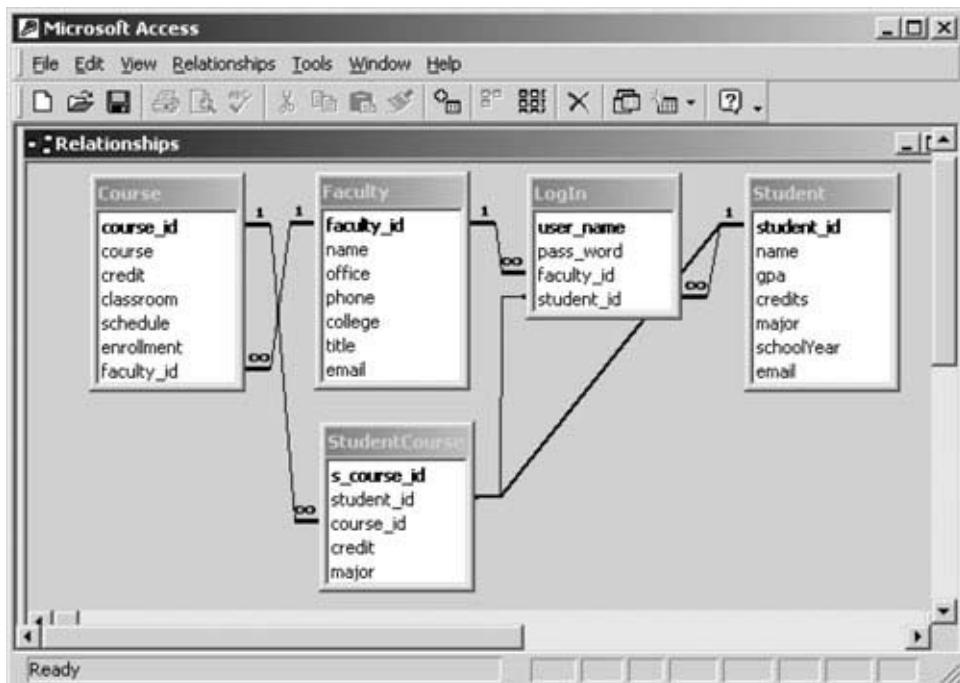


Figure 2.21. The completed relationships for tables.



Figure 2.22. Connect to the SQL Server 2005.

all other settings unchanged. Then click the OK button. You will find that a new database named CSE\_DEPT has been created and is located under the Databases folder in the Object Explorer window.

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



Figure 2.23. The open server management studio.

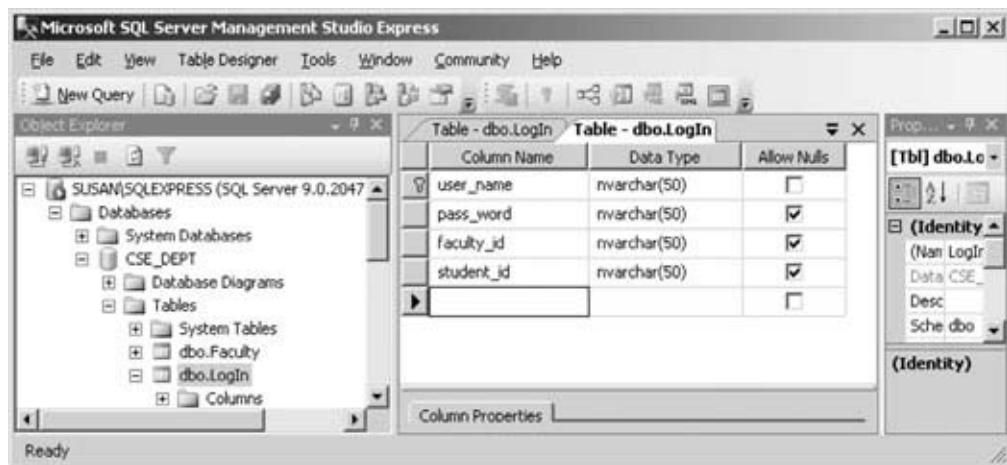


Figure 2.24. The new table window.

### 2.10.1 Create the LogIn Table

A default data table named dbo.Table\_1 is created. Three columns are displayed in this new table: Column Name, Data Type, and Allow Nulls, which allow you to enter the name, the data type, and, check-mark for each column. You can check the box if you want to allow that column to be empty. Do not check it if that column must contain valid data. Generally, for the column that has been selected to serve as the primary key, you should not check the box associated with that column.

The first table will be the LogIn table, which has four columns with the following column names: user\_name, pass\_word, faculty\_id, and student\_id. Enter these four names into the Column Names column. 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 these data types into Data Type column. The first column the user\_name is selected as the primary key, so leave the check box blank for that column and check the other three check boxes.

To designate the first column, user\_name, as the primary key, click the first row and then go to the Toolbar and select the Primary Key tool (displayed as a key). In this way, a primary key symbol is displayed to the left of this row, as shown in Figure 2.24.

Before we can finish this table, we first need to save and name it. Go to File|Save Table\_1 and enter LogIn as the name for this new table. Click the OK button to finish 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 to this LogIn table, right-click this table and select the Open Table item from the popup menu. Enter the data shown in Table 2.18 into this table. Your finished LogIn table should match the one shown in Figure 2.25.

One point to be noted is that you must place a NULL in any field that has no value in this LogIn table, since it is different for blank fields between the Microsoft

**Table 2.18.** The Data in the LogIn Table

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

Access and the SQL Server database. Go to File|Save All to save this table. Now let's create the second table, Faculty.

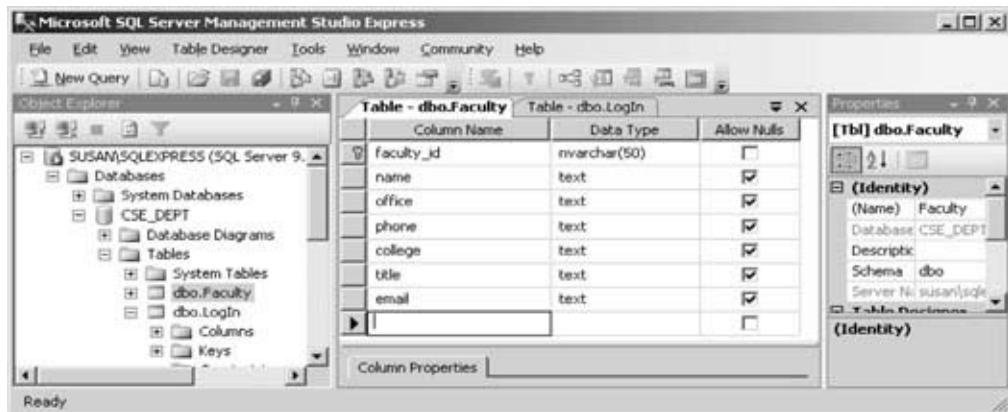
### 2.10.2 Create the Faculty Table

Right-click 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, name, office, phone, college, title, and email. The data type of the faculty\_id is nvarchar(50), and all other data

The screenshot shows the Microsoft SQL Server Management Studio Express interface. On the left, the Object Explorer pane displays the database structure, including the CSE\_DEPT database and its tables (Faculty, LogIn). The Faculty table is currently selected. In the center, the Table - dbo.LogIn tab is active, showing the data from Table 2.18. The Properties pane on the right shows the table's properties, including the primary key (Identity) and various column properties like Name, Type, and Length. The status bar at the bottom indicates 'Ready'.

Figure 2.25. The finished LogIn table.



**Figure 2.26.** The design view of the Faculty table.

types are text, since all of them are string variables. The reason we selected nvarchar(50) as the data type for faculty\_id is that a primary key can be this data type but cannot be 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 that row and then go to the Toolbar and select the Primary Key tool. In this way, 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 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 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 the Open Table item to open this table. Enter the data shown in Table 2.19 into this table.

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

Now go to the File menu 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 in the Tables folder under the database CSE\_DEPT in the Object Explorer window.

### 2.10.3 Create Other Tables

In a similar way, you need to create the remaining 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, 2.21, and 2.22). For the data type selections, follow the directions below:

The data type selections for the Course table are:

- course\_id – nvarchar(50) (Primary key)
- credit – float
- enrollment – int
- faculty\_id – nvarchar(50)
- All other columns – either nvarchar(50) or text

**Table 2.19.** The Data in the Faculty Table

faculty_id	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	Assistant Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@college.edu
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
K699880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

The screenshot shows the Microsoft SQL Server Management Studio Express interface. On the left, the Object Explorer pane displays the database structure under 'SUSANSQLEXPRESS (SQL Server)'. It shows databases like 'System Databases' and 'CSE\_DEPT', and tables such as 'dbo.Faculty', 'dbo.Login', and 'dbo.StudentCourse'. The central area is a grid titled 'Table - dbo.Faculty' showing data for faculty members. The columns are 'faculty\_id', 'name', 'office', 'phone', 'college', 'title', and 'email'. The data includes entries for Black Anderson, Debby Angles, Alice Brown, Ying Bai, Salish Bhalla, Jeff Henry, Steve Johnson, and Jenney King. The bottom of the window shows navigation buttons and a status bar indicating 'Ready'.

faculty_id	name	office	phone	college	title	email
A62990	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
B78000	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Assistant Professor	ybai@college.edu
B86590	Salish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sballa@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
K98880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 2.27. The completed Faculty table.

The data type selections for the Student table are:

- student\_id – nvarchar(50) (Primary key)
- gpa – float
- credits – int
- All other columns – either nvarchar(50) or text

The data type selections for the StudentCourse table are:

- 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 shown in Tables 2.20, 2.21, and 2.22 into each associated table, and save the tables as Course, Student, and StudentCourse, respectively. The finished tables should match the ones shown in Figures 2.28, 2.29, and 2.30.

One point to note is that you can copy the content of each table from the Microsoft Access database file to the associated data table in the Microsoft SQL Server environment if you have already developed the Microsoft Access database.

To copy the data, you must first select a whole blank row from your destination table, the table in the Microsoft SQL Server database, and then select all data rows from your source table, the Microsoft Access database file, by highlighting them and choosing Copy from the Edit menu. Next, you need to paste those rows by clicking that blank row in the Microsoft SQL Server database and then choosing Paste from the Edit menu. An error message may be displayed, as shown in Figure 2.31. Just click the OK button and your data will be pasted to your destination table without a problem. The reason for that error message is that the primary

**Table 2.20. The Data in the Course Table**

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

**Table 2.21.** The Data in the Student Table

<b>student_id</b>	<b>name</b>	<b>gpa</b>	<b>credits</b>	<b>Major</b>	<b>schoolYear</b>	<b>email</b>
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

<b>s_course_id</b>	<b>student_id</b>	<b>course_id</b>	<b>credit</b>	<b>major</b>
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 screenshot shows the Microsoft SQL Server Management Studio Express interface. On the left is the Object Explorer pane, which lists the database structure for 'SUSAN\SQLEXPRESS'. In the center is a grid view of the 'Course' table data. On the right is a query editor window titled '(Identity)' containing a script for creating a table named 'CSE\_DEPT'.

		Table - dbo.Course		Table - dbo.StudentCourse		Table - dbo.Student		Table - dbo.Course	
		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	B66750	
		CSC-131E	Computers in Society	3	TC-301	M-W-F: 1:00-1:55 PM	25	B66750	
		CSC-131H	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-200	Algorithms & Structures	3	TC-301	M-W-F: 1:00-1:55 PM	20	A77587	
		CSC-232A	Programming I	3	TC-303	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-303	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-303	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-303	T-H: 11:00-12:25 PM	15	A77587	
		CSC-305	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	B66590	
		CSC-439	Database Systems	3	TC-206	M-W-F: 1:00-1:55 PM	18	B66590	
		CSE-130A	Introduction to CSE	3	TC-301	T-H: 1:00-2:25 PM	15	A52990	
		CSE-130B	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	

Figure 2.28. The completed Course table.

key cannot be a NULL value. Before you have finished this paste operation, the table cannot identify whether you have non-null values in the source row that will be pasted in this column.

#### 2.10.4 Create Relationships Among Tables

Next, we need to set up relationships among these five tables using the primary and foreign keys. In the Microsoft SQL Server 2005 Express database environment, the

The screenshot shows the Microsoft SQL Server Management Studio Express interface. On the left is the Object Explorer pane, which lists the database structure for 'SUSAN\SQLEXPRESS'. In the center is a grid view of the 'Student' table data. On the right is a query editor window titled '(Identity)' containing a script for creating a table named 'CSE\_DEPT'.

		Table - dbo.Student		Table - dbo.Course		Table - dbo.StudentCourse		Table - dbo.Student	
		student_id	name	gpa	credits	major	schoolYear	email	
		A78035	Andrew Woods	3.24	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	
		NULL	NULL	NULL	NULL	NULL	NULL	NULL	

Figure 2.29. The completed Student table.

The screenshot shows the Microsoft SQL Server Management Studio Express interface. The left pane displays the Object Explorer with the database 'SUSAN\SQLEXP...StudentCourse' selected. The right pane shows a table named 'StudentCourse' with the following data:

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-430	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

Figure 2.30. The completed StudentCourse table.

relationship between tables can be set by using the Keys folder under each data table in the Object Explorer window. Now let's begin by setting up the relationship between the LogIn and Faculty tables.

#### 2.10.4.1 Create Relationship Between the LogIn and Faculty Tables

The relationship between the Faculty and LogIn tables is one-to-many, which implies that one occurrence of faculty\_id (primary key) in the Faculty table may



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

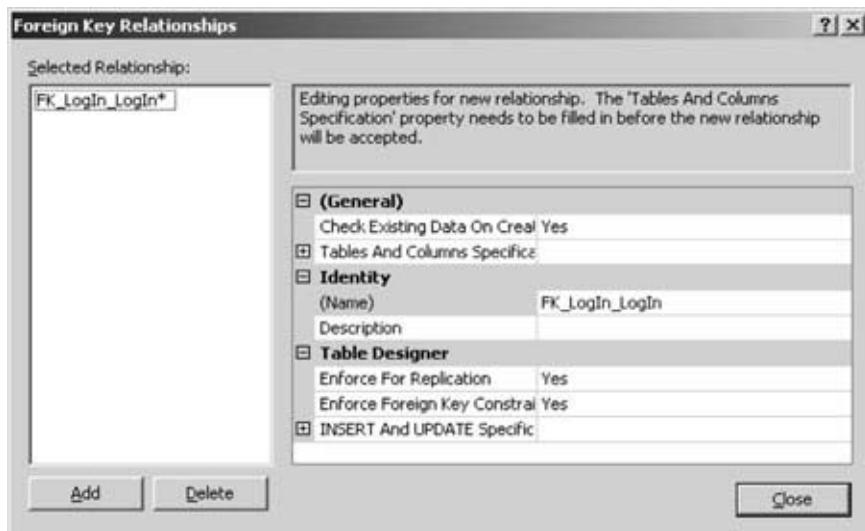


Figure 2.32. The opened Foreign Key Relationships dialog box.

be mapped to many occurrences of faculty\_id (foreign key) 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 the Keys folder and select the New Foreign Key item from the popup menu to open the Foreign Key Relationships dialog box, 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 Faculty tables, so change the name of this foreign relationship to FK\_LogIn\_Faculty by modifying its name in the Name box under the Identity pane, and then press Enter on your keyboard. Then select two tables by clicking on the Tables And Columns Specification item under the General pane. Click the expansion button that is located to the left of the Tables And Columns Specification item to open the Tables and Columns dialog, which is shown in Figure 2.33.

Click the drop-down arrow on 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 box should match the one shown in Figure 2.34.

Click OK to close this dialog box.

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.

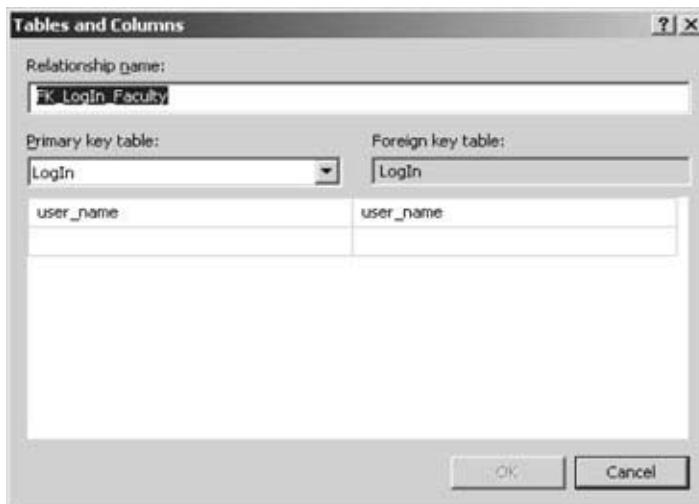


Figure 2.33. The opened Tables and Columns dialog box.

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

#### **2.10.4.2 Create Relationship Between the LogIn and Student Tables**

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

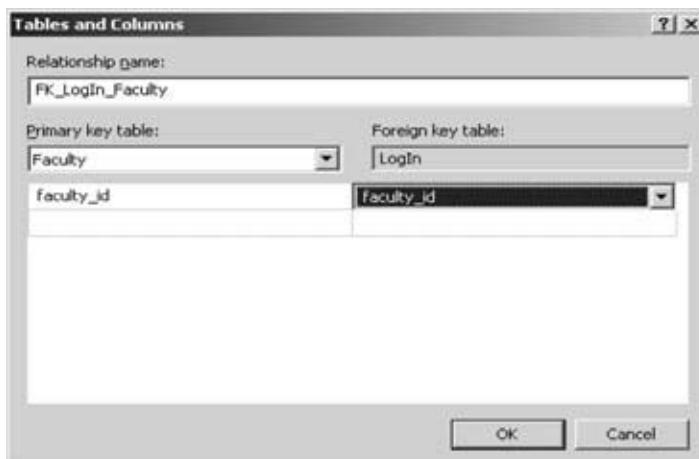


Figure 2.34. The finished Tables and Columns dialog box.



Figure 2.35. The completed Tables and Columns dialog box.

Right-click the Keys folder that is under the dbo.LogIn table and select the New Foreign Key item from the popup menu to open the Foreign Key Relationships dialog. Change the name to FK\_LogIn\_Student and press the Enter key on your keyboard. Go to the Tables And Columns Specification item to open the Tables and Columns dialog box, then select the Student table from the Primary key table combo box and student\_id from the box under the Primary key table combo box. Select student.id from the box under the Foreign key table combo box. Your finished Tables and Columns dialog box should match the one shown in Figure 2.35.

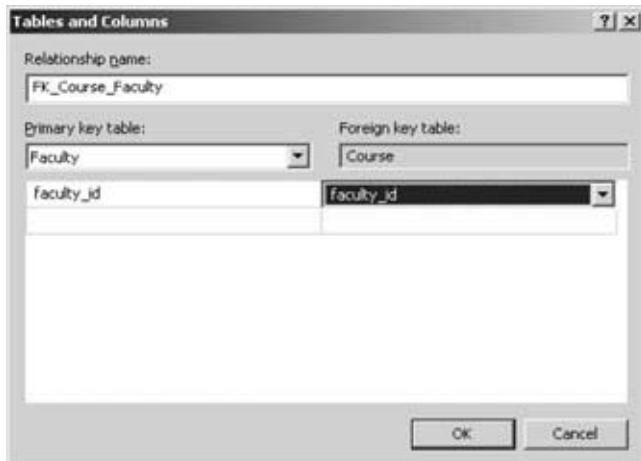
Click OK to close this dialog box, and click the Close button to close the Foreign Key Relationships dialog box. Go to the File|Save LogIn menu item to save this relationship. Click Yes in the following dialog box to finish this saving.

Now right-click the Keys folder under the dbo.LogIn table and select the Refresh item to show the newly created foreign key FK\_LogIn\_Student.

#### 2.10.4.3 Create Relationship Between the Faculty and Course Tables

The relationship between the Faculty and Course tables is one-to-many. 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 the Keys folder under the dbo.Course table in the Object Explorer window and select the New Foreign Key item from the popup menu. In the Foreign Key Relationships dialog box, change the name of this new relationship to FK\_Course\_Faculty in the Name box and press the Enter key on your keyboard. In the open Tables and Columns dialog box, select the Faculty table from the Primary key table combo box and select faculty\_id from the box that is just below the Primary key table combo box. Then select faculty\_id from the box that is just below the Foreign key table combo box. Your finished Tables and Columns dialog box should match the one shown in Figure 2.36.



**Figure 2.36.** The finished Tables and Columns dialog box.

Click OK to close this dialog box and click the Close button to close the Foreign Key Relationships dialog box. Then go to the File|Save Course menu item and click Yes in the following dialog box to save this setting.

Now right-click the Keys folder under the dbo.Course table, and select the Refresh item. Immediately you will see our newly created relationship key, FK\_Course\_Faculty.

#### **2.10.4.4 Create Relationship Between the Student and StudentCourse Tables**

The relationship between the Student and StudentCourse tables is one-to-many. 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 the Keys folder under the dbo.StudentCourse table in the Object Explorer window and select the New Foreign Key item from the popup menu. In the Foreign Key Relationships dialog box, change the name of this new relationship to FK\_StudentCourse\_Student in the Name box and press the Enter on your keyboard. In the Tables and Columns dialog box, select the Student table from the Primary key table combo box and select student.id from the box that is just below the Primary key table combo box. Then select student.id from the box that is just below the Foreign key table combo box. The finished Tables and Columns dialog box should match the one shown in Figure 2.37.

Click OK to close this dialog box, and click the Close button to close the Foreign Key Relationships dialog box. Then go to the File|Save StudentCourse menu item and click Yes in the following dialog box to save this relationship.

Now right-click the Keys folder under the dbo.StudentCourse table and select the Refresh item. Immediately you will see our newly created relationship key, FK\_StudentCourse\_Student.

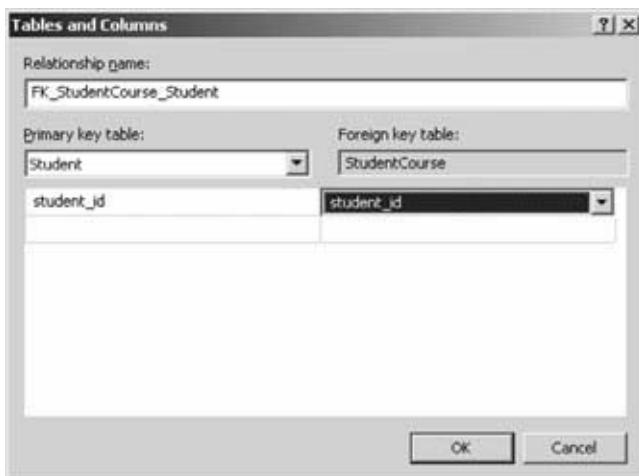


Figure 2.37. The finished Tables and Columns dialog.

#### 2.10.4.5 Create Relationship Between the Course and StudentCourse Tables

The relationship between the Course and StudentCourse tables is one-to-many. 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 the Keys folder under the dbo.StudentCourse table in the Object Explorer window and select the New Foreign Key item from the popup menu. In the Foreign Key Relationships dialog box, change the name of this new relationship to FK\_StudentCourse\_Course in the Name box, and press the Enter key on your keyboard. In the Tables and Columns dialog box, select the Course table from the Primary key table combo box and select course\_id from the box that is just below the Primary key table combo box. Then select course\_id from the box that is just below the Foreign key table combo box. Your finished Tables and Columns dialog box should match the one shown in Figure 2.38.

Click OK to close this dialog box and click the Close button to close the Foreign Key Relationships dialog box. Then go to the File|Save StudentCourse menu item and click Yes in the following dialog box to save this relationship.

Now right-click the Keys folder under the dbo.StudentCourse table and select the Refresh item. Immediately you will see our newly created relationship key, FK\_StudentCourse\_Course.

At this point, we have finished setting the relationships among our five data tables. The completed relationships for these tables are shown in Figure 2.39.

A completed Microsoft SQL Server 2005 sample database file, CSE\_DEPT.mdf, can be found in the folder **database\SQLServer** that is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). Refer to Appendix F if you want to use this sample database in your Visual Basic applications.

You use this database file to develop sophisticated implementation programs in the following chapters when the data provider is Microsoft SQL Server.

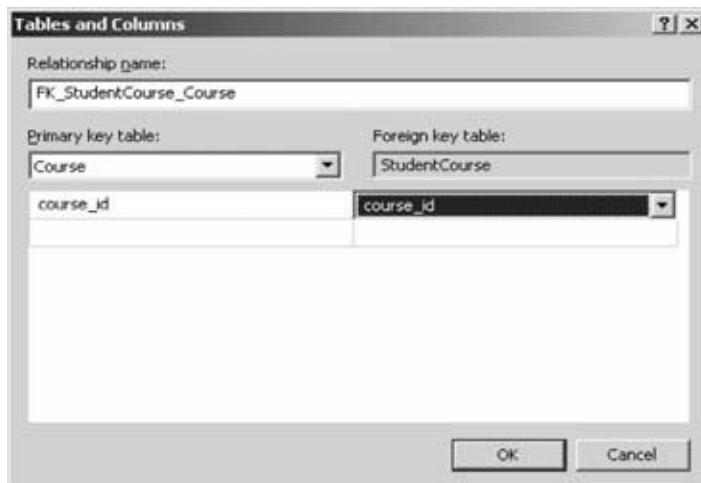


Figure 2.38. The finished Tables and Columns dialog box.

## 2.11 CREATE ORACLE 10G XE SAMPLE DATABASE

After you finish downloading and installing Oracle Database 10g Express Edition (refer to Appendix C), you can begin to connect your client computer to your server to create, access, and manipulate your database. In this section, you will learn how to create, edit, and manipulate your database using Oracle Database 10g Express Edition (XE).

To connect your computer to your Oracle server, go to Start>All Program Files|Oracle Database 10g Express Edition|Go To Database Home Page to open the Login page, which is shown in Figure 2.40.

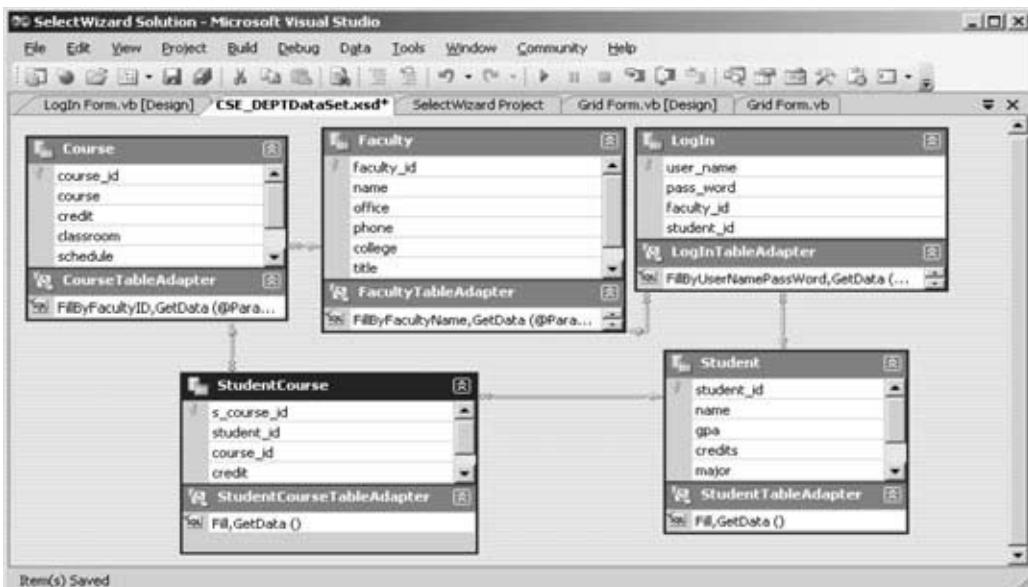


Figure 2.39. Relationships among tables.



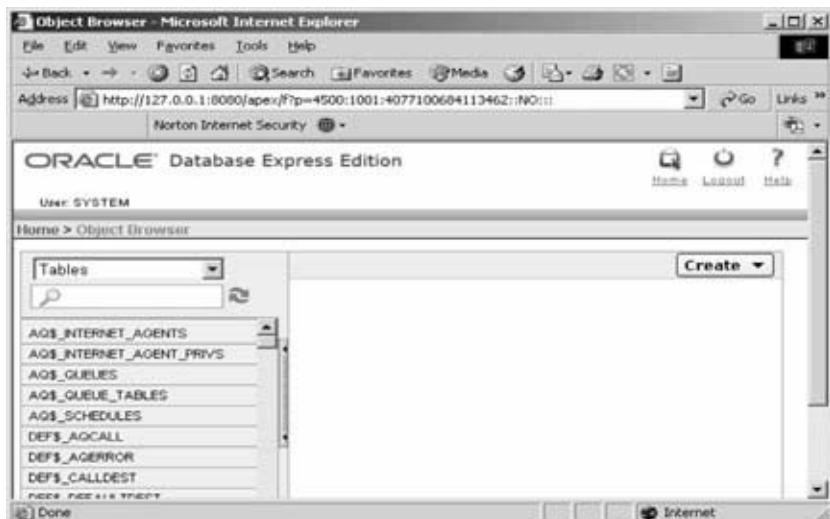
Figure 2.40. The Login page.

You can log on as an administrator by entering SYSTEM into the Username box and the password you selected during your download and installation of Oracle Database 10g XE into the Password box if you want to use the whole system. But if you want to create customer databases, you must create a new user and log on as that user. Refer to Appendix D for detailed information on how to create a new user to create a new database.

The open home page is shown in Figure 2.41.



Figure 2.41. The home page.



**Figure 2.42.** The opened Object Browser window.

You need to use Object Browser to create, access, and manipulate your database in the Oracle Database 10g XE server. Click this icon to open the Object Browser window, which is shown in Figure 2.42.

Some developed data tables are displayed in the left column. These tables are created by the system, and you can use them as sample tables to test your applications. But we want to create a sample database ourselves, so we need to create five new tables: LogIn, Faculty, Course, Student, and StudentCourse.

### **2.11.1 Create the LogIn Table**

Make sure that the content of the text box in the left column is Tables, which is the default value. Click the drop-down arrow on the Create button and select Tables to create our first new table, LogIn, which is shown in Figure 2.43.

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

The first step in the flowchart is Columns, which means that you need to create each column by entering the information for your data table, such as the Column Name, Type, Precision, Scale, and Not Null. 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 (scale 10) since this data type is flexible and can contain varying-length characters. The upper bound of the length is 10, which means that each column can contain up to 10 characters. Since user\_name will be the primary key for this table, check the Not Null check box next to this column to indicate that this column cannot contain a blank value. Your finished first step is shown in Figure 2.44.

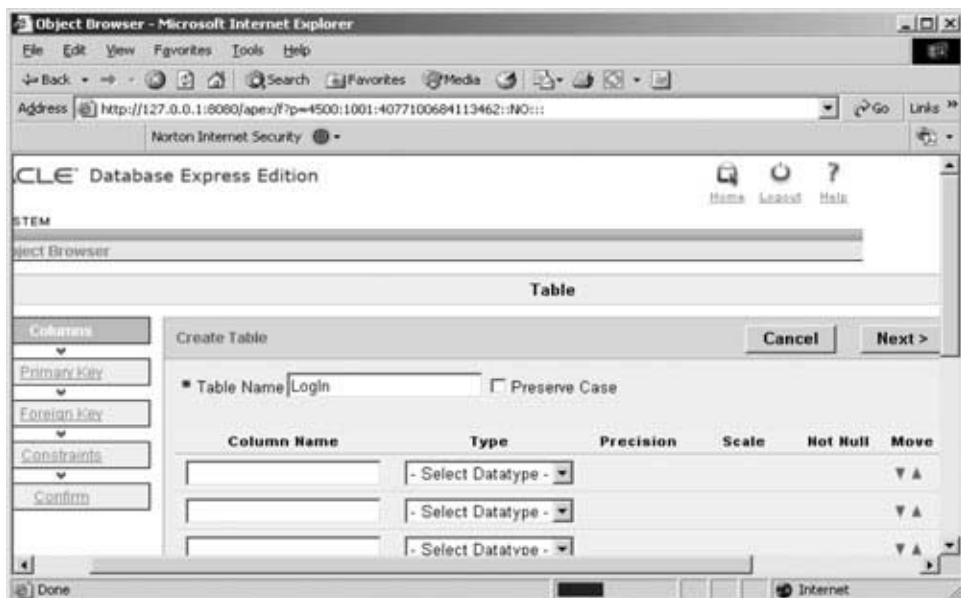


Figure 2.43. Create a new table.

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

To assign a primary key our new LogIn table, select Not Populated for the Primary Key because we don't want to use a Sequence object to assign a sequence to our primary key. The Sequence object in Oracle is used to automatically create a sequence of numeric numbers for the primary key. In our case, the primary key

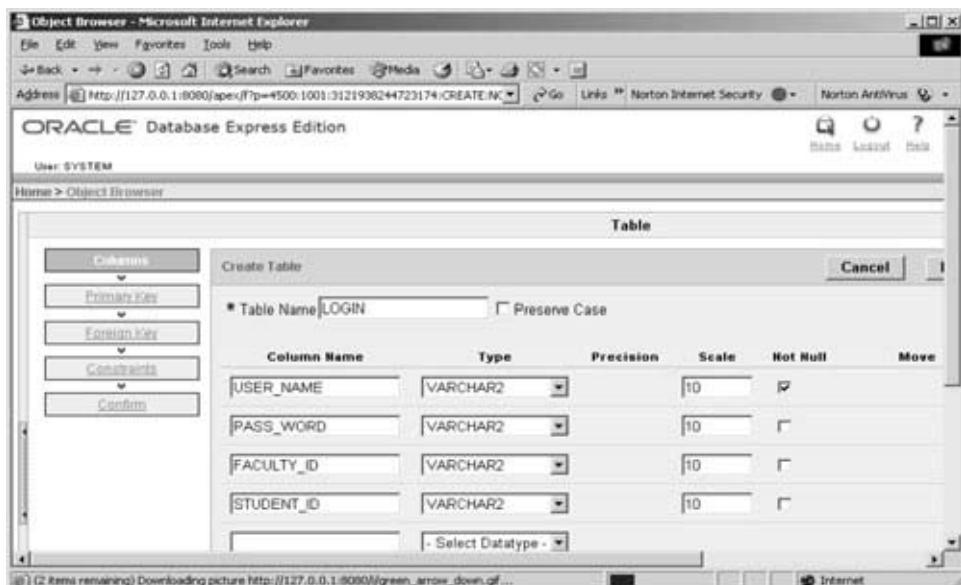


Figure 2.44. The finished first step.

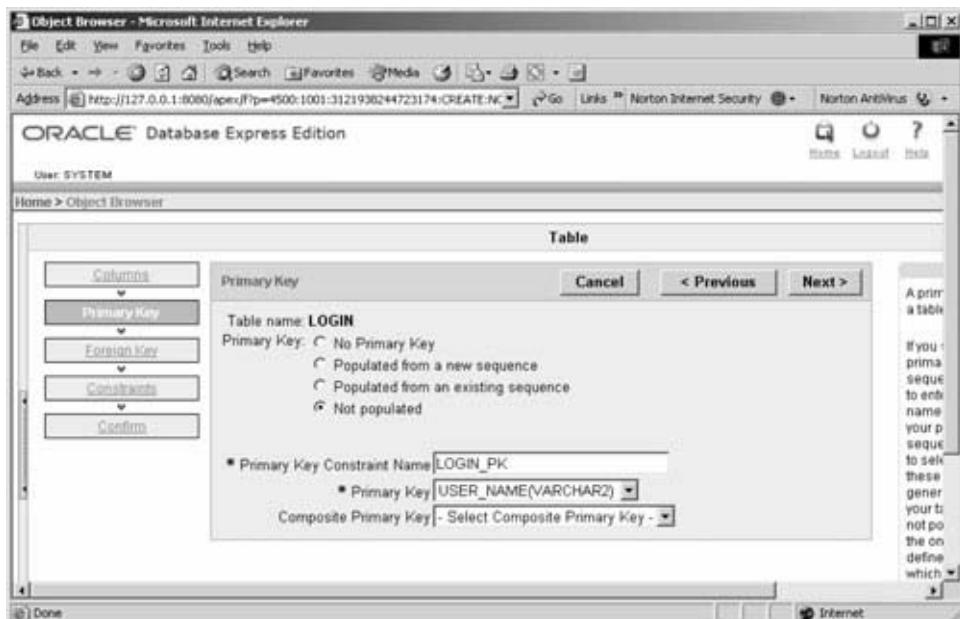


Figure 2.45. The second step – assign the primary key.

is a string; therefore, we cannot use a sequence. Keep the Primary Key Constraint Name, LOGIN\_PK, unchanged, and select USER\_NAME(VARCHAR2) from the Primary Key box. In this way, we select user\_name as the primary key for this table. Your finished second step should match the one shown in Figure 2.45. Click the Next button to continue to the next page to set the foreign key.

Since we have not yet created any other tables, we cannot select our foreign key for this LogIn table right now. We leave this job to be handled later. Click the Next button to go to the next page. The next page allows you to set some constraints on this table, as is shown in Figure 2.46.

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

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

After the LogIn table is created, the necessary editing tools are displayed at the top of this table. The top line of tools contains objects to be edited, and the next line includes the actual edit methods. The edit methods Add Column, Modify Column, Rename Column, and Drop Column are straightforward without question.

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

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

Add the following data into the first row: User Name – abrown, Pass Word – america, Faculty Id – B66750. Since this user is faculty, leave the Student Id column

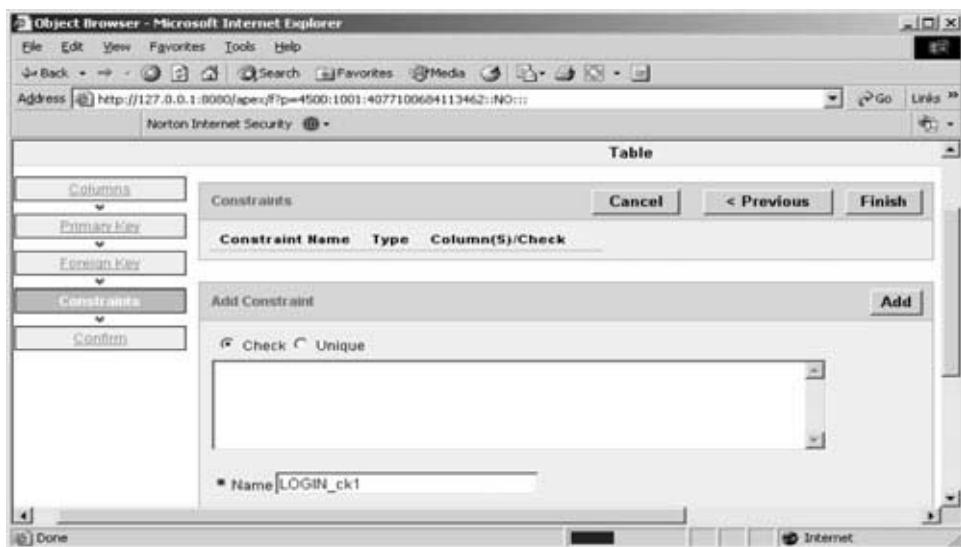


Figure 2.46. The fourth step – set up constraints.

blank (don't place a NULL in here, or you will have trouble when you create a foreign key for this table later!). Your finished first row is shown in Figure 2.50.

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

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

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

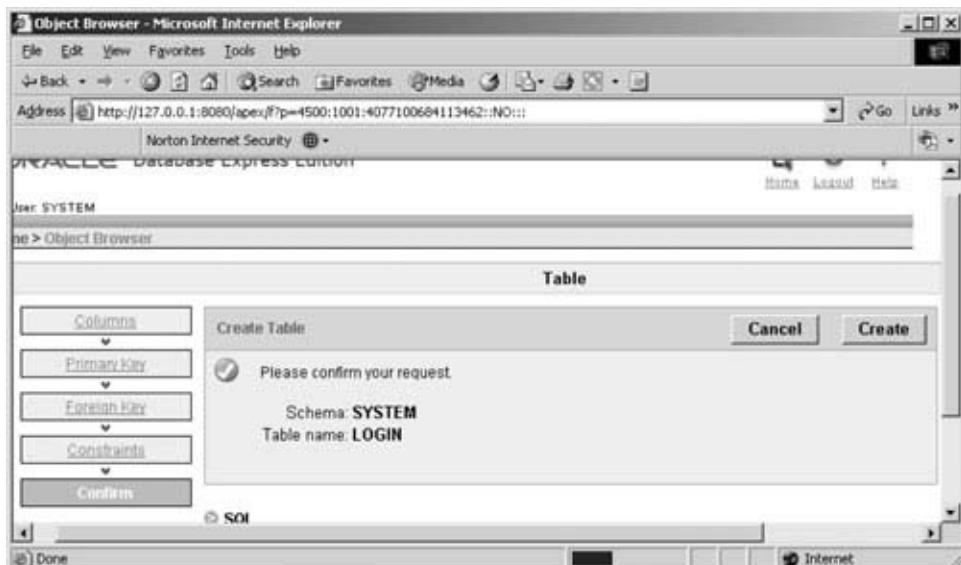


Figure 2.47. The last step – confirmation.

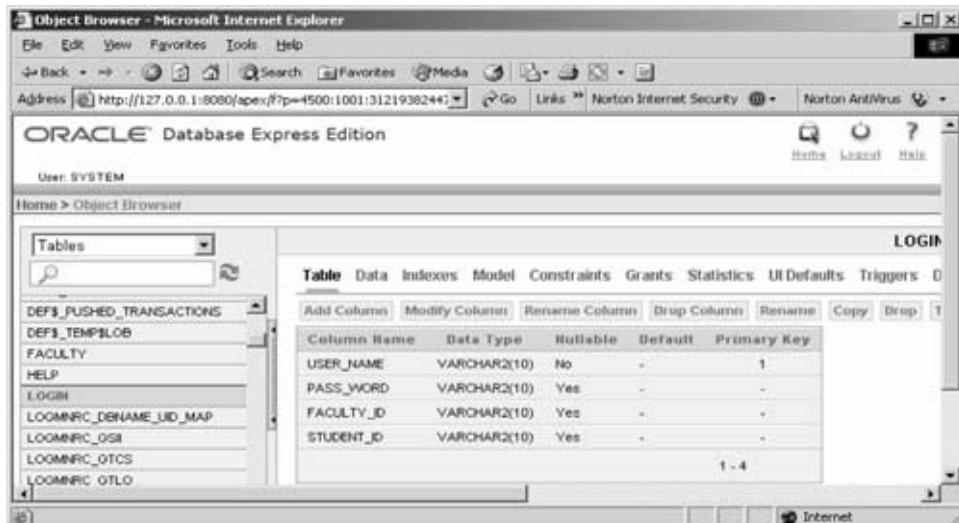


Figure 2.48. The created Login table.

### 2.11.2 Create the Faculty Table

Click the object tool Table and click the Create button to create another new table. Select the Table item to open a new table page. Enter Faculty in the Table Name box, and enter the following columns for this new table:

- faculty.id – VARCHAR2(10)
- name – VARCHAR2(20)
- office – VARCHAR2(10)
- phone – CHAR(12)



Figure 2.49. The opened Data page.

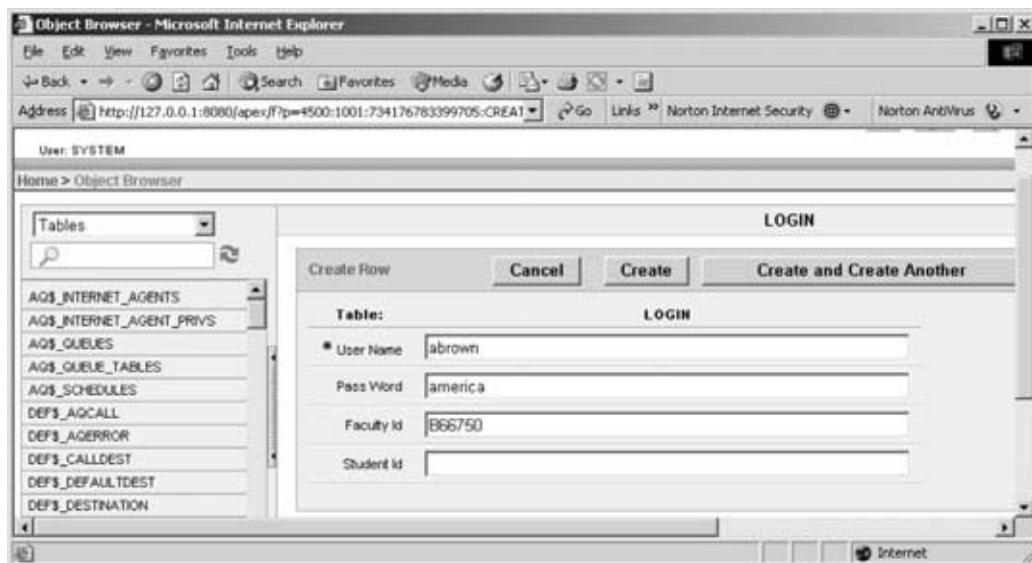


Figure 2.50. The data sheet view of the Login table.

- college – VARCHAR2(40)
- title – VARCHAR2(30)
- email – VARCHAR2(30)

Popular data types used in the Oracle database include NUMBER, CHAR, and VARCHAR2. Each data type has an upper bound and a lower bound. The difference between CHAR and VARCHAR2 is that the former is used to store a fixed-length string and the latter can store a varying-length string, which means that the real length of the string depends on the number of actual letters assigned to it. The data types for all columns are VARCHAR2 with one exception, the phone column, which has a CHAR type with an upper bound of 12 letters since the phone numbers

**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	
jkings	goodman	K69880	
sbhalla	india	B86590	
sjohnson	jermany	J33486	
ybai	reback	B78880	

The screenshot shows the Microsoft Internet Explorer Object Browser window. On the left, a sidebar lists various database objects such as AQ\$\_INTERNET\_AGENT\_PRIVS, AQ\$\_QUEUES, AQ\$\_QUEUE\_TABLES, AQ\$\_SCHEDULES, COURSE, DEF\$\_AQCALL, DEF\$\_AERROR, DEF\$\_CALLODEST, DEF\$\_DEFAULTDEST, DEF\$\_DESTINATION, DEF\$\_ERROR, DEF\$\_LOC, DEF\$\_ORIGIN, DEF\$\_PROPAGATOR, DEF\$\_PUSHED\_TRANSACTIONS, DEF\$\_TEMPLOB, FACULTY, HELP, LOGIN, LOGMNR\$\_DONAME\_UID\_MAP, LOGMNR\$\_DSI, LOGMNR\$\_GTC\$S, LOGMNR\$\_GTLO, and LOGMNR\$\_CTAS\_PART\_MAP. The main pane displays a table titled 'EDIT' with columns: USER\_NAME, PASS\_WORD, FACULTY\_ID, and STUDENT\_ID. The table contains 12 rows of data:

EDIT	USER_NAME	PASS_WORD	FACULTY_ID	STUDENT_ID
brown	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	K09800	-	
sbhalla	india	B86590	-	
sjohnson	jermamy	J33486	-	
ybai	reback	B78880	-	

Figure 2.51. The completed LogIn table.

are composed of 10 digits and the length 12 with two dashes. For all other columns, the length varies with the different information, so VARCHAR2 is selected for those columns.

The finished design view of the Faculty table is shown in Figure 2.52. You need to check the Not Null check box for the faculty\_id column since this column will be the primary key for this table.

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

Check Not populated for Primary Key since we don't want to use a Sequence object to automatically generate a sequence of numeric numbers for our primary key, and then select FACULTY\_ID(VARCHAR2) from the Primary Key text box. In this way, the faculty\_id column is selected as the primary key for this table.

Click the Next button to go to the next page.

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

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

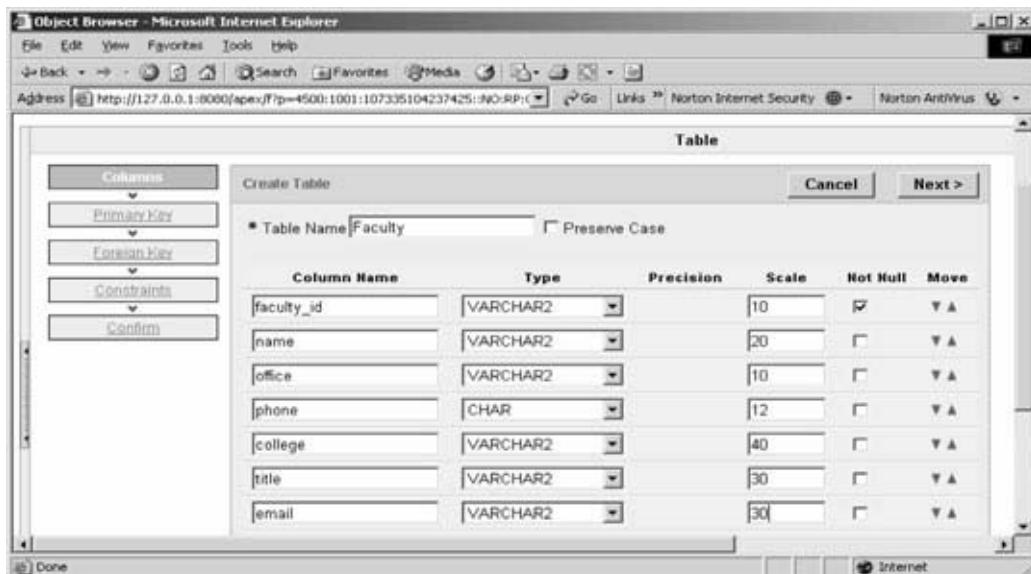


Figure 2.52. The finished design view of the Faculty table.

Click 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 the Create button after the last row. Your finished Faculty table should match the one shown in Figure 2.55.

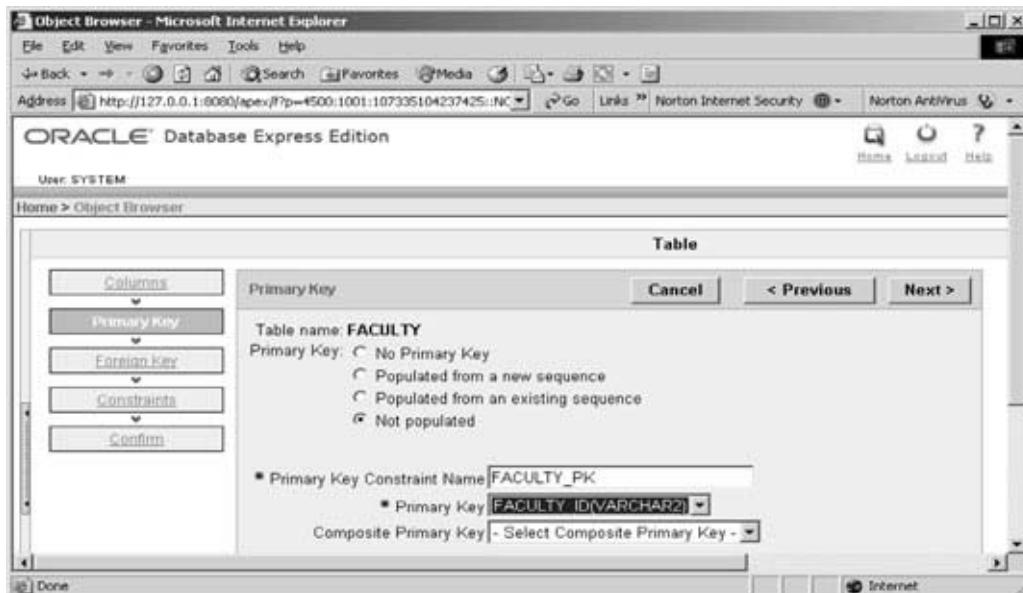


Figure 2.53. The open Primary Key window.

The screenshot shows the Oracle Database Express Edition Object Browser in Microsoft Internet Explorer. The left sidebar lists various system tables like AQ\$\_INTERNET\_AGENTS, AQ\$\_INTERNET\_AGENT\_PRIVS, etc. The main area displays the 'FACULTY' table structure with columns: FACULTY\_ID, NAME, OFFICE, PHONE, COLLEGE, TITLE, and EMAIL. The table has 7 rows and 7 columns. The primary key is FACULTY\_ID.

Column Name	Date Type	Nullable	Default	Primary Key
FACULTY_ID	VARCHAR2(10)	No	-	1
NAME	VARCHAR2(20)	Yes	-	-
OFFICE	VARCHAR2(10)	Yes	-	-
PHONE	CHAR(12)	Yes	-	-
COLLEGE	VARCHAR2(40)	Yes	-	-
TITLE	VARCHAR2(30)	Yes	-	-
EMAIL	VARCHAR2(30)	Yes	-	-

Figure 2.54. The completed columns in the Faculty table.

### 2.11.3 Create Other Tables

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

The data types used in the Course table are:

- course\_id: VARCHAR2(10) – Primary Key
- course: VARCHAR2(40)
- credit: NUMBER(1, 0) – precision = 1, scale = 0 (1-bit integer)
- classroom: CHAR(6)
- schedule: VARCHAR2(40)
- enrollment: NUMBER(2, 0) – precision = 2, scale = 0 (2-bit integer)

The data types used in the Student table are:

- student\_id: VARCHAR2(10) – Primary Key
- name: VARCHAR2(20)
- gpa: NUMBER(3, 2) – precision = 3, scale = 2 (3-bit floating point data with 2-bit after the decimal point)
- credits: NUMBER(3, 0) – precision = 3, scale = 0 (3-bit integer)
- major: VARCHAR2(40)
- schoolYear: VARCHAR2(20)
- email: VARCHAR2(20)

**Table 2.24.** The Data in the Faculty Table

faculty_id	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	Assistant Professor	ybai@college.edu
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sballa@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	Jenny King	MTC-324	750-378-1230	East Florida University	Professor	jkking@college.edu

EDIT	FACULTY_ID	NAME	OFFICE	PHONE	COLLEGE	TITLE	EMAIL
	A52990	Black Anderson	MTC-218	750-378-9967	Virginia Tech	Professor	banderson@college.edu
	A77567	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
	B73880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Assistant Professor	ybai@college.edu
	B86590	Sathish Balaji	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbalaji@college.edu
	H99118	Jeff Henry	MTC-306	750-330-0650	Ohio State University	Associate Professor	jhenry@college.edu
	J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson@college.edu
	K69880	Jeremy King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

Figure 2.55. The finished Faculty table.

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)

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

## 2.11.4 Create the Foreign Keys for Tables

### 2.11.4.1 Create the Foreign Key Between the LogIn and Faculty Tables

Now let's create the foreign key for the LogIn table and connect it to the primary key in the Faculty table. The faculty\_id is a foreign key in the LogIn table, but it is a primary key in the Faculty table. A one-to-many relationship exists between the faculty\_id in the Faculty table and the faculty\_id in the LogIn table.

Open the home page of Oracle Database 10g XE if it is not open, and then click the Object Browser icon and select Browse|Table to list all tables. Select the LogIn table from the left pane to open it, click the Constraints tab, and then click the Create button. Enter LOGIN\_FACULTY\_FK into the Constraint Name box, and select Foreign Key from the Constraint Type box, as shown in Figure 2.59. Check the On Delete Cascade check box. Then select FACULTY\_ID from the LogIn table as the foreign key column. Select the FACULTY table in the Reference Table Name box as the reference table, and select FACULTY\_ID

**Table 2.25. The Data in the Course Table**

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

**Table 2.26.** The Data in the Student Table

<b>student_id</b>	<b>name</b>	<b>gpa</b>	<b>credits</b>	<b>Major</b>	<b>schoolYear</b>	<b>email</b>
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

<b>s_course_id</b>	<b>student_id</b>	<b>course_id</b>	<b>credit</b>	<b>major</b>
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	CSIS
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

**Object Browser - Microsoft Internet Explorer**

The screenshot shows a Microsoft Internet Explorer window titled "Object Browser - Microsoft Internet Explorer". The address bar contains the URL "http://127.0.0.1:8080/apex/f?p=4500:1001:107335104237425::NO::RP;". The main content area displays a table titled "Course" with the following data:

EBIT	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	B66590
	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

row(s) 1 - 5 of 36

Figure 2.56. The completed Course table.

**Object Browser - Microsoft Internet Explorer**

The screenshot shows a Microsoft Internet Explorer window titled "Object Browser - Microsoft Internet Explorer". The address bar contains the URL "http://127.0.0.1:8080/apex/f?p=4500:1001:107335104237425::NO::RP;OB\_CURR1". The main content area displays a table titled "Student" with the following data:

EBIT	STUDENT_ID	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	3.67	78	Computer Engineering	Sophomore	hsmith@college.edu
	J77896	Erica Johnson	3.95	127	Computer Science	Senior	ejohnson@college.edu

row(s) 1 - 5 of 5

Figure 2.57. The completed Student table.

The screenshot shows the Oracle Object Browser in Microsoft Internet Explorer. The left sidebar lists various database objects like AQ\$\_INTERNET\_AGENTS, AQ\$\_QUEUES, etc. The main pane displays the 'StudentCourse' table with the following data:

	EDIT	S_COURSE_ID	STUDENT_ID	COURSE_ID	CREDIT	MAJOR
	[Edit]	1000	H10210	CSC-131D	3	CE
	[Edit]	1001	B92996	CSC-132A	3	CS/MS
	[Edit]	1002	J77896	CSC-335	3	CS/MS
	[Edit]	1003	A78835	CSC-331	3	CE
	[Edit]	1004	H10210	CSC-234B	3	CE
	[Edit]	1009	A78835	CSE-434	3	CE
	[Edit]	1006	B92996	CSC-233A	3	CS/MS
	[Edit]	1007	A78835	CSC-132A	3	CE
	[Edit]	1008	A78835	CSE-432	3	CE
	[Edit]	1014	A78835	CSE-438	3	CE
	[Edit]	1010	J77896	CSC-439	3	CS/MS
	[Edit]	1011	H10210	CSC-132A	3	CE
	[Edit]	1012	H10210	CSC-331	2	CE
	[Edit]	1013	A78835	CSC-335	3	CE
	[Edit]	1016	A97850	CSC-132B	3	ISE

Figure 2.58. The completed StudentCourse table.

The screenshot shows the Oracle Object Browser in Microsoft Internet Explorer. The left sidebar lists various database objects. The main pane shows the 'Add Constraint' dialog for the 'LOGIN' table:

- Schema:** SYSTEM
- Table:** LOGIN
- Constraint Name:** LOGIN\_FACULTY\_FK
- Constraint Type:** Foreign Key (selected)
- On Delete Cascade:** Checked
- Foreign Key Column(s):** FACULTY\_ID, STUDENT\_ID
- Reference Table Name:** FACULTY
- Reference Table Column List:** COLLEGE, EMAIL, FACULTY\_ID, NAME, OFFICE

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

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

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

#### 2.11.4.2 Create the Foreign Key Between the LogIn and Student Tables

The relationship between the Student table and the LogIn table is a one-to-many relationship. The student\_id in the Student table is a primary key, and the student\_id in the LogIn table is a foreign key. Multiple instances of a student.id can exist in the LogIn table, but only one, unique student.id can be in the Student table.

To create a foreign key from the LogIn table and connect it to the primary key in the Student table, you first need to open the home page of Oracle Database 10g XE if it is not open. On the open home page window, click the Object Browser icon and select Browse|Table to list all tables. Select the LogIn table from the left pane to open it, click the Constraints tab, and then click the Create button. Enter LOGIN\_STUDENT\_FK into the Constraint Name box, and select Foreign Key from the Constraint Type box, which is shown in Figure 2.60. Check the On Delete Cascade check box. Then select STUDENT\_ID from the LogIn table in the Foreign Key Column box. Select the STUDENT table in the Reference Table Name box as the reference table, and select STUDENT\_ID from the Reference Table Column List box as the reference table column. Your finished Add Constraint window should match the one shown in Figure 2.60.

Recall that when we created the LogIn table in Section 2.11.1, we emphasized that a NULL should not be placed into the blank fields in the faculty\_id and student\_id columns and that these fields should be left blank. The reason for this is that an ALTER TABLE command will be issued when you create a foreign key for the

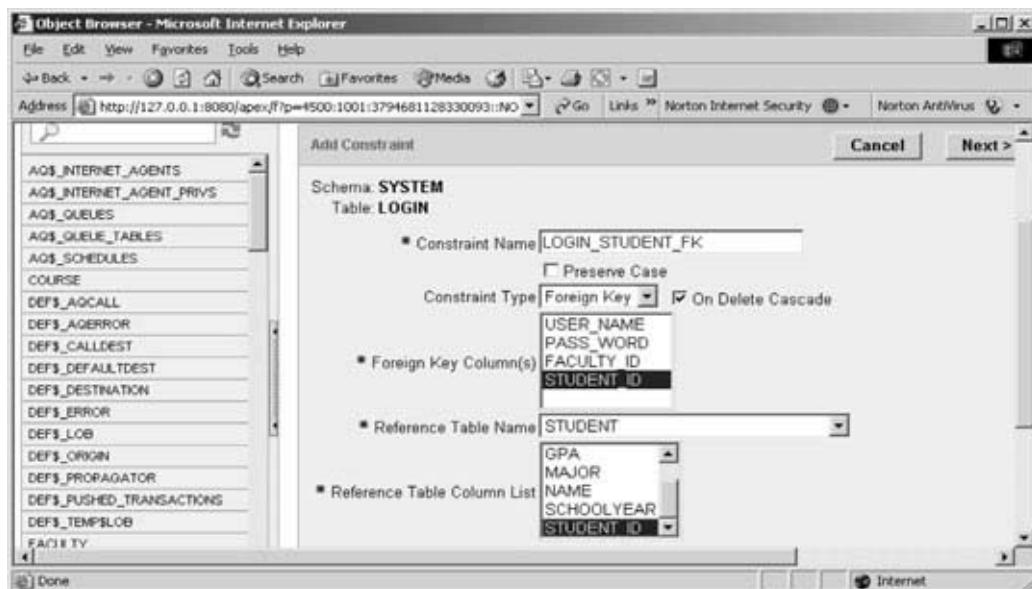


Figure 2.60. Create the foreign key between the LogIn and the Student table.

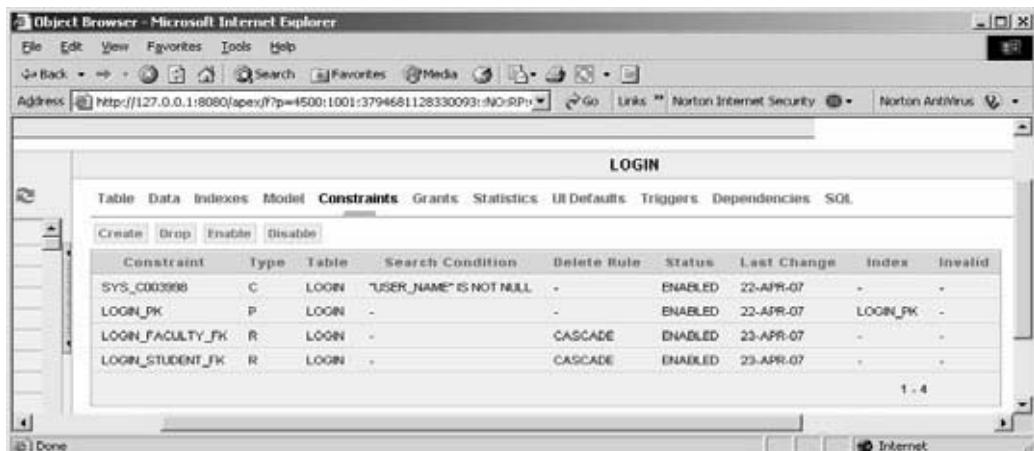


Figure 2.61. The finished foreign key creation window for the LogIn table.

LogIn table, and NULL cannot be recognized by this command; therefore an error, ORA-02298, would occur and your creation of the foreign key would fail.

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

Your finished foreign key creation window for the LogIn table should match the one shown in Figure 2.61.

#### **2.11.4.3 Create the Foreign Key Between the Course and Faculty Tables**

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

Open the home page of Oracle Database 10g XE if it is not open. On the home page window, click the Object Browser icon and select Browse|Table to list all tables. Select the Course table from the left pane to open it, click the Constraints tab, and then click the Create button. Enter COURSE\_FACULTY\_FK into the Constraint Name box, and select Foreign Key in the Constraint Type box, which is shown in Figure 2.62. Check the On Delete Cascade check box. Then select FACULTY\_ID from the Course table in the Foreign Key Column box. Select the FACULTY table from the Reference Table Name box as the reference table, and select FACULTY\_ID from the Reference Table Column List box as the reference table column. Your finished Add Constraint window should match the one shown in Figure 2.62.

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

#### **2.11.4.4 Create the Foreign Key Between the StudentCourse and Student Tables**

The relationship between the Student table and the StudentCourse table is one-to-many. The primary key student\_id in the Student table is a foreign key in

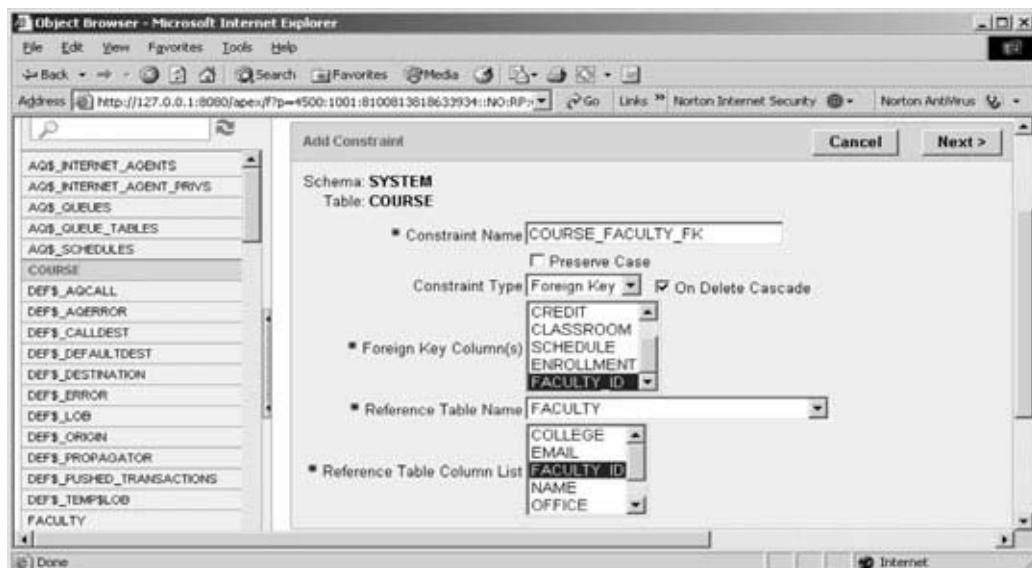


Figure 2.62. Create the foreign key between the Course and the Faculty table.

the StudentCourse table since one student can take multiple courses. To create this relationship by using the foreign key, open the home page of Oracle Database 10g XE if it is not open. On the open home page window, click the Object Browser icon and select Browse|Table to list all tables. Select the StudentCourse table from the left pane to open it, click the Constraints tab, and then click the Create button. Enter STUDENTCOURSE\_STUDENT\_FK into the Constraint Name box, and select Foreign Key from the Constraint Type box, which is shown in Figure 2.64. Check the On Delete Cascade check box. Then select STUDENT\_ID from the StudentCourse table in the Foreign Key Column box. Select the STUDENT table in the Reference Table Name box as the reference table, and select STUDENT\_ID in the Reference Table Column List box as the reference table column. Your finished Add Constraint window should match the one shown in Figure 2.64.

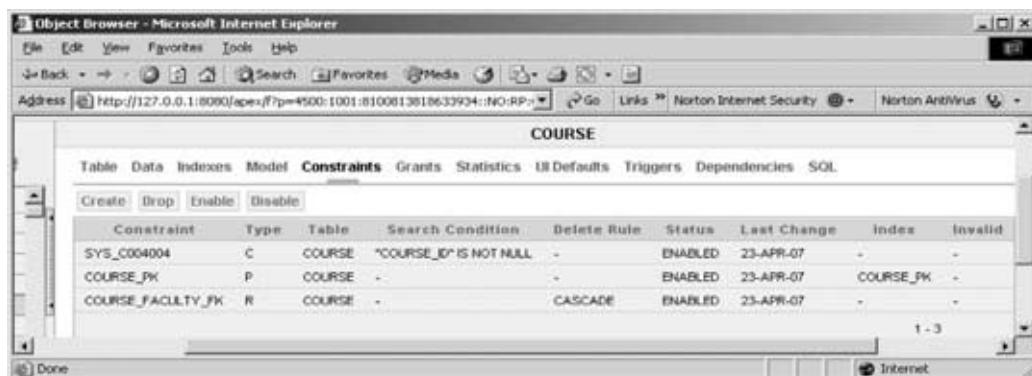


Figure 2.63. The finished foreign key creation window for the Course table.



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

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

#### **2.11.4.5 Create the Foreign Key Between the StudentCourse and Course Tables**

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

To create this relationship by using the foreign key, open the home page of Oracle Database 10g XE if it is not open. On the opened home page window, click the Object Browser icon and select Browse|Table to list all tables. Select the StudentCourse table from the left pane to open it, click the Constraints tab, and then click the Create button. Enter STUDENTCOURSE.COURSE.FK into the Constraint Name box, and select the Foreign Key from the Constraint Type box, which is shown in Figure 2.65. Check the On Delete Cascade check box. Then select COURSE\_ID from the StudentCourse table in the foreign key column. Select the COURSE table in the Reference Table Name box as the reference table, and select COURSE\_ID in the Reference Table Column List box as the reference table column. Your finished Add Constraint window should match the one shown in Figure 2.65.

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

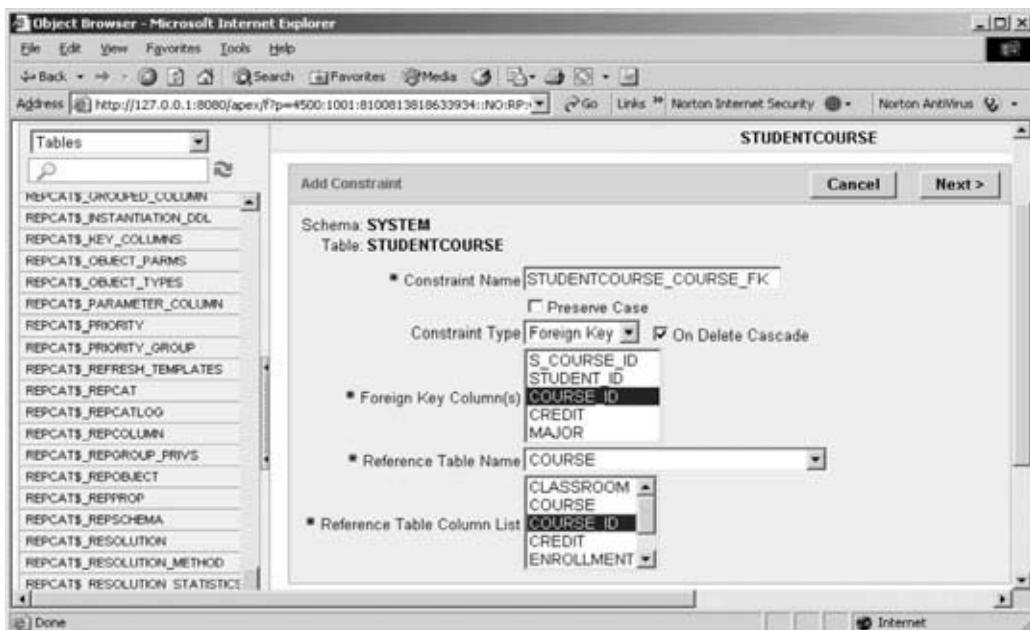


Figure 2.65. Create the foreign key between the StudentCourse and Course tables.

Our database creation for Oracle Database 10g XE is completed. The completed Oracle Database 10g XE sample database CSE\_DEPT that is represented by a group of table files can be found in the folder **database\Oracle**, located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). Refer to Appendix F to get more detailed information if you want to use this sample database in your Visual Basic applications.

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

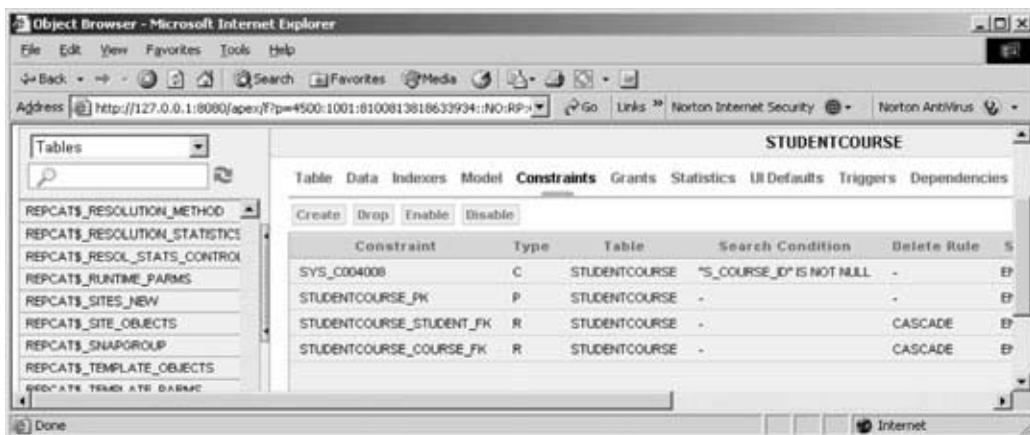


Figure 2.66. The finished foreign key creation window for the StudentCourse table.

## **2.12 CHAPTER SUMMARY**

A detailed discussion and analysis of the structure and components of databases was provided in this chapter. Some key technologies for developing and designing a database were also discussed in this part. The procedure to develop a relational database was analyzed in detail and demonstrated with some real data tables in our sample database, CSE\_DEPT. The process of developing and building a sample database was discussed in detail, including the following points:

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

In the second part of this chapter, three sample databases were developed with three popular database management systems, namely, Microsoft Access, SQL Server 2005, and Oracle Database 10g XE. All three sample databases will be used in the following chapters.

## **2.13 HOMEWORK**

### **I. True/False Selections**

- \_\_\_\_\_ 1. The 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 a table in 1NF is automatically in 2NF.
- \_\_\_\_\_ 4. A table is in first normal form if there are no repeating groups of data in any column.
- \_\_\_\_\_ 5. When a user perceives a database as made up of tables, it is called a Network Model.
- \_\_\_\_\_ 6. The 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 blank without any letter in it.
- \_\_\_\_\_ 8. When creating data tables in an SQL Server database, a blank field can be kept blank without any letter in it.
- \_\_\_\_\_ 9. The name of each data table in an SQL Server database must be prefixed by the keyword .dbo.
- \_\_\_\_\_ 10. The Sequence object in an Oracle database is used to automatically create a sequence of numeric numbers that work as the primary key.

### **II. Multiple Choices**

1. There are many advantages to using an integrated database approach over a file processing approach. These include \_\_\_\_\_.

- a. Minimizing data redundancy
  - b. Improving security
  - c. Data independence
  - d. All of the above
2. The entity integrity rule implies that no attribute that is a member of the primary key may accept a(n) \_\_\_\_\_.
    - 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. \_\_\_\_\_ 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. Client server databases have several advantages over 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 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, one must first 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 the Oracle database is a string variable with a \_\_\_\_\_.
  - a. Limited length
  - b. Fixed length
  - c. A 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. Indicated by either NULL or a blank
  - d. Avoided

### **III. Exercises**

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

---

# 3

---

## Introduction to ADO.NET

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

### 3.1 ADO AND ADO.NET

ActiveX Data Objects (ADO) is an interface developed on the basis of 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 this 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 Data Access Objects (DAO), followed by Remote Data Objects (RDO), which was based on Open Database Connectivity (ODBC). In the late 1990s, the ADO that is based on OLE DB was developed. This technology was widely applied in most object-oriented programming and database applications during the last decade.

ADO.NET is a new version of ADO, and it is based mainly on the Microsoft .NET Framework. The underlying technology applied in ADO.NET 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 Framework provides developers with a significant number of benefits, including a more robust, evidence-based security model; automatic memory management; and native Web services support. For new development, ADO.NET is highly recommended as a preferred technology because of its powerful managed runtime environment and services.

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

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

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, refer to both System.Data.dll and System.Xml.dll.

Basically speaking, ADO.NET provides a set of classes to support development of 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 central to 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 different namespaces. The DataSet and DataTable classes are located at the System.Data namespace. The classes of the Data Provider are located at 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 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 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 a table container and can contain multiple data tables. These data tables are only a mapping to the real data tables in your database. But these data tables can also be used separately without connecting to the DataSet class. In this case, each data table can be considered 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. The four classes that belong to Data Provider, namely, Connection, Command, DataAdapter, and DataReader, are often called Data Provider-dependent components.

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

### 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 3.1 shows a typical architecture of ADO.NET.

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 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

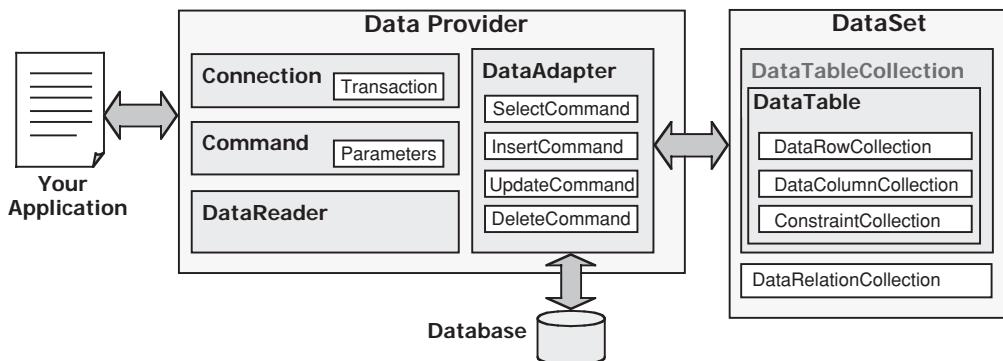


Figure 3.1. A typical architecture of ADO.NET.

the DataAdapter to perform the data query is exactly to call the Command object with different parameters.

Another ADO.NET architecture is shown in Figure 3.2.

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

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

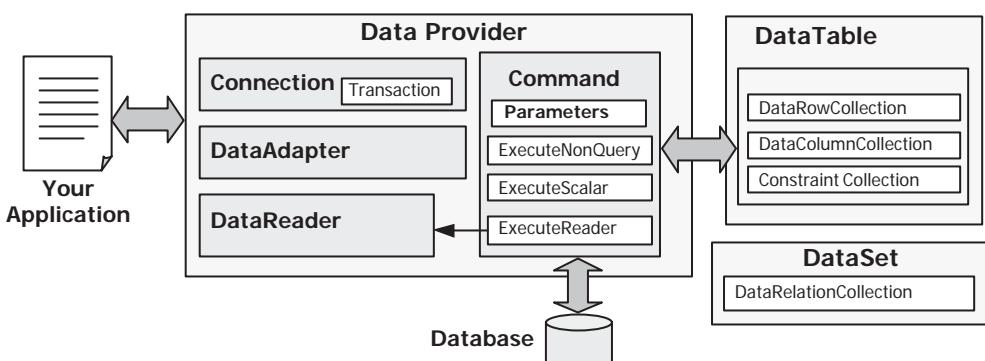


Figure 3.2. Another architecture of ADO.NET.

## 3.4 THE COMPONENTS OF ADO.NET

As 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 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 data query and data-accessing operations between your data source and your applications
- Disconnect the data source when the data operations are done

The Data Provider is physically composed of a binary library file, and this library is in the DLL file format. Sometimes this DLL file depends on other DLL files, so in fact a Data Provider can be made up of several DLL files. Based on the different kinds of databases, 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 Embeding 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 them from older Data Providers such as Microsoft ODBC, Microsoft OLE DB, Microsoft SQL Server, and Oracle, in some books all Data Providers included in ADO.NET are extended by the suffix .NET, such as OLEDB.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 the different Data Providers and the DataSet and DataTable classes.

Since the different Data Providers are located at different namespaces, you must first import the appropriate namespace into your Visual Basic.NET 2005 project, exactly into the form's code window, whenever you want to use that Data Provider. Also, all classes provided by that Data Provider must be prefixed by the associated keyword. For example, you must use “imports System.Data.OleDb” to import the namespace of the OLEDB.NET Data Provider if you want to

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

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

use this Data Provider in your project. Also, all classes belonging to that Data Provider must be prefixed by the associated keyword OleDb, such as OleDbConnection, OleDbCommand, OleDbDataAdapter, and OleDbDataReader. The same thing holds true for all other Data Providers.

Although different Data Providers are located at the different namespaces and have different prefixes, the classes of these Data Providers have similar methods or properties with the same names. For example, no matter what kind of Data Provider you are using, such as OleDb, Sql, or Oracle, it has methods and properties with the same names, such as Connection String property, Open() and Close() methods, and the ExecuteReader() method. This provides flexibility and allows the programmer to use different Data Providers to access different data sources by modifying only 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 select the appropriate Data Provider to develop 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 the 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 includes older versions of SQL Server and Oracle as well as some third-party databases such as Sybase.

**Table 3.2.** The Compatibility between OLE DB and OLEDB.NET

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

### 3.4.1.2 The OLE DB 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 OLEDB.NET data access technique supports the following Data Providers:

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

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

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

### 3.4.1.3 The SQL Server Data Provider

This Data Provider provides access to SQL Server databases version 7.0 or later 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 are located at the System.Data.SqlClient namespace. Although Microsoft provides different Data Providers to access data in the SQL Server database, such as 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 can only work for SQL Server version 7.0 and later. If an older version of SQL Server is used, you need to use either an OLEDB.NET or an 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 provides the most efficient way to access the Oracle database.

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

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

### **3.4.2 The Connection Class**

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

The Connection object you want to use depends on the type of the data source you selected. The Data Provider provides four different Connection classes, and each one is matched to a 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 2005. One of them does not accept any arguments, but the other accepts a connection string as the argument, and this constructor is most commonly used for data connections.

The connection string is a property of the Connection class and provides all necessary information to connect to your data source. This connection string contains quite a few parameters to define a connection, but only five of them are popularly utilized for most data-driven applications:

1. Provider
2. Data Source

**Table 3.3. The Connection Classes and Databases**

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

3. Database
4. User ID
5. Password

For different databases, the parameters contained in the connection string may differ. For example, both OLE DB and ODBC databases need all these five parameters to be included in a connection string to connect to an OLEDB or ODBC data source. But for the SQL Server database connection, you may need to use the Server to replace the Provider parameter, and for the Oracle database connection, you do not need the Provider and Database parameters at all. You can find these differences in Section 4.18.1 in Chapter 4.

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

---

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

---

In your real application, **xxx** should be replaced by the selected Data Provider, such as OleDb, Sql, or Oracle. You need to use the real parameter values implemented in your applications to replace the 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 SQL Server 2005 Express on your computer, then Provider should be localhost. If you are using a remote SQL Server, 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 security issues 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.Jet.OLEDB.4.0;" &_
    "Data Source=C:\database\CSE_DEPT.mdb;" &_
    "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 the 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 needed when you finish the data operations and 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. It is used to open a connection to a data source with the property settings specified by the connection string. You must make sure that this connection is bug-free, or in other words, that the connection is successful and you can use it to access data from your application to your desired data source without any problem. An efficient way to do this is to use the Try....Catch block to embed the Open() operation to try to find and catch the typical possible errors caused by this connection. Sample coding of the opening of an OLE DB connection is shown in Figure 3.3.

```
Dim strConnectionString As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\database\CSE_DEPT.mdb;"

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 3.3. Sample coding of the opening of a connection.

The Microsoft.Jet.OLEDB.4.0 driver is used as the data provider, and the Microsoft Access database file CSE\_DEPT.mdb is located in the database folder on the 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 happen after this Open() method is executed. A related message would be displayed if any one of those errors occurred and was caught.

To make sure that the connection is bug-free, one of the properties of the Connection class, state, is used. This property has two possible values: Open or Closed. By checking this property, you can confirm whether 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 is used to close a connection between your database and your applications when you finish your data operations using the data source. You should close any connection object you connected to your data source after you finish the data access to that data source; otherwise a possible error may be encountered when you try reopen that connection the next time you run your project.

Unlike the Open() method, which is key to your data access and operations on 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 errors 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 is used to release the resources used by the Connection object. You need to call this method after the Close() method is executed to perform a cleanup job to release all resources used by the Connection object during data access and operations on your data source. Although it is not necessary to call the Dispose() method to do the cleanup job since one of system tools, Garbage Collection, can periodically check

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

**Figure 3.4.** Sample coding for the cleanup of resources.

and clean all resources used by unused objects in your computer, it is highly recommended for you to use this coding to make your program more professional and efficient.

After the Close() and Dispose() methods have been executed, you can release your reference to the Connection instance by setting it to Nothing. A piece of sample code is shown in Figure 3.4.

Now that we have finished the discussion of the first component 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 objects, we discuss these two objects in one section.

### 3.4.3 The Command and Parameter Classes

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

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

Depending on the architecture of ADO.NET, the Command object can have two different roles when you are using it to perform a data query or a data action. Refer to Figures 3.1 and 3.2. In Figure 3.1, if a TableAdapter is utilized to perform a data query and all data tables are embedded into the DataSet as a data catching unit, the Command object is embedded into the different data query methods of the TableAdapter, namely, 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 an Executing method to run the Command object directly; instead you can run it by executing the associated method of the TableAdapter.

In Figure 3.2, each data table can be considered 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 suitable properties to it in order to use it 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 ten properties, but only four of them are used in most applications:

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

The Connection property is used to hold a valid Connection object, and the Command object can be executed to access the connected database based on this Connection object.

The CommandType property is used to indicate what kind of command stored in the CommandText property should be executed. In other words, the CommandType property specifies how the CommandText property can be interpreted. In all, 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 Parameter objects. You need to note that Parameters is a collection but Parameter is an object, which means that the former contains a group of objects and you can add the latter to the former.

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

### 3.4.3.2 The Constructors and Properties of the Parameter Class

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

The first constructor is a blank one, and you need to initialize each property of the Parameter object one by one if you want to use this constructor to instantiate a new Parameter object. The three popular properties of a Parameter object are

- ParameterName
- Value
- DbType

```
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 3.5. 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

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

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

Even though the data types provided by ADO.NET and ADO.NET Data Provider are different, they have a direct connection between them. As a user, you can use any data type you like, and the other 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 different Data Providers.

The default data type for the DbType property is String.

### 3.4.3.3 Parameter Mapping

When you add a Parameter object to the Parameters collection of a Command object by attaching the Parameter object to the Parameters property of the Command class, the Command object needs to know the relationship between the 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.

**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

Different parameter mappings are used for different Data Providers. Table 3.5 lists these mappings.

Both OLE DB and ODBC Data Providers use 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 other words, the order in which the parameters appear in an SQL statement and the order in which the parameters are added into the Parameters collection should be exactly identical. Positional Parameter Mapping is indicated with a question mark.

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

---

```
SELECT 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 the 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 the 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 use Named Parameter Mapping, which means that each parameter, either defined in an SQL statement or added into a Parameters collection, is identified by the name. In other words, the name of the parameter appearing in an SQL statement or a stored procedure must be identical to the name of the parameter 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)
```

---

```

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 3.6.** An example of initializing the property of a Parameter object.

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 keyword LIKE for two parameters. These changes are required by the SQL Server Data Provider.

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

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

You can see from this piece of code that the name of each parameter you use for each Parameter object must be identical with the name 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 add 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 ten methods, but only two of them are most often utilized in the data-driven applications: Add() and AddWithValue().



The Parameters property in the Command class is a collection of a set of Parameter objects. You first need 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 mentioned in the last section, you do not need to worry about the order in which you add the parameter into the Parameter object if you are using a Named Parameter Mapping Data Provider such as SQL Server or Oracle. But you must pay attention to the order in which you add the parameter into the Parameter object if

you are using a Positional Parameter Mapping Data Provider such as OLE DB or ODBC.

To add Parameter objects to a Parameters collection of a Command object, two popular methods are generally used, Add() and AddWithValue().

The Add() method is an overloaded method and 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.

The second method contains two arguments. The first 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 3.7.

The top section is used to create and initialize the Parameter objects, which we have discussed in the previous sections.

Then, 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 previously.

```
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 3.7. 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 3.8.** Three popular protocols of the constructor of the Command class.

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 3.8 (an SQL Server Data Provider is used as an example).

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

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

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

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

```
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 3.9.** An example of creating an SqlCommand object.

**Table 3.6.** Methods of the Command Class

Method Name	Functionality
ExecuteReader	Executes commands that return rows, such as an 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.

The top two lines of the coding create an SQL statement with two dynamic parameters, username and password. Then two strings are concatenated to form a complete string. Two SqlParameter objects and one SqlCommand object 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 look at 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 a prerequisite to execute a Command object. The actual execution of a Command object is to run one of the 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. Table 3.6 lists these methods.

As 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 that the user selected and used in the application, such as OleDbCommand, OdbcCommand, SqlCommand, and 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 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, different DataReader objects should be utilized as the data receiver to hold the returned rows. Remember, the DataReader class is a read-only class and can only be used

```

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 3.10.** An example of running the ExecuteReader method.

as a data holder. You cannot perform any data updating by using the Data-Reader.

The sample coding in Figure 3.10 can be used to execute an SQL SELECT statement. As shown in Figure 3.10, as the ExecuteReader method is called, an SQL SELECT statement is executed to retrieve the id, user\_name, and pass\_word from the LogIn table. The returned rows are assigned to the SqlDataReader object. Please note that the SqlCommand object should already be created and initialized before the ExecuteReader() method is called.

#### **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.

An example using this method is shown in Figure 3.11.

In this example, the SQL SELECT statement is used to pick up a password based on the username ybai from the LogIn data table. This password can be considered 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 4.9 in Chapter 4 provides an example of using this method to pick up a single value, a password, from the LogIn data table in the CSE\_DEPT database.

#### **3.4.3.6.3 The ExecuteNonQuery Method**

As mentioned, the ExecuteReader method is a read-out method and can only be used to perform a data query job. To execute SQL statements such as the INSERT, UPDATE, or DELETE commands, the ExecuteNonQuery method is needed.

```

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 3.11.** An example 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 3.12.** An example of using the ExecuteNonQuery method.

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

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

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

#### 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 the database using a Connection object and uses Command objects to retrieve data from the database and populate the data to the DataSet and related classes such as DataTables. Also, the DataAdapter uses Command objects to send data from the DataSet to the database.

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

As 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 Data Provider that is used. Four popular DataAdapters are OleDbDataAdapter, OdbcDataAdapter, SqlDataAdapter, and OracleDataAdapter. Different DataAdapters are located at different namespaces.

If you are connecting to an SQL Server database, you can increase overall performance by using the SqlDataAdapter along with its associated SqlCommand and SqlConnection objects. For OLE DB-supported data sources, use the OleDb-DataAdapter with its associated OleDbCommand and OleDbConnection objects.

**Table 3.7. The Constructors of the DataAdapter Class**

Constructor	Description
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

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 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.

#### **3.4.4.3 The Methods of the DataAdapter Class**

The DataAdapter has more than ten methods available to help 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 most often used. 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 data manipulations such as Insert, Update, and Delete with the database and the DataSet.

**Table 3.8. The Public Properties of the DataAdapter Class**

Property	Description
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

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

#### 3.4.4.4 The Events of the DataAdapter Class

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

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

Starting from step **A**, an SQL SELECT statement string is created with some other new object declarations, such as a new instance of 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. 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**.

**Table 3.10.** The Events of the DataAdapter Class

Event	Description
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 sqlDataSet As DataSet

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

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

F Dim intValue As Integer = sqlDataAdapter.Fill(sqlDataSet)
If intValue = 0 Then
    MessageBox.Show("No valid faculty found!")
End If

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

```

**Figure 3.13.** An example of using the SqlDataAdapter to fill the DataSet.

An integer variable `IntValue` is used to hold the value returned by 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 no row has been filled into the `Faculty` table in the `DataSet`, an error message is displayed. Otherwise, this fill is successful.

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

### 3.4.5 The DataReader Class

The `DataReader` class is a read-only class and 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. 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 constructors. The following code, which 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`

**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.

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 you finish a data query using that DataReader to avoid the troubles caused by multiple connections to the database.

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

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

**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.

```

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

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

```

**Figure 3.14.** An example of using the SqlDataReader object.

When you run the ExecuteReader method to retrieve data rows from a database and assign them to a DataReader object, the DataReader can only retrieve and hold one row each time. 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 way to read data from the database, and you should use this object whenever you want to read data from the database from start to finish to populate a list on a form or to populate an array or collection. It can also be used to populate a DataSet or a DataTable.

Figure 3.14 shows an example of using the SqlDataReader object to continuously retrieve all records (rows) from the Faculty data table, supposing that a Connection object sqlConnection has been created. Starting from section **A**, a new SqlCommand and an SqlDataReader object are created with an SQL SELECT statement string object. The Command object is initialized in section **B**. In section **C**, the ExecuteReader method is called to retrieve a data row from the Faculty data table and assign the resulting row to the SqlDataReader object. By checking 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

**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.
InvalidOperationException	If you try to convert a database value using one of the Get methods to convert a column value to a specific data type, this exception occurs.
NotSupportedException	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.

clean up the sources we used. In section E, the Close and Dispose methods (refer to Table 3.12) are utilized to finish this cleaning job.

Before we move on to the next section, we need to discuss one more thing, the DataReader Exceptions. Table 3.13 lists frequently used Exceptions.

You can use the Try...Catch block to handle those Exceptions in your applications to avoid the unnecessary debugging process 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 the ADO.NET architecture. The DataSet consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. In other words, a DataSet object can be considered a table container that contains a set of data tables with the DataRelation as a bridge to relate the tables together. The relationship between a DataSet and a set of DataTable objects can be defined as follows:

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

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

Although not tied to any specific database, the `DataSet` class is designed to contain relational tabular data such 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 a direct mapping to the real table in the database. For example, the `LogInDataTable` is a data table component or `DataTable` that can be mapped to the real table `LogIn` in the `CSE_DEPT` database. Any relationship between 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 a mini database engine: it can contain information about tables it holds such as the column name and data type, all relationships between tables, and, more important, most management functionalities of the tables, such as the ability to browse, select, insert, update, and delete data from tables.

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

With Visual Basic 2005, 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`).

To summarize, 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 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* that contains all mapped data tables from the database you connected to your project. In some books, the term *data source* means the `DataSet`.

Figure 3.15 shows a global relationship between the `DataSet` object, other data objects, and the Visual Basic 2005 application.

A `DataSet` can be typed or untyped, and the difference between them is that a typed `DataSet` object has a schema and an untyped `DataSet` does not. In your data-driven applications, you can use either kind of `DataSet`. But the typed `DataSet` has more support in Visual Studio 2005.

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

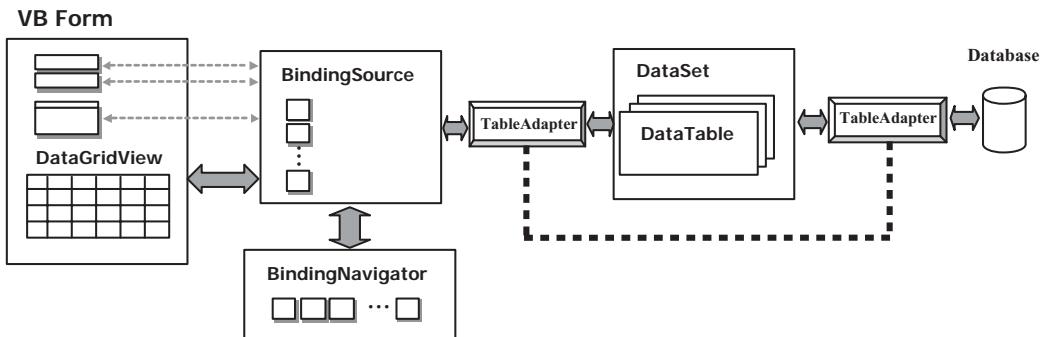


Figure 3.15. A global representation of the DataSet and other data objects.

### 3.4.6.1 The DataSet Constructor

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

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

### 3.4.6.2 The DataSet Properties

The DataSet class has more than fifteen public properties. Table 3.15 lists the most often used properties.

Among these properties, DataSetName, IsInitialized, and Tables are the properties most often used in data-driven applications.

### 3.4.6.3 The DataSet Methods

The DataSet class has more than thirty 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, you should 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 finish 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

The DataSet class has three public events. 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

**Table 3.14. Popular Constructors of the DataSet Class**

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

**Table 3.15.** Public Properties of the DataSet Class

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

has been initialized to your applications. The `MergeFailed` event is triggered when a conflict occurs and the `EnforceConstraints` property is set to `True` when you try to merge a `DataSet` with an array of `DataRow` objects, another `DataSet`, or a `DataTable`.

Before we finish this section, we will create, initialize, and implement a real `DataSet` object in a data-driven application. The code shown in Figure 3.16 is used to illustrate these issues and an SQL Server Data Provider is utilized for this

**Table 3.16.** Public Methods of the DataSet Class

Method Name	Functionality
BeginInit	Begins the initialization of a <code>DataSet</code> that is used on a form or used by another component. The initialization occurs at run time.
Clear	Clears the <code>DataSet</code> of any data by removing all rows in all tables.
Copy	Copies both the structure and data for the <code>DataSet</code> .
Dispose	Releases the resources used by the <code>MarshalByValueComponent</code> .
GetChanges	Gets a copy of the <code>DataSet</code> containing all changes made to it since it was last loaded, or since <code>AcceptChanges</code> was called.
HasChanges	Gets a value indicating whether the <code>DataSet</code> has changes, including new, deleted, or modified rows.
Load	Fills a <code>DataSet</code> with values from a data source using the supplied <code>IDataReader</code> .
Merge	Merges a specified <code>DataSet</code> , <code>DataTable</code> , or array of <code>DataRow</code> objects into the current <code>DataSet</code> or <code>DataTable</code> .
Reset	Resets the <code>DataSet</code> to its original state. Subclasses should override <code>Reset</code> to restore a <code>DataSet</code> 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 <code>DataSet</code> .
WriteXmlSchema	Writes the <code>DataSet</code> structure as an XML schema.

**Table 3.17.** Public Events of the DataSet Class

Event Name	Description
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.

example. 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 value returned by 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 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 (the DataTable in the sqlDataSet, to be precise) has been filled by some data rows, the returned value should be greater than 0. Otherwise it means that errors occurred for this fill and an error message will be displayed to warn the user.

```

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 3.16.** An example of using the DataSet.

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

Note that when the `Fill` method is executed to fill a `DataSet`, it 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 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**

The `DataTable` class can be considered as a container that holds the `Rows` and `Columns` collections, and the `Rows` and `Columns` collections contain a set of rows (or `DataRow` objects) and a set of columns (or  `DataColumn` objects) from a data table in a database. The `DataTable` is a direct mapping to a real data table in a database or a data source, and it stores its data in a mapping area or a block of memory space that is associated with a data table in a database as your project runs. The `DataTable` object can be used in two ways, as mentioned in the previous sections. One way is that a group of `DataTable` objects, with each `DataTable` mapped to a data table in the real database, can be integrated into a `DataSet` object. All these `DataTable` objects can be populated by executing the `Fill` method of the `DataAdapter` object (refer to the example in Section 3.4.5.4). The argument of the `Fill` method is not a `DataTable`, but a `DataSet` object since all `DataTable` objects are embedded into that `DataSet` object already. The second way to use the `DataTable` is that each `DataTable` can be considered as a single, stand-alone data table object, and each table can be populated or manipulated by executing either the `ExecuteReader` or `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 exists no matter what kind of Data Provider you are using in your applications.

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

When accessing `DataTable` objects, note that they are conditionally case sensitive. For example, if one `DataTable` is named “faculty” and another is named

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

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

The DataTable also contains a collection of Constraint objects that can be used to ensure the integrity of the data. The DataTable class is a member of the System.Data namespace within the .NET Framework class library. You can create and use a DataTable independently or as a member of a DataSet, and DataTable objects can also be used in conjunction with other .NET Framework objects, including the DataView. As 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 states.

### **3.4.7.1 The DataTable Constructor**

The DataTable has four overloaded constructors. Table 3.18 lists the 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.

**Table 3.18. Three Popular Constructors of the DataTable Class**

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

```

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

```

**Figure 3.17.** An example of adding a DataTable into a DataSet.

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

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

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

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 twenty 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. 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 a DataColumn object and can be added into this Columns collection. The Rows property is similar. It contains a collection of DataRow objects that are composed of all rows in the current DataTable object. You can get the total number of columns and rows from the current DataTable by calling these two properties.

### **3.4.7.3 The DataTable Methods**

The DataTable class has about fifty different methods with thirty-three public methods. Table 3.20 lists some of the most often used methods.

Among these methods, three 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

**Table 3.19.** The Popular Properties of the DataTable Class

Property	Description
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 a DataColumn object. By calling this property, a collection of DataColumn objects existing 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 a DataRow object. By calling this property, a collection of DataRow objects existing in the DataTable can be retrieved.
TableName	Gets or sets the name of the DataTable.

**Table 3.20.** The Popular Methods of the DataTable Class

Method	Description
Clear	Clears the DataTable of all data.
Copy	Copies both the structure and data for this DataTable.
Dispose	Releases 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	Merges 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.

**Table 3.21.** The Public Events of the DataTable Class

<b>Event</b>	<b>Description</b>
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.

Accept Condition (As Boolean). The Value is used to update the data row if that row is 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 eleven public events. Table 3.21 lists these events.

The most often used events are ColumnChanged, Initialized, RowChanged, and RowDeleted. By using these events, one can track and monitor the real situations occurring in the DataTable.

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

Refer to Figure 3.18. Starting from step **A**, a new instance of the data table FacultyTable is created and initialized to a blank table.

In order to add data into this new table, you need to use the Columns and Rows collections. These two collections contain the DataColumn and DataRow objects. So next you need to create DataColumn and DataRow objects, respectively. Step **B** finishes 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.

In Step **D**, the second data column, with the column data type as string (System.String) and the column name as FacultyOffice, is added into the FacultyTable in a similar way as the first data column, FacultyId.

```

A   'Create a new DataTable
B   Dim FacultyTable As DataTable = New DataTable("FacultyTable")
C   'Declare DataColumn and DataRow variables
D   Dim column As DataColumn
E   Dim row As DataRow
F   Create new DataColumn, set DataType, ColumnName and add to DataTable
G   column = New DataColumn
H   column.DataType = System.Type.GetType("System.Int32")
I   column.ColumnName = "FacultyId"
J   FacultyTable.Columns.Add(column)

K   Create another column.
L   column = New DataColumn
M   column.DataType = Type.GetType("System.String")
N   column.ColumnName = "FacultyOffice"
O   FacultyTable.Columns.Add(column)

P   'Create new DataRow objects and add to DataTable.
Q   Dim Index As Integer
R   For Index = 1 To 10
S   row = FacultyTable.NewRow
T   row("FacultyId") = Index
U   row("FacultyOffice") = "TC- " & Index
V   FacultyTable.Rows.Add(row)
W   Next Index

```

**Figure 3.18.** An example of creating a new table and adding data into the table.

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 NewRow method of the DataTable (refer to Table 3.20). In all, we create and add ten 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 shown in Table 3.22.

**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

### **3.5 CHAPTER SUMMARY**

The main topic of this chapter was an introduction to ADO.NET that included the architecture, organization, and components of ADO.NET.

The chapter provided detailed discussions and descriptions to give readers both fundamental and practical ideas and pictures of how to use components in ADO.NET to develop professional data-driven applications. Two ADO.NET architectures were 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 was introduced. Different data-related objects were discussed, such as Data Access Objects (DAO), Remote Data Objects (RDO), Open Database Connectivity (ODBC), OLE DB, and ADO. The difference between ADO and ADO.NET is discussed 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 the OLE DB Data provider and the ODBC Data Provider 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 OLE DB and ODBC can work for that kind of database since the former can provide a 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 the two ADO.NET architectures: you can use the DataAdapter to access and manipulate data in the DataSet that is considered a DataTables collector by executing 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 a different format and style depending on the database you are using. The popular components of the connection string include Provider, Data Source, Database, User ID, and Password. But some connection strings, such as for the Oracle Data Provider, only use a limited number of components.

An important element 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 of the Command object's properties.

Having finished this chapter, you should now be able to

- Understand the architecture and organization of ADO.NET
- Understand three components of ADO.NET, namely, the Data Provider, DataSet, and DataTable
- Use the Connection object to connect to Microsoft Access, Microsoft SQL Server, and Oracle databases
- 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 4, we will discuss the data query technique with two methods: using Tools and Wizards provided by Visual Basic 2005 and using the runtime object method to develop simple but efficient data query applications with three databases: Access, SQL Server, and Oracle. These methods are introduced in two parts: Part I discusses using the tools and wizards provided by Visual Basic.NET 2005 to develop a data query project, and Part II presents using the runtime objects to perform the data query for three databases.

## 3.6 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 Object Linking and Embedding (OLE) and Component Object Model (COM) technologies.
- \_\_\_\_\_3. ADO.NET is a new version of ADO and 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 different namespaces. The DataSet and DataTable are located at the System.Data namespace.

- \_\_\_\_\_ 7. The DataSet can be considered 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 can only be used to execute a read-out operation from a database.
- \_\_\_\_\_ 9. Both SQL Server and Oracle Data Providers use a so-called Named Parameter Mapping technique.
- \_\_\_\_\_ 10. The DataTable object is a Data Provider-independent object.

## **II. Multiple Choices**

- 1. To populate data from a database to a DataSet object, one needs to use the \_\_\_\_\_.
  - a. Data Source
  - b. DataAdapter (TableAdapter)
  - c. Runtime object
  - d. Wizards
- 2. The Parameters property of the Command class \_\_\_\_\_.
  - a. Is a Parameter object
  - b. Contains a collection of Parameter objects
  - c. Contains a Parameter object
  - d. Contains the parameters of the Command object
- 3. To add a Parameter object to the Parameters property of the Command object, one needs to use the \_\_\_\_\_ method that belongs to the \_\_\_\_\_.
  - a. Insert, Command
  - b. Add, Command
  - c. Insert, Parameters collection
  - d. Add, Parameters collection
- 4. The DataTable class is a container that holds the \_\_\_\_\_ and \_\_\_\_\_ objects.
  - a. DataTable, DataRelation
  - b. DataRow, DataColumn
  - c. DataRowCollection, DataColumnCollection
  - d. Row, Column
- 5. \_\_\_\_\_ is a property of the DataTable class, and it is also a collection of DataRow objects. Each DataRow can be mapped to a \_\_\_\_\_ in the DataTable.
  - a. Rows, column
  - b. Columns, column
  - c. Row, row
  - d. Rows, row

6. The \_\_\_\_\_ Data Provider can be used to execute the data query for the \_\_\_\_\_ 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, the \_\_\_\_\_ object is executed with suitable parameters.
  - a. DataAdapter
  - b. Connection
  - c. DataReader
  - d. Command
8. The DataReader is a read-only class and can only be used to retrieve and hold the data rows returned from a database when executing a(n) \_\_\_\_\_ method.
  - a. Fill
  - b. ExecuteNonQuery
  - c. ExecuteReader
  - d. ExecuteQuery
9. One needs to use the \_\_\_\_\_ method to release all objects used for a data-driven application before one can exit the project.
  - a. Release
  - b. Nothing
  - c. Clear
  - d. Dispose
10. To \_\_\_\_\_ data between the DataSet and the database, the \_\_\_\_\_ object should be used.
  - a. Bind, BindingSource
  - b. Add, TableAdapter
  - c. Move, TableAdapter
  - d. Remove, DataReader

### III. Exercises

1. Explain two architectures of 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, using an example, how to add Parameter objects to the Parameters collection that is a property of the Command object. Assume that an

SQL Server Data Provider is used with two parameters: parameter\_name: username, password; parameter\_value: “NoName,” “ComeBack.”

4. Explain the relationship between the DataSet and DataTable. Illustrate how to use the Fill method to populate a DataTable in the DataSet. Assume that the data query string is the SQL SELECT statement SELECT faculty\_id, name FROM Faculty, and that an SQL Server Data Provider is utilized.

---

# 4

---

## Data Selection Query with Visual Basic.NET

Starting from Visual Studio 2005, Visual Basic.NET adds new components and wizards to simplify the data access, inserting, and updating functionalities for database development and applications. Compared with Visual Basic.NET 2003, Visual Basic.NET 2005 greatly reduced the programming load and the amount of program code to provide significant assistance to people who are new to database programming with Visual Basic. Some of the most important components and wizards are

- Data Components in the Toolbox Window
- Wizards in the Data Source Window

The Toolbox window in Visual Basic.NET 2005 contains data components that enable you to quickly and easily build simple database applications without needing to address 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 a simple Visual Basic project to navigate, scan, retrieve, and manipulate data stored in the data source with a few of lines of code.

This chapter is divided into two parts. Part I provides a detailed description and discussion of how to use Visual Basic.NET 2005 tools and wizards to build simple but efficient database applications without complicated coding. In Part II, a deeper insight into how to develop advanced database applications using runtime objects is presented. More complicated coding techniques are provided in this part. Some real examples are provided in detail in these two parts to give readers a clear picture of simple and efficient ways to develop professional database applications. 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 the 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
- Build and execute simple dynamic data query commands to retrieve desired data

To successfully complete this chapter, you need to understand topics such as the fundamentals of databases, introduced in Chapter 2, and ADO.NET, discussed in Chapter 3. Also, the sample database developed in Chapter 2 will be used through this chapter.

## PART I DATA QUERY WITH VISUAL BASIC.NET DESIGN TOOLS AND WIZARDS

Before we can start the next section, a preview of a completed sample database application is necessary. This preview will give readers an idea of how a database application works and what it can do. The database used for this project is Access.

### 4.1 A COMPLETED SAMPLE DATABASE APPLICATION EXAMPLE

This sample application is composed of five forms, named LogIn, Selection, Faculty, Student, and Course. This example is designed to map a Computer Science and Engineering (CSE) Department in a university and to allow users to scan and browse all information about the department, including faculty members, courses taught by selected faculty members, students, and courses taken by the students.

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 relationships between the forms and tables are shown in Table 4.1.

Controls on each form are bound to the associated fields in certain data tables located in the CSE\_DEPT database. As the project runs, a data query will be executed via a dynamic SQL statement that is built during the configuration of each TableAdapter in the Data Source wizard. The retrieved data will be reflected in the associated controls that have been bound to those data fields.

**Table 4.1. Relationships Between the Forms and Data Tables**

VB Form	Tables in Sample Database
LogIn	LogIn
Faculty	Faculty
Course	Course
Student	Student, StudentCourse



Figure 4.1. The LogIn form.

Go to the Web site [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) to find the folder **DBProjects\Chapter 4\SampleWizards Solution\Sample Wizards Project** and locate the executable file SampleWizards Project.exe. You can locate and run this project in two ways:

1. Double-click the executable file SampleWizards Project.exe to run it.
2. Copy this project and paste it in your local drive, such as C:, and then double-click this project to run it.

As the project runs, a login form will be displayed to ask users to enter a user-name and password, which is shown in Figure 4.1. Enter ybai and reback as the user-name and password. Then click the LogIn button to call the LogIn TableAdapter to execute a query to pick up a record that matches the username and password entered by the user from the LogIn table located in the CSE\_DEPT database.

If a matched record is found based on the username and password, this means that the login is successful, and the next window form, Selection, will be displayed to allow the user to select desired information, such as faculty, course, or student, as shown in Figure 4.2.

Select the default information, Faculty Information, by clicking the OK button. The Faculty form appears as shown in Figure 4.3.

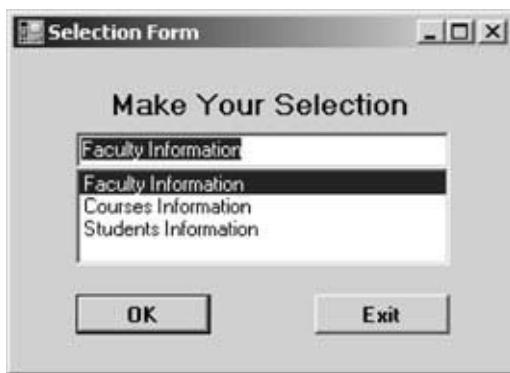


Figure 4.2. The Selection form.



Figure 4.3. The Faculty form.

All faculty names in the CSE department are listed in a ComboBox control on the form. Select the desired faculty from the ComboBox control by clicking the drop-down arrow, and click the name selected. To query all information for this faculty member, click the Select button to execute a prebuilt dynamic SQL statement. All information on the selected faculty member, which is stored in the Faculty table in the database, will be fetched from the database and reflected in five label controls in the Faculty form, as shown in Figure 4.3. Also, the faculty photo will 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 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 member, which will be displayed in the Course List box, as shown in Figure 4.4.

An interesting thing is that when you select the specified course by clicking 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 in each associated text box control under the Course Information frame control.

Click the Back button to return to the Selection form, and select Student Information to open the Student form. You can continue to work on this form to see what will happen to it.

In the following sections, you will learn how to design and build this demo project step by step by using SQL Server 2005. It is very easy to develop a project similar to this one using different databases, such as 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 desire.

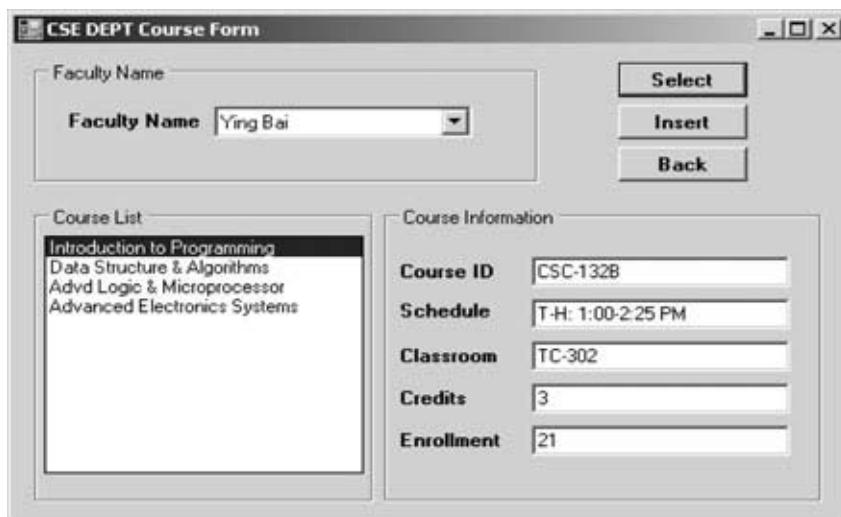


Figure 4.4. The Course form.

## 4.2 VISUAL BASIC.NET 2005 DESIGN TOOLS AND WIZARDS

When developing and building a Windows application that needs to interface with a database, a powerful and simple way is to use tools and wizards called design tools provided by Visual Basic.NET 2005. The length of the coding process can be significantly reduced, and the development procedures can also be greatly simplified. Now let's first take a look at the components residing in the Toolbox window.

### 4.2.1 Data Components in the Toolbox Window

Each database-related Windows application contains three components that can be used to develop a database application using the data controls in the Toolbox: DataSet, BindingSource, and TableAdapter. Two other useful components are the DataGridView and the BindingNavigator. All these components are located in the Toolbox window, as shown in Figure 4.5.



Figure 4.5. Data components in the Toolbox window.

Compared with Visual Studio 2003, in which three components, DataConnection, DataAdapter, and DataSet, are used to perform data operations for a Visual Basic application, Visual Basic.NET 2005 adds some new components.

#### 4.2.1.1 DataSet

A DataSet object can be considered as a container, and it is used to hold data from one or more data tables. It maintains the data as a group of data tables with optional relationships defined between those tables. The definition of the DataSet class is a generic idea, which means that it is not tied to any specific type of database. Data can be loaded into a DataSet by using a TableAdapter from many different databases, such as Microsoft Access, Microsoft SQL Server, Oracle, Microsoft Exchange, Microsoft Active Directory, or any OLE DB or ODBC compliant database when your application begins to run, or the Form\_Load() event procedure is called if one uses a DataGridView object.

Although not tied to any specific database, the DataSet class is designed to contain relational tabular data such 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 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 two 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 a mini database engine: it can contain information on the tables it holds, such as the column name and data type, all relationships between tables, and, more important, most management functionalities of the tables, such as the ability to browse, select, insert, update, and delete data from the tables.

A DataSet is a container. 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 edit an XML schema or .xsd file. Although there is a visual designer, the terminology and user interface are not consistent with a DataSet and its constituent objects.

With the Visual Basic.NET 2005, 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).

In summary, the DataSet object is a very powerful component that can contain multiple data tables with all the 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 from your database by using data query methods or the Fill method. From this point of view, you can consider the DataSet as a *data source* that contains all mapped data from the database you connect to your project.

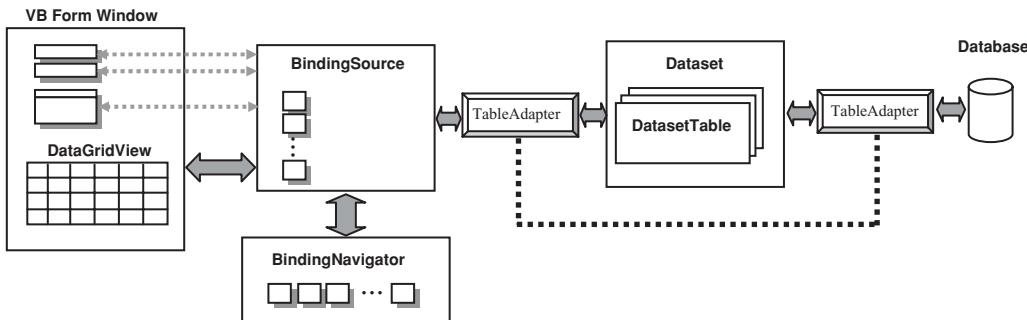


Figure 4.6. The relationship between data components.

Refer to Figure 4.6 for a global picture of the DataSet and other components in the Toolbox window to obtain more detailed ideas of these components.

#### 4.2.1.2 DataGridView

The next useful data component defined in the Toolbox window is DataGridView.

As its name suggests, you can consider the DataGridView as a view container, and it can be used to bind data from your database and display it in a tabular or 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, the DataGridView control

- 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 in Windows XP and the Windows Server 2003 family when the EnableVisualStyles method is called from the application's Main method

Refer to Figure 4.6 to see the relationship between DataGridView and other data components. A more detailed description of how to use the DataGridView control to bind and display data in Visual Basic.NET 2005 will be provided in Section 4.5 of this chapter.

#### 4.2.1.3 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 the data source, and then binding the controls on the 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 that holds all mapped data from the data source. As a data-driven project runs, the DataSet will be filled with data from the database by using a TableAdapter. Also, the BindingSource control will create a set of data that is mapped to the 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, or more precisely, when you set up a connection between your controls on the Visual Basic form and the mapped data in the BindingSource object. As your project runs and the data need to be reflected in the associated controls, a request to BindingSource is issued and the BindingSource control will control the access 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 data binding in Visual Basic.NET 2005, you need to bind the data referenced by the BindingSource control to the DataSource property of your controls on the forms.

#### **4.2.1.4 BindingNavigator**

The BindingNavigator control allows user to scan and browse all records stored in the data source (DataSet) one by one in a sequence. The BindingNavigator component provides a standard user interface (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 text box controls to display how many records exist in the current data table and the current displayed record's index.

As shown in Figure 4.6, the BindingNavigator is also bound to the BindingSource component as the other components are. When the user clicks either the Previous or the Next button on the BindingNavigator UI, a request is sent to the BindingSource for the previous or the next record, and in turn, this request is sent to the data source to pick up the desired data.

#### **4.2.1.5 TableAdapter**

From Figure 4.6, one can see that a TableAdapter is equivalent to an adapter and just works as a connection medium between the database and the DataSet, and between the BindingSource and the DataSet. This means that the TableAdapter has double functionalities – it works in different roles for 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 data operations between the BindingSource and the DataSet when the data-bound controls in a 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 was needed to the data in the DataSet, you had 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 is new to Visual Basic.NET 2005, and you cannot find this component in the Toolbox window. The TableAdapter belongs to designer-generated components that connect your DataSet objects with their underlying databases, and it is created automatically when you add and configure new data sources via design tools such as Data Source Configuration Wizard.

The TableAdapter is similar to DataAdapter in that both components can handle 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 DataAdapter is that each TableAdapter is a unique class that is automatically generated by Visual Studio 2005 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. It also 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.

### 4.2.2 Data Sources Window

Starting with Visual Studio 2005, two new Integrated Development Environment (IDE) features, the Data Sources Window and the Data Source Configuration Wizard, are added to assist you 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 before dragging it onto a form. You can also customize the control list with additional controls, such as ones that you have created.

A more detailed description of how to use the Data Sources window to develop a data-driven project is provided in Section 4.4.

#### 4.2.2.1 Add New Data Source

When you create a new data-driven project in Visual Basic.NET 2005 environment for the first time, there is no data source connected to your project, and therefore the

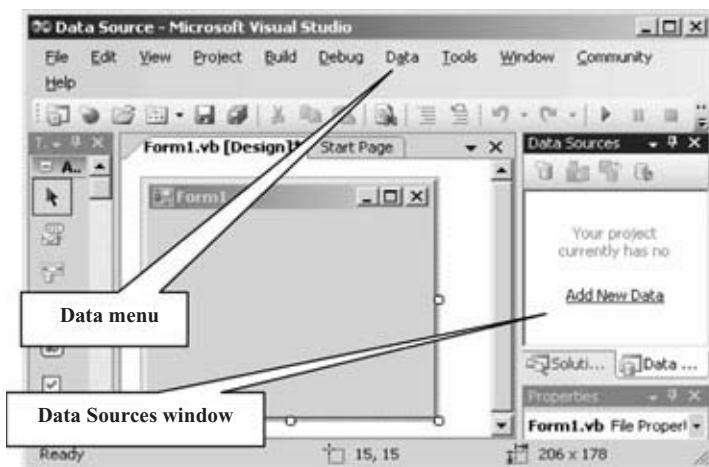


Figure 4.7. The Data Sources window.

Data Sources window is a blank window with no data source in it. For example, you can create a new Visual Basic.NET 2005 Windows application by selecting File|New Project menu items and Data Source as the project name. After this new project is created and opened, you can find the Data Sources window by clicking the Data menu in the menu bar, which is shown in Figure 4.7.

To open the Data Sources window, click the Data>Show Data Sources item. Because you have no previous database connected to this new project, the open Data Sources window is empty. To add a new data source or database to this new project, click Add New Data in the Data Sources window.

Once you click Add New Data in the Data Sources window to add a new data source, the Data Source Configuration Wizard will be displayed. You need to use this wizard to select the desired database to be connected with your new project.

#### **4.2.2.2 Data Source Configuration Wizard**

The opened Data Source Configuration Wizard is shown in Figure 4.8.

By using the Data Source Configuration Wizard, you can select the desired data source or database that will be connected to your new project. The Data Source Configuration Wizard supports four types of data sources. The first option, Database, allows you to select a data source from a database server on your local computer or on a remote server. Examples of this kind of data source are SQL Server 2005 Express, Microsoft Data Engine (MSDE) 2000, or SQL Server 2005. This option also allows you to choose either an SQL Server .mdf database file or a Microsoft Access .mdb 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. The second option, Web Service, enables you to select a data source that is located at a Web service. The third option, Object, allows you to bind your user interface to one of your own database classes.



**Figure 4.8.** The Data Source Configuration Wizard.

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 4.9.

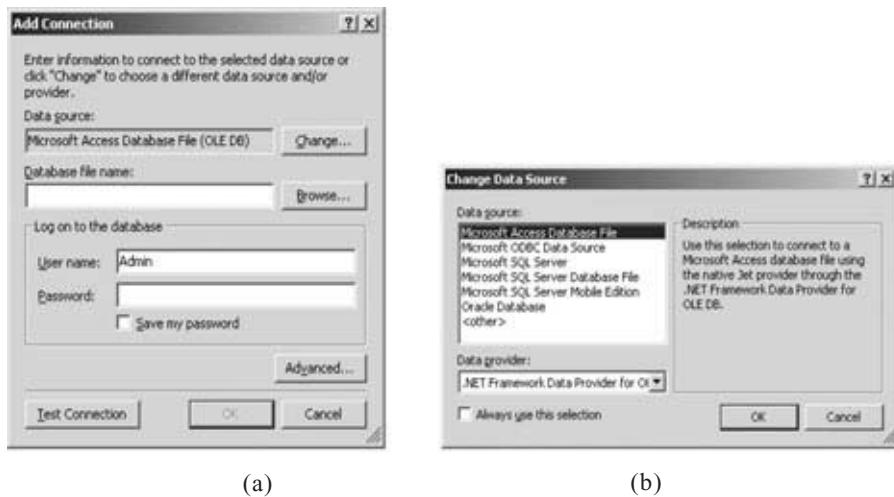
The first time you run this wizard, there are no preexisting connections available, but on subsequent uses of the wizard you can reuse previously created connections. To make a new connection, click the New Connection button. The Add Connection dialog box is displayed, which is shown in Figure 4.10a.

You can select different data source types by clicking the Change button. The Change Data Source dialog box is displayed, as shown in Figure 4.10b.

Six popular data sources can be chosen on the basis of your application:



**Figure 4.9.** The next step in the Data Source Configuration Wizard.



**Figure 4.10.** The Add Connection and Change Data Source dialog boxes.

1. Microsoft Access Database File
2. Microsoft ODBC Data Source
3. Microsoft SQL Server
4. Microsoft SQL Server Database File
5. Microsoft SQL Server Mobile Edition
6. Oracle Database

The second choice allows users to select any kind of data source that is compatible with a Microsoft ODBC data source. The fourth choice is for users who select a SQL Server 2005 Express data source. For example, you want to connect your new project with a Microsoft Access database named CSE\_DEPT.mdb. You need to keep the default data source as Microsoft Access Database File, and click the Browse button to locate and select that file. You can click the Test Connection button to test your connection. A “Test connection succeeded” message will be displayed if your connection is correct, as shown in Figure 4.11.

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 2005 project. You can save this connection string for future use if you will want to use the same connection again later for your application.

When you click the Next button, a message box will be displayed to ask you if you want to save this data source into your new project, as shown in Figure 4.12. 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 will not have to worry about connection problems between your project and your data source, and they will be one body and easily portable. The disadvantage is that the size of your project increases and more memory space is needed to save your application.

The next configuration step, which is shown in Figure 4.13, allows you to select the database objects for this data source. Although you can select any number of

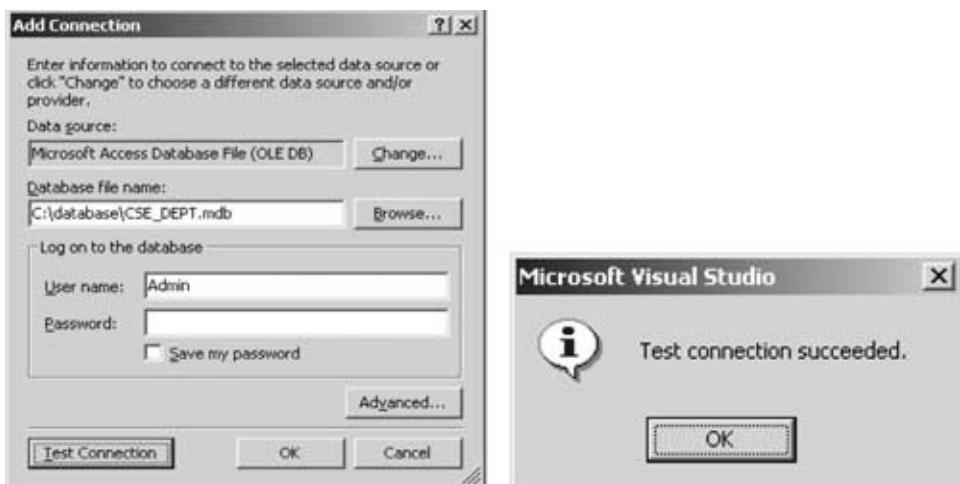


Figure 4.11. The Add Connection Dialog and Test messagebox.

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 the new instance of your DataSet class; in this example it is CSE\_DEPTDataSet, which can be seen in the DataSet name box shown in Figure 4.13. The data in all tables in your database (CSE\_DEPT.mdb) should have been copied to the 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 point 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 actions between your application and your database later.

After the Data Source Configuration is finished, a new data source is added into your project, that is, into the Data Sources window, which is shown in Figure 4.14.

The data source added into your project is actually a DataSet object that contains data tables that are mapped to the tables in your real database. As shown in Figure 4.14, the data source window displays the data source or tables as a tree view, and

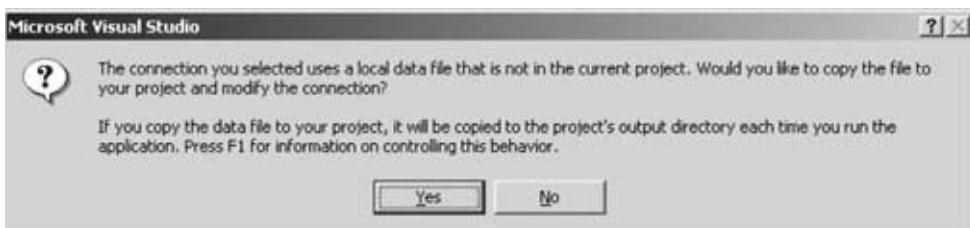


Figure 4.12. A message to ask you to save the data source.



Figure 4.13. Select database objects in the configuration wizard.

each table is connected to this tree via a node. If you click a node (represented by a small “plus” symbol surrounded with a box), all columns of the selected table will be displayed.

Even after the data source is added into your project, the story is not finished and you still have some control over this data source. This means that you can still make some modifications to the data source, or, to be precise, 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 Sources window.



Figure 4.14. The added data source.

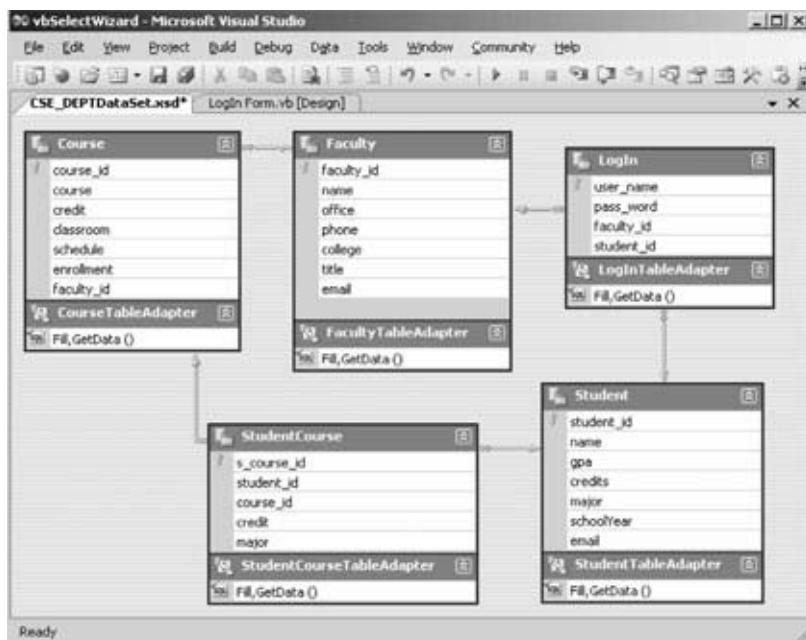


Figure 4.15. A sample DataSet Designer.

#### 4.2.2.3 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.

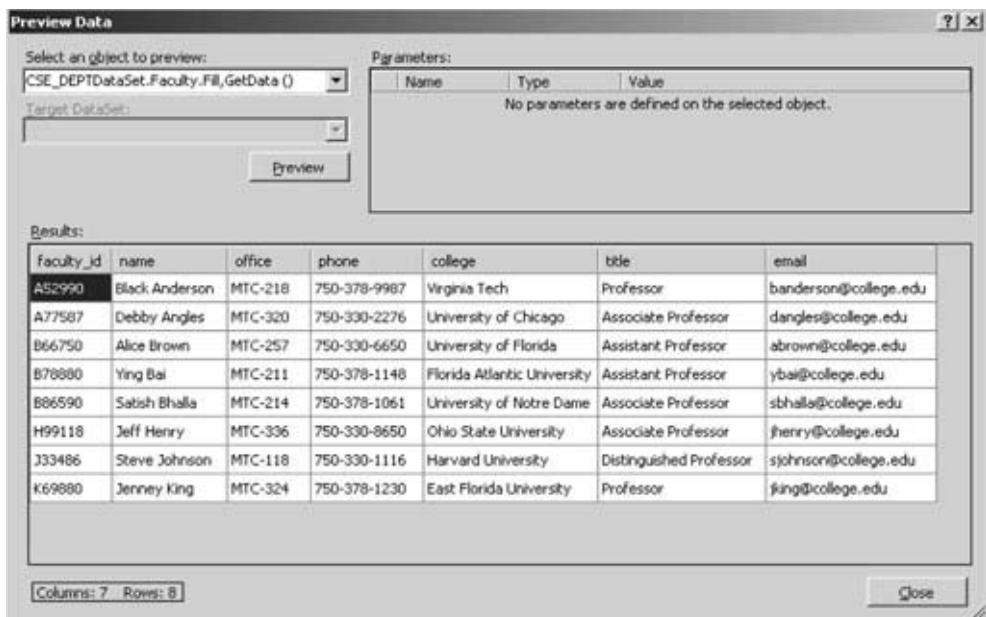
The DataSet Designer provides visual representations of the objects contained in the DataSet. By using the DataSet Designer, you can create and modify TableAdapters, TableAdapter Queries, DataTables, DataColumns, and DataRelations.

To open the DataSet Designer, right-click anywhere inside the Data Sources window, then select Edit DataSet with Designer. A sample DataSet Designer is shown in Figure 4.15.

In this sample database, we have five tables: LogIn, Faculty, Course, Student, and StudentCourse. To edit any item, just right-click the component that you want to modify. For example, if you want to edit the LogIn table, right-click that table. A popup window will be displayed with multiple editing selections. You can add new queries, new relationships, new keys, and even new columns to the LogIn table. Also, you can modify or edit any built-in method of the TableAdapter, namely, the LogInTableAdapter in this example.

In addition to the multiple editing abilities mentioned above, you can perform the following popular data operations using the DataSet Designer:

- **Configure:** configure and build data operations such as building a data query by modifying the default methods of the TableAdapter such as Fill() and GetData()
- **Delete:** delete the whole table



**Figure 4.16.** An example of Preview Data.

- Rename: rename the table
- Preview Data: view the contents of the table in a grid format

Preview Data is a very powerful tool, and it allows users to preview the contents of a data table. Figure 4.16 shows an example of Preview Data using the – Faculty table.

On the basis of the above discussions, it can be seen that the DataSet Designer is a powerful tool to help users design and manipulate a data source or DataSet even if the data source has been added into the project. However, that is not all! The DataSet Designer has another important functionality, which is to allow users to add a missing table to the project.

Of course, you should not have this kind of problem in a normal situation as you develop a data-driven application using the DataSet Designer. But in some cases, if you have forgotten to add a data table or added a wrong table (in my experience, this happens a lot for students who select the wrong data source), you need to use this functionality to add that missed table or delete the wrong table and add the correct one.

To add a missed table, just right-click a blank area of the designer surface and choose Add|DataTable. You can also use this functionality to add a TableAdapter, Query, or Relation to this DataSet.

A more detailed exploration of DataSet Designer will be provided in Section 4.6.

### 4.3 BUILD A SAMPLE DATABASE PROJECT – SELECTWIZARD

So far we have seen most of the design tools located in the Visual Basic.NET 2005 Toolbox and Data Sources window. In the following sections, we will illustrate how

**Table 4.2. Relationships Between the Forms and Data Tables**

VB Form	Tables in Sample Database
LogIn	LogIn
Faculty	Faculty
Course	Course
Student	Student, StudentCourse

to utilize those tools to build a data-driven application by using a real example. First, let's build a Visual Basic.NET 2005 project named SelectWizard, which means that we want to build a project with design tools provided in the Toolbox window and the Data Sources window.

### 4.3.1 Application User Interface

We made a demo for this sample data-driven application in Section 4.1. This project is composed of five forms: LogIn, Selection, Faculty, Student, and Course. The project is designed to map a Computer Science and Engineering Department in a university and to allow users to scan and browse all information about the department, including faculty members, courses taught by selected faculty members, students, and courses taken by the students.

Each form, except the Selection form, is associated with one or two data tables in the sample database CSE\_DEPT, which was developed in Chapter 2. The relationships between the forms and tables are shown in Table 4.2.

Controls on each form are bound to the associated fields in a 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 in 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 an SQL Server 2005 Express database since it is compatible with the SQL Server 2005 database and, most important, it is free and can be easily downloaded from the Microsoft Knowledge Base site. You can use any kind of database, such as Microsoft Access, SQL Server 2005, or Oracle, for this sample project. The only thing you need to do is to select the desired data source when you add and connect that data source to your project.

Let's begin to develop this sample project with five forms.

#### 4.3.1.1 The LogIn Form

First, create a new Visual Basic.NET 2005 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 2005 and go to File|New Project to open the New Project window, make sure that the Windows Application icon is selected from the



Figure 4.17. A new project window.

Templates window, enter SelectWizard into the Name text box as the project's name, and click OK to continue.

When the new project is created and the default windows form is opened, which is shown in Figure 4.17, make the following modifications:

- 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 controls shown in Table 4.3 into the LogIn form. You should select the LogIn button, cmdLogIn, as the default button by choosing this button from the AcceptButton property. Also, you need to select CenterScreen from the StartPosition property of the form. The finished LogIn form should match the one shown in Figure 4.18.

**Table 4.3. Controls for the LogIn Form**

Type	Name	Text	TabIndex
Label	Label1	Welcome to CSE Department	0
Label	Label2	UserName	1
TextBox	txtUserName		2
Label	Label3	Pass Word	3
TextBox	txtPassWord		4
Button	cmdLogIn	LogIn	5
Button	cmdCancel	Cancel	6



Figure 4.18. The LogIn form.

#### 4.3.1.2 The Selection Form

This form allows users to select different windows forms to connect to different data tables, and furthermore to browse data from the associated tables. No data table is connected to this form.

Go to the Project|Add Windows Form menu item to add a new form with the file name Selection Form.vb. The objects in Table 4.4 need to be added into this form.

You should select the OK button, cmdOK, as the default button by choosing this button from the AcceptButton property of the form. Also, you need to select CenterScreen from the StartPosition property of the form.

The completed Selection form should match the one shown in Figure 4.19.

#### 4.3.1.3 The Faculty Form

The Faculty form contains controls that are related to faculty information stored in the database CSE\_DEPT, which is the sample database built in Chapter 2.

Go to the Project|Add Windows Form menu item to add a new form with the file name Faculty Form.vb. The objects in Table 4.5 need to be added into this form.

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 CenterScreen from the StartPosition property of the form.

The finished Faculty form should match the one shown in Figure 4.20.

In this chapter, we will use only the Select button to make data queries to the data source. Other buttons will be used in the following chapters.

**Table 4.4. 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		

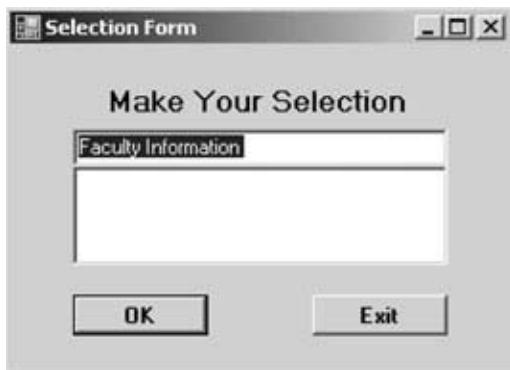


Figure 4.19. The Selection form.

#### 4.3.1.4 The Course Form

This form is used to interface the Course table in your data source to retrieve course information associated with a specific faculty member selected by the user. Recall that in Chapter 2, we developed the sample database CSE\_DEPT, and the Course table is one of five tables in that database. A one-to-many relationship exists between the Faculty and the Course tables, which is created 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 the file name Course Form.vb. The objects in Table 4.6 need to be added into this form. The finished Course form should match the one that is shown in Figure 4.21.

In this chapter, we will use only the Select button to make data queries to the data source. The Insert button will be used in Chapter 5.

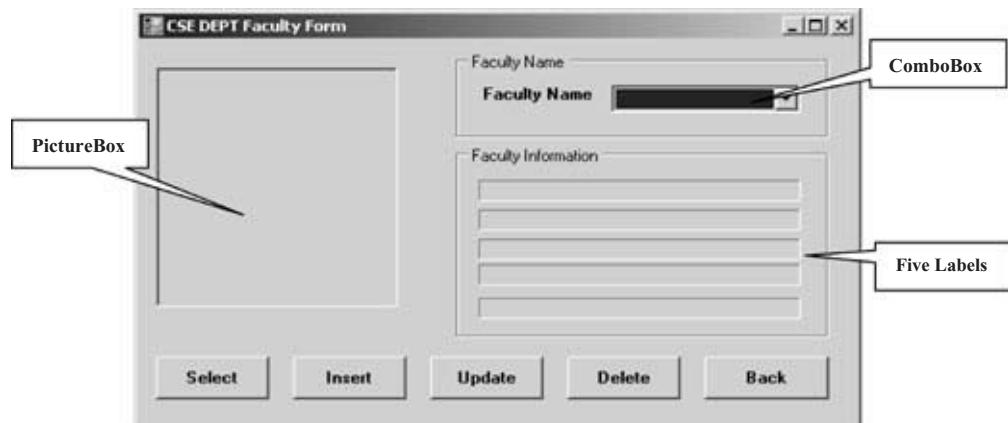


Figure 4.20. The Faculty form.

**Table 4.5.** Objects for the Faculty Form

Type	Name	Text	TabIndex	DropDownStyle
PictureBox	PhotoBox			
GroupBox	FacultyBox	Faculty Name	0	
Label	Label1	Faculty Name	0.0	
ComboBox	ComboName		0.1	DropDownList
GroupBox	FacultyInfoBox	Faculty Information	1	
Label	TitleLabel		1.0	
Label	OfficeLabel		1.1	
Label	PhoneLabel		1.2	
Label	CollegeLabel		1.3	
Label	EmailLabel		1.4	
Button	cmdSelect	Select	2	
Button	cmdInsert	Insert	3	
Button	cmdUpdate	Update	4	
Button	cmdDelete	Delete	5	
Button	cmdBack	Back	6	
Form	FacultyForm	CSE DEPT Faculty Form		

#### 4.3.1.5 The Student Form

The Student form is used to collect and display student information, including the courses taken by the student. As mentioned in Section 4.1, the Student form needs two data tables in the database: the Student table and the StudentCourse table. This is a typical example of using two data tables for one graphical user interface (form).

Go to the Project|Add Windows Form menu item to add a new window form with the file name Student Form.vb. The objects in Table 4.7 need to be added into this form.

Set the following properties for controls and objects in this form:

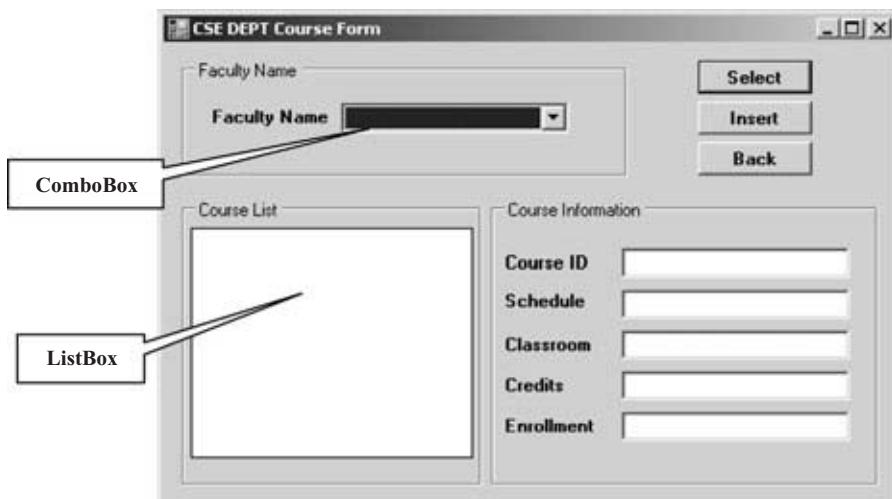


Figure 4.21. The Course form.

**Table 4.6.** Objects for the Course Form

Type	Name	Text	TabIndex	DropDownStyle
GroupBox	NameBox	Faculty Name	0	
Label	Label1	Faculty Name	0.0	
ComboBox	ComboName		0.1	DropDownList
GroupBox	CourseBox	Course List	1	
ListBox	CourseList		1.0	
GroupBox	CourseInfoBox	Course Information	2	
Label	CourseIDLabel	Course ID	2.0	
TextBox	txtTitle		2.1	
Label	ScheduleLabel	Schedule	2.2	
TextBox	txtSchedule		2.3	
Label	ClassRoomLabel	Classroom	2.4	
TextBox	txtClassRoom		2.5	
Label	CreditsLabel	Credits	2.6	
TextBox	txtCredits		2.7	
Label	EnrollLabel	Enrollment	2.8	
TextBox	txtEnroll		2.9	
Button	cmdSelect	Select	3	
Button	cmdInsert	Insert	4	
Button	cmdBack	Back	5	
Form	CourseForm	CSE DEPT Course Form		

- Make the Select button the default button by selecting this button from the AcceptButton property of the form
- Select CenterScreen from the StartPosition property of the form
- Set the BorderStyle property of the CourseList ListBox control to FixedSingle

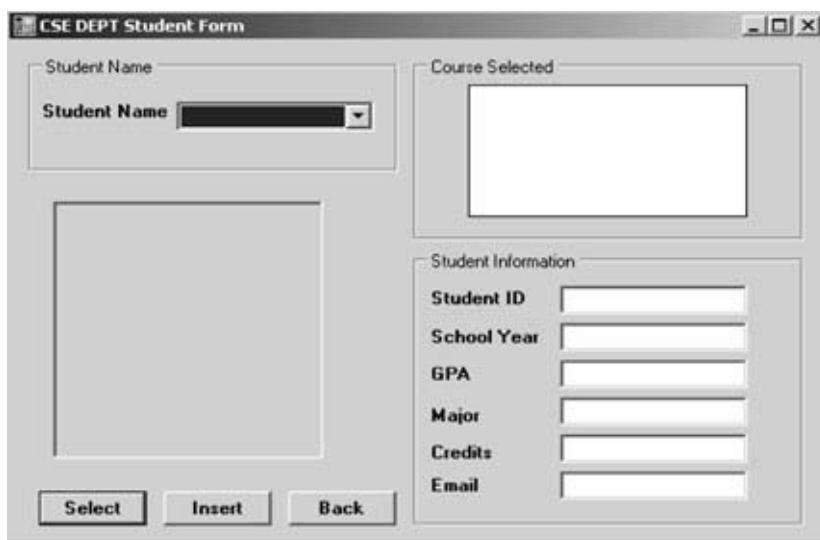


Figure 4.22. The Student form.

**Table 4.7.** Objects for the Student Form

Type	Name	Text	TabIndex	DropDownStyle
PictureBox	PhotoBox			
GroupBox	StudentNameBox	Student Name	0	
Label	Label1	Student Name	0.0	
ComboBox	ComboName		0.1	DropDownList
GroupBox	CourseSelectedBox	Course Selected	1	
ListBox	CourseList		1.0	
GroupBox	StudentInfoBox	Student Information	2	
Label	Label2	Student ID	2.0	
TextBox	txtID		2.1	
Label	Label3	School Year	2.2	
TextBox	txtSchoolYear		2.3	
Label	Label4	GPA	2.4	
TextBox	txtGPA		2.5	
Label	Label5	Major	2.6	
TextBox	txtStatus		2.7	
Label	Label6	Credits	2.8	
TextBox	txtCredits		2.9	
Label	Label7	Email	2.10	
TextBox	txtEmail		2.11	
Button	cmdSelect	Select	3	
Button	cmdInsert	Insert	4	
Button	cmdBack	Back	5	
Form	StudentForm	CSE DEPT Student Form		

Also, in this Student form, we use TextBoxes to replace Labels for binding and displaying the student information. The courses taken by the student are reflected and displayed in a ListBox control, CourseList.

Your finished Student form should match the one shown in Figure 4.22.

## 4.4 ADD AND UTILIZE VISUAL BASIC WIZARDS AND DESIGN TOOLS

After the graphical user interface (GUI) is completed, we need to add a data source to this new project and set up a connection between the project and the database. In Section 4.2.2, we discussed in detail how to add a new data source and configure it. In what follows, we illustrate this with a real Visual Basic.NET 2005 project, SelectWizard, which we created in the last section.

### 4.4.1 Add and Configure a New Data Source

Open the project SelectWizard and select the LogIn form window.

Go to the Data>Show Data Sources menu item to open the Data Sources window. Currently this window is blank since we have not added and connected any data source to this project. Click the link Add New Data Source as we did in Section 4.2.2.1 to add a new data source to your project.

In the opened Data Source Configuration Wizard, keep the default selection Database and click Next to go the next step, which is shown in Figure 4.23a.

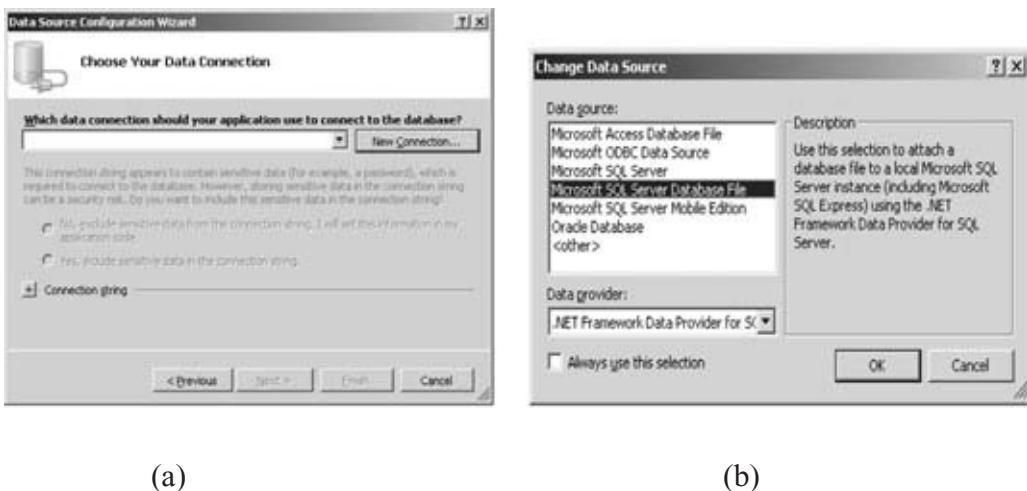


Figure 4.23. The Change Data Source dialog.

Click the New Connection button to open the Add Connection dialog. This dialog allows you to select your database with two specifications: database type and database name, which is determined by the actual database you want to use for this project. As mentioned before, we want to use an SQL Server Express database for this project, so we first need to change the default database type from Microsoft Access Database File to Microsoft SQL Server Database File by clicking the Change button that is located next to the Data Source box.

In the opened Change Data Source dialog box, select Microsoft SQL Server Database File, as is shown in Figure 4.23b. Click OK to select this database type and return to the Add Connection dialog box.



Microsoft SQL Server is different from Microsoft SQL Server Database File. The former is used for the real server/client database, but the latter is used for a local SQL Server instance, such as SQL Server Express. This means that you can install both an SQL Server and an SQL Client on your local computer by using SQL Server Express 2005. Your computer works as both a server and a client, and you access the SQL Server database file from that local server.

Next, we need to select the database file. Click the Browse button to locate your database file. You should have already developed and built your database files using Microsoft SQL Server 2005 Express in Chapter 2, and this database file should be located at **C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data** in your computer. In this example, the database name is CSE\_DEPT.mdf, with five data tables. If you have not developed this database file, you can get it from the Web site [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **database\SQLServer**. You can copy this database file to the folder listed above in your computer.

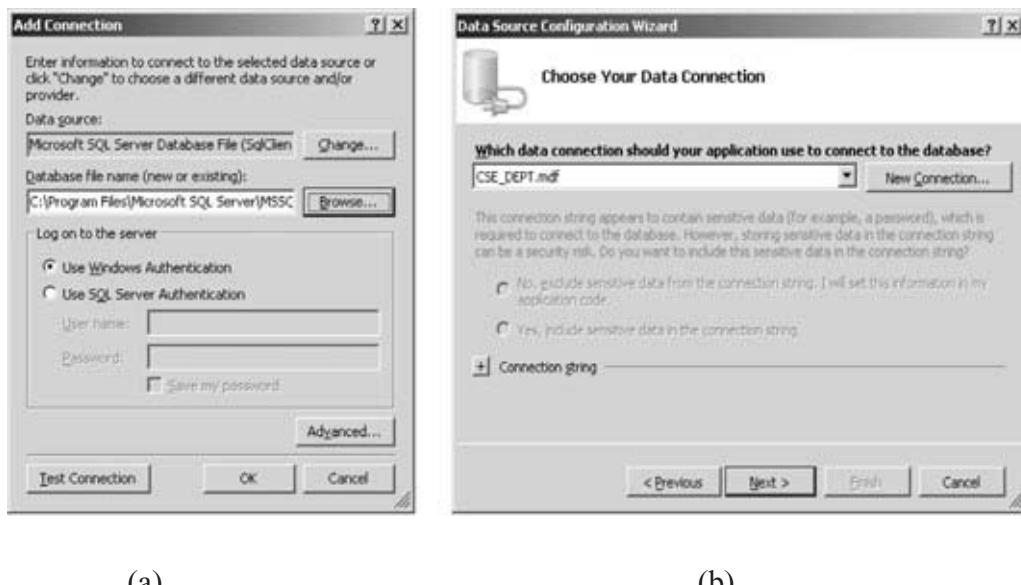


Figure 4.24. The Add Connection dialog box and Data Source Configuration Wizard.

Select this database file and click the Open button to add it into your project. The completed Add Connection dialog box is shown in Figure 4.24a.

You can test this connection by clicking the Test Connection button to confirm that the connection to your database works. For the login security, we use the default Windows Authentication mode. If you like, you can use the SQL-specific username and password by selecting Use SQL Server Authentication. Click the OK button to return to the Data Source Configuration Wizard, which is shown in Figure 4.24b. Click Next to go to the next step.

A message box will pop up to ask if you would like to add this data source into your project. As discussed in Section 4.2.2.2, click Yes to save the data source in your project.

The next window shows a message to ask if you would like to save this connection string for future use. Select the check box to save it, and click the Next button to continue.

The next step allows you to select different database objects. Generally all data tables must be selected because we need to use them to perform data operations between the Visual Basic.NET 2005 project and the tables in the connected database. The View object provides users with a view of tables and allows users to open and scan all data using the Preview Data functionality. The other objects are not necessary for developing simple data-driven applications. Stored Procedures are used to combine a sequence of queries to form a procedure to speed up the data query operations. Functions objects provide special functions to facilitate the building of data-driven applications. No damage will be done to our project if we select all of them. So just check the boxes to select all of them, as shown in Figure 4.25.



Figure 4.25. Select the database objects.

You will find that a new DataSet with the name CSE\_DEPTDataSet is created and is located in the DataSet name: box. Click the Finish button to complete this configuration.

After you finish this Data Source Configuration, a new instance of the DataSet with the name CSE\_DEPTDataSet is added into your project, as shown in Figure 4.26. Five data tables, LogIn, Faculty, Course, Student, and StudentCourse, are included in this DataSet instance. These five data tables are only mappings or copies

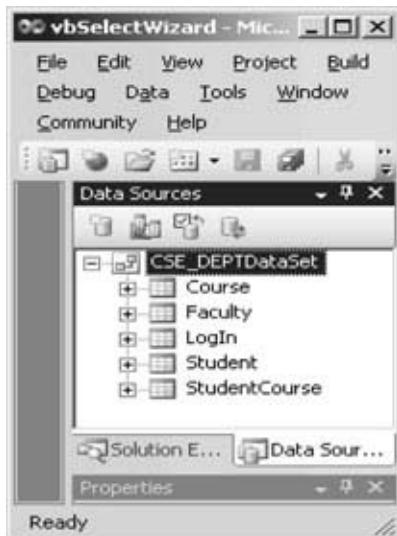


Figure 4.26. The new data source CSE\_DEPTDataSet.

of the real tables in the database. The connection you set up between your project and your database is closed as this wizard is finished. You will need to call data query or manipulation methods to reopen this connection as you perform data queries or actions later in your application.

## 4.5 QUERY AND DISPLAY DATA USING THE DATAGRIDVIEW CONTROL

Now we have added a data source into your Visual Basic.NET 2005 project. Before we develop this data-driven project, we will look at a popular and important functionality provided by the Toolbox window: DataGridView. As discussed in Section 4.2.1.2, DataGridView is a view container that 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, add a new blank form to the project SelectWizard, and name this new form Grid Form. To do this, go to Project|Add Windows Form to open the Add New Item dialog, enter Grid Form.vb into the Name box, and click the Add button.

Select the newly added form Grid Form, and open the Data Sources window (if it is not open) by clicking the Data menu item from the menu bar. You can view data of any table in your Data Sources window. Two popular views are Full Table view and Detail view for specified columns.

Here we use the Faculty table to illustrate how to use these two views.

### 4.5.1 View the Entire Table

To view the full Faculty table, click the Faculty table from the Data Sources window, click the drop-down arrow, and select the DataGridView item. Then drag the Faculty table to the Grid Form window, which is shown in Figure 4.27.

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 and the Add, Delete, and Save buttons. This set of components helps you to view data from the selected table. To see a full table view, make sure to set two properties of DataGridView, AutoSizeColumnsMode and AutoSizeRowsMode, to AllCells. In this way, you can have all data displayed on this grid view tool.

Now you can run your project by clicking the Start button. But wait a moment! One more thing to do before you run your project is to check whether you have selected the Grid Form as the startup object. To do that, go to Project|SelectWizard Properties, and in the opened window, select the Grid Form from the Startup form box. Now run your project, and you will find that the entire Faculty table is shown in this grid view tool, as shown in Figure 4.28.

By using this grid view tool, you can not only view data from the Faculty table but also add new data into and delete data from the table by clicking the Add button (which looks like a plus symbol) or the Delete button (to the right of the Add button). Just type the new data in the new line after you click the Add button if you

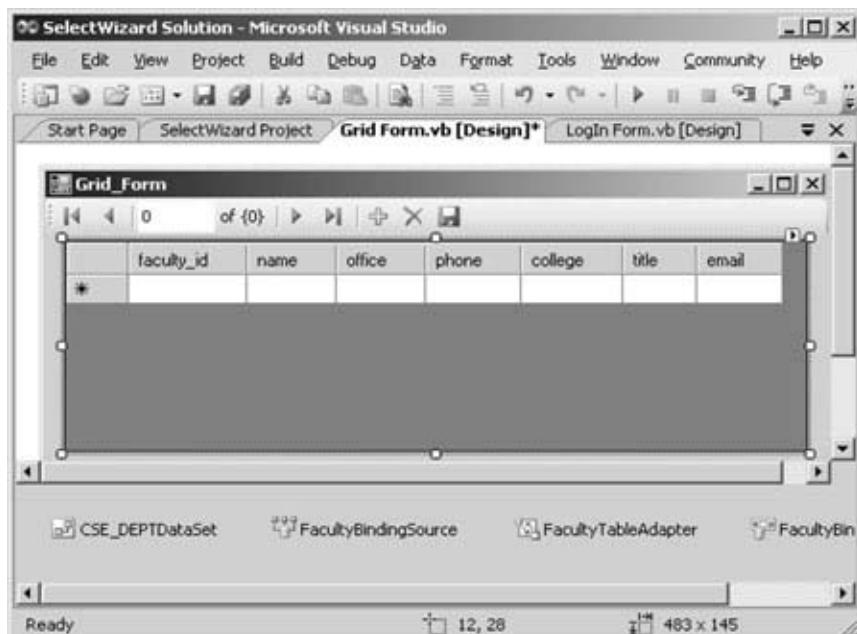


Figure 4.27. The DataGridView tool.

want to add new data, or navigate 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 affect only data in your data tables in the DataSet; they have 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 happens behind this dragging? Let's look a little deeper into 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 mentioned in sections 4.2.1.1–4.2.1.5, those components are objects or instances that are created based on their associated

faculty_id	name	office	phone	college	title	email
A52990	Black Anderson	MTC-218	750-379-9987	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	Assistant Professor	ybai@college.edu
B96590	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	750-379-1230	East Florida University	Professor	jking@college.edu
**						

Figure 4.28. The entire table view for the Faculty table.

classes such as the BindingSource, BindingNavigator, and TableAdapter as you drag the Faculty table into your form window.

Second, let's look at the situation occurring with the program code that is related to those objects and is created automatically by the system when new objects or instances are created as the dragging occurs. Open the Solution Explorer window and select Grid Form.vb, then click the View Code button to open the code window. Browse to the Grid\_Form\_Load() event procedure, and you can find this line of code in there:

---

```
Me.FacultyTableAdapter.Fill(Me.CSE_DEPTDataSet.Faculty)
```

---

It looks as if the Fill() method, which belongs to the FacultyTableAdapter, is called to load data from the database into your DataGridView tool. The Fill() method is very powerful, and it performs an operation equivalent to the SQL SELECT statement. To clarify this process further, open the Data Sources window, and right-click anywhere inside that window. Select the Edit the DataSet with Designer item to open the DataSet Designer Wizard. Right-click 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, 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 fill the grid view tool in the Faculty form.

#### 4.5.2 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 Sources window and click the Faculty table. Click the drop-down arrow and select the Detail item. Drag the Faculty table from the Data Sources window to the Faculty form window.

Immediately, you will see that three new objects, FacultyBindingSource, FacultyTableAdapter, and FacultyBindingNavigator, are added into the project. All column headers in the Faculty table are displayed, as shown in Figure 4.29.

Now click 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 4.30.

To view each record, you can click the forward arrow on the top of the form to scan all records from the top to the bottom of the Faculty table.

If you only want to display specified columns from the Faculty table, go to the Data Sources window and select the Faculty table. Expand the table to display the

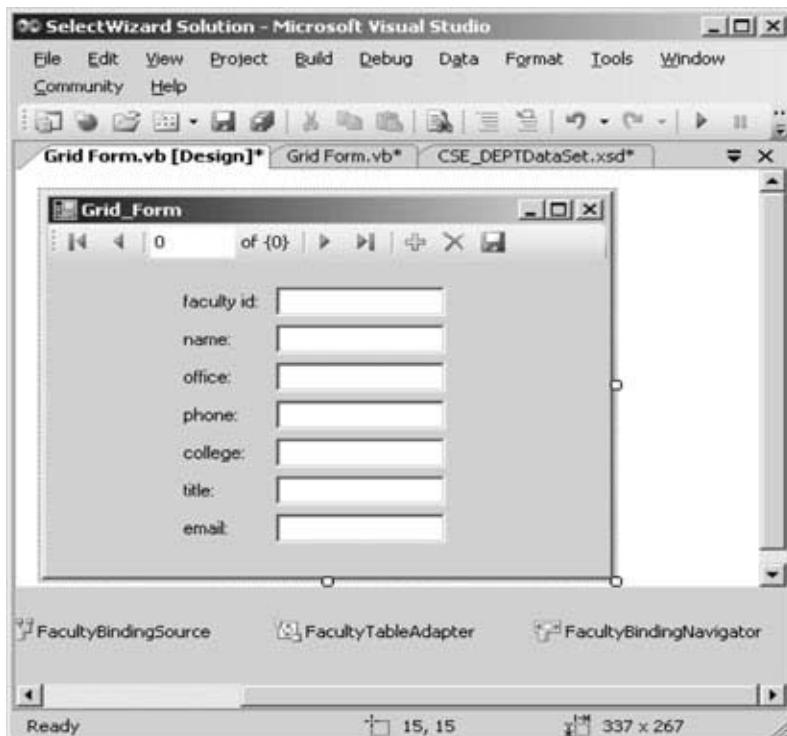


Figure 4.29. The grid view for specified columns.

individual columns, and drag the desired column from the Data Sources 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.

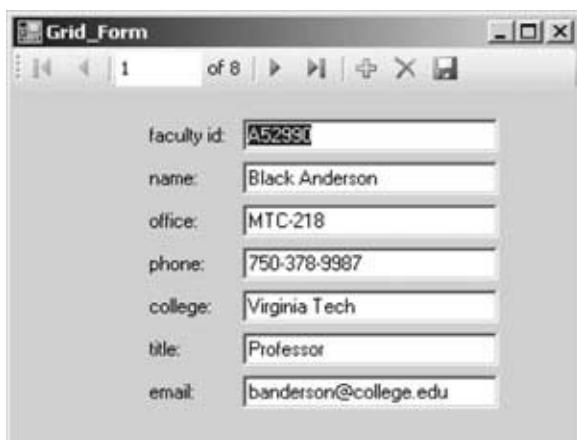


Figure 4.30. The running status of the grid view for each record.

The DataGridView is a powerful tool that allows users to view all data from a table. But generally we do not want to perform that data view as an in-line SQL statement. A so-called in-line 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, as opposed to what we call 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 we will concentrate on this technique in the following sections.

## 4.6 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 developed in the previous sections. We start from the LogIn table.

Open the Data Sources window and right-click anywhere 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 Configure item from the popup menu. Of course, you could select other items such as Add|Query or Preview Data. But right now we do not want to add a new query; we just want to modify the default query to perform our specified data query.

In the opened TableAdapter Configuration Wizard, click the Query Builder button to build our desired dynamic query. The opened Query Builder window is shown in Figure 4.31.

Query Builder provides a GUI for creating SQL queries. The top two panes are graphical panes, and the third pane is the text pane. You can select desired columns from the top graphical pane, and each column you select will be added into the second graphical pane. By using the second graphical pane, you can install the desired criteria to build user-defined queries. The query you build 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 or columns you want to query by checking the associated box 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 4.31.

Since we wish to build a dynamic SQL query for the LogIn table, what we want to do is as follows: the username and password are entered by the user when the project runs, and those two items will be embedded into the SQL SELECT statement

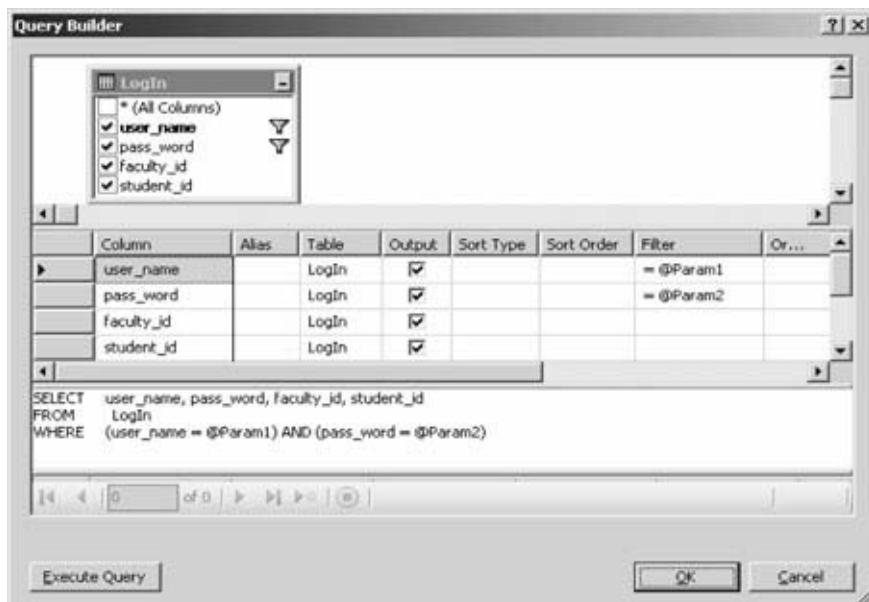


Figure 4.31. The Query Builder window.

that is sent to the data source, namely, 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 to the DataSet, then to the BindingSource via the TableAdapter, and furthermore will be reflected on the bound control on the Visual Basic.NET 2005 form window.

The problem is that when we build this query, we do not know the values for username and password, which will be entered by the user as the project runs. In other words, these two parameters are dynamic parameters. So in order to build a dynamic query with two dynamic parameters, we need to use two question marks to temporarily replace those two parameters in the SQL SELECT statement. We do this by typing a question mark in the Filter column for user\_name and pass\_word rows in the second graphical pane, which is shown in Figure 4.31. The two question marks will become two dynamic parameters represented by =@Param1 and =@Param2, respectively, after you press the Enter key on your keyboard after typing the question mark. 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. You will find that a WHERE clause is attached at the end of the SELECT statement, which is shown in Figure 4.31. The clause

---

```
WHERE (user_name = @Param1) AND (pass_word = @Param2)
```

---

is used to set up dynamic criteria for this SELECT statement. The two dynamic parameters Param1 and Param2 will be replaced by the username and password entered by the user as the project runs. You can consider the @ symbol as similar to



Figure 4.32. The Choose Methods to Generate window.

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 OK to continue to the next step. The next window shows the complete query built in the last step to ask for confirmation, and you can make modifications if you want. Click the Next button to go to the next step.

The next window provides three options. The first one allows you to modify the Fill() method to meet your specified query for your application. The second option allows you to modify the GetData() method that returns a new data table filled with the results of the SQL statement. The third one allows you to add other SQL statements such as the Insert, Update, and Delete methods.

For this application, we only need to use and modify the Fill() method. Attach the word ByUserNamePassWord to the end of the Fill method, as shown in Figure 4.32. We will use this method in our project to run the dynamic SQL statement we built in the last step. Click Next to go to the next page.

The next window shows the result of your TableAdapter configuration. If everything is going smoothly, all statements and methods should be created and modified successfully, as shown in Figure 4.33. Click the Finish button to close the configuration wizard.

Before we can begin our coding job, we need to bind data to controls on the LogIn form to set up the connection between each control on the form and the data in the data source.

## 4.7 BIND DATA TO ASSOCIATED CONTROLS IN THE LOGIN FORM

Open the Solution Explorer window and select the LogIn Form from that window, then click the View Designer button to open its GUI. Now we want to use the BindingSource to bind controls in the LogIn form, namely, the UserName and



Figure 4.33. The result of the TableAdapter Configuration Wizard.

Pass Word TextBoxes, to the associated data fields in the LogIn table in the data source.

Click the UserName TextBox, 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 the drop-down arrow to expand the following items:

- Other Data Sources
- Project Data Sources
- CSE\_DEPTDataSet
- LogIn

The expansion result is shown in Figure 4.34. Select the user\_name column by clicking it. In this way, we finish the data binding and set up a connection between the UserName TextBox control on the LogIn form and the user\_name column in the LogIn data table.

You can find that two objects, LogInBindingSource and LogInTableAdapter, are added into the project and displayed at the bottom of the window after you finish this binding.

Well, is that easy? Yes. Perform 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 pay attention to is this: when we performed the data binding for the UserName TextBox, there was no BindingSource object available because you had not performed any data binding before. So you needed to perform the steps illustrated in Figure 4.34. But after you have finished that binding, a new BindingSource object, LogInBindingSource, is created. You need to use this

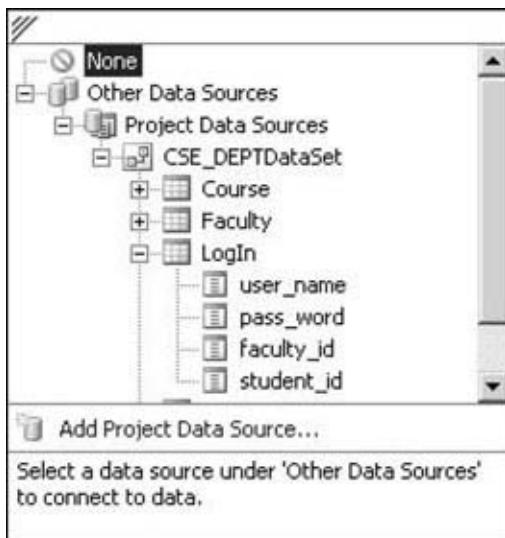


Figure 4.34. The DataBindings property.

BindingSource object to handle the data-binding jobs for all other controls on the LogIn form.



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 the following data bindings should use the newly created BindingSource to perform the data binding.

Let's perform the data binding for the Pass Word TextBox now.

Click the TextBox to select it, and then go to the DataBindings property, select the Text item, and click the drop-down arrow. This time you will find that a new BindingSource object, LogInBindingSource, shows up (Figure 4.35).

Expand this new object and select the pass\_word column by clicking it. The data binding is done.

Some readers may have noted that when we call the Fill() method, that is, the FillByNamePassWord(), 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 fill only two text box controls on the form, txtUserName and txtPassWord, with two associated columns in the LogIn table, user\_name and pass\_word. We only need to know if we can find matches for the username and password entered by the user in the LogIn table. If both items can be found in the LogIn table, the login is successful and we can continue to the next step. Two bound controls on the form, txtUserName and txtPassWord, will be filled with identical values stored in the LogIn table. It looks as if this does not make sense. In fact, we do not want to retrieve any columns from the LogIn table; instead, we only want to



Figure 4.35. The newly created BindingSource object.

find the matched items of 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 matched username and password, we do not care whether we fill the faculty\_id and student\_id. If no matched items can be found, it means that the login has failed and a warning message should be given.



To check the matched username and password entered by the user from the Data Source, one can use *Return a Single Value to Query Data* for LogIn table. But here in order to simplify this check, we use the Fill() method to fill three columns in a mapped data table in the DataSet. Then we can check whether this Fill() is successful. If it is, the matched data items have been found. Otherwise no matched data items are found.

Before we can go ahead with our coding, one thing we need to point out is the password display style in the txtPassWord text box control. Generally the password letters will be represented by a sequence of asterisks when users enter them as the project runs. To make this happen to our project, you need to set the PasswordChar property of the txtPassWord text box control to an asterisk.

Now it is time to develop code that is related to the objects we created in the previous steps, such as the BindingSource and TableAdapter, to complete the dynamic query. The operation sequence of the LogIn form is shown below:

1. When the project runs, the user first needs to enter the username and password in two text box controls, txtUserName and txtPassWord.
2. Then the user will click the LogIn button on the form to execute the LogIn event procedure.

3. The LogIn event procedure will create some local variables or objects that will be used for the data query and the next form.
4. Then the procedure will call the FillByUserNamePassWord() method to fill the LogIn form.
5. If this Fill is successful, which means that the matched data items for username and password have been found from the LogIn table, then the next window form, SelectionForm, will be displayed. Otherwise, a warning message is displayed.

Step 3 needs a little more explanation. As you know, starting with Visual Basic.NET 2003, any new item added into the project is a class, not an object. These items include Windows Forms, Labels, Buttons, and so on. In order to use those new items, one must create a new object based on the newly added class. This is a significant difference between Visual Basic.NET and Visual Basic 6.0.

The new objects created in step 3 include a new object of the LogInTableAdapter class and 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.

Keeping these points in mind, let's begin the coding for the LogIn button event procedure.

## 4.8 DEVELOP CODE TO QUERY DATA USING THE FILL() METHOD

Select the LogIn Form from the Solution Explorer window and click the View Designer button to open its GUI. Double-click the LogIn button to open its event procedure.

Based on step 3 in the operation sequence above, we first need to create two local objects: the LogInTableApt, which is an object of the LogInTableAdapter class, and the selForm, which is an object of the SelectionForm class. The newly created objects are shown in the top two lines (**A**) in Figure 4.36.

	<b>cmdLogIn</b>	<b>Click</b>
<b>A</b>	<pre>Private Sub cmdLogIn_Click (By Val sender As System.Object, ByVal e As System.EventArgs) Handles cmdLogIn.Click     Dim LogInTableApt As New CSE_DEPTDataSetTableAdapters.LogInTableAdapter     Dim selForm As New SelectionForm</pre>	
<b>B</b>	<pre>    LogInTableApt.ClearBeforeFill =True     LogInTableApt.FillByUserNamePassWord(CSE_DEPTDataSet.LogIn, txtUserName.Text, txtPassWord.Text)</pre>	
<b>C</b>		
<b>D</b>		
<b>E</b>	<pre>    If CSE_DEPTDataSet.LogIn.Count = 0 Then         MessageBox.Show("No matched username/password found!")         Exit Sub</pre>	
<b>F</b>	<pre>    End If</pre>	
<b>G</b>	<pre>    selForm.Show()     Me.Hide()</pre>	
<b>H</b>	<pre>End Sub</pre>	
	<pre>Private Sub cmdCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdCancel.Click     Me.Close() End Sub</pre>	

Figure 4.36. The coding of the LogIn button event procedure.

You need to note that all TableAdapters in this project are located in the namespace CSE\_DEPTDataSetTableAdapters. You need to use that namespace to access your desired TableAdapter.

Let's illustrate this code line by line.

- B. Before filling the LogIn table, we clean up that table in the DataSet. As mentioned in Section 4.2.1.1, the DataSet is a table holder and contains multiple data tables. But these data tables are only mapped to the real data tables in the database. Data can be loaded into these tables in the DataSet by using the TableAdapter when your project runs. Here the 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 call the Fill() method we modified in Section 4.6, FillByUserNamePassWord(), to fill the LogIn data table in the DataSet. Because we have already bound two text box 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 the bound text box controls, txtUserName and txtPassWord, when this Fill() method is executed. This Fill() method has three arguments. The first is the data table; in this case it is the LogIn table that is held by the DataSet, CSE\_DEPTDataSet. The other two parameters are dynamic parameters that were temporarily replaced by two question marks when we modified this Fill() method in Section 4.6. 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 in two bound text box 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, that is, if a matched username and password are 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 the user to handle the failure.
- F. An Exit Sub is executed to exit the event procedure. Note that exiting the event procedure does not mean exiting the project. Your project continues to run and waits 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.
- H. After displaying the next form, the current form, LogIn form, is hidden by calling the Hide() method. The keyword Me represents the current window form.



Figure 4.37. The running status of the project.

The coding 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 test this piece of code by running the project, make sure that the LogIn form has been selected as the Start form. To confirm this, go to Project|SelectWizard Properties to open the Application window and select LogInForm from the Start form box. Then click the Start button to run the project.

Your running project should match the one shown in Figure 4.37.

Enter a valid username, such as ybai, in the UserName text box and a valid password, such as reback, in the Pass Word text box, then click the LogIn button. The FillByUserNamePassWord() method will be called to fill the LogIn table in the data source. Because we entered a correct username and password, this fill will be successful and the next form, SelectionForm, will appear.

Now try to enter a wrong username or password, then click the LogIn button. A message will be displayed, as shown in Figure 4.38, to ask the user to handle this situation.

In this section, we used the LogIn form and LogIn table to perform a dynamic data query and fill a mapped data table in the DataSet from the columns in a data table in the database by using the Visual Basic.NET 2005 tools and data wizards. The coding is relatively simple and easy to follow. In the next section, we will use another method provided by the TableAdapter to pick up a single value from the database.



Figure 4.38. The warning message.

## 4.9 USE RETURN A SINGLE VALUE TO QUERY DATA FOR THE LOGIN FORM

Many people have experienced forgetting either the username or the password when they try to log on to a specified Web site to perform a purchase, order something, or obtain some information. In this section, we 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 the LogIn table as an example. Suppose you forget your password, but you want to log on to this project by using the LogIn form with your username. In this example, you can retrieve your password by using your username.

The DataSet Designer allows us to edit the structure of the DataSet. As discussed in Section 4.6, by using the DataSet Designer you can configure an existing query, add a new query, and add a new column or even a new key. The Add Query method allows us to add a new data query with the SQL SELECT statement, which returns a single value.

Open the LogIn form window from the Solution Explorer window and open the Data Sources window by clicking the Data menu item from the menu bar. Right-click anywhere inside that window and select Edit DataSet with Designer, then locate the LogIn table and right-click the last line of that table, which contains our modified method FillByUserNamePassWord(), which we used in the last section. Then select Add Query to open the TableAdapter Query Configuration Wizard.

In the opened wizard, keep the default selection Use SQL Statements, which means that we want to build a query with SQL Statements, and then click Next and choose SELECT, which returns a single value radio button. Click Next to go to the next window, and click the Query Builder button to build our query.

In the opened Query Builder dialog box, delete the default query from the third pane by right-clicking on that query and selecting Delete. Then right-click on the top pane and select the Add Table item from the popup menu to open the Add Table dialog box, select the LogIn table and click the Add button, and then click the Close button to close the Add Table dialog box. Select the pass\_word and user\_name from the LogIn table by checking those two check boxes. Right-click on the Group By column in the middle graphical pane and select Delete to remove any group choice if a Group By item is displayed. Uncheck the Output check box 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 the filter of this query. Type a question mark in the Filter field of the user\_name column, and press the Enter key on your keyboard. Your finished Query Builder should match the one shown in Figure 4.39.

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 user\_name, which is a dynamic parameter that will be entered by the user when the project runs. Click the OK button to go to the next window.

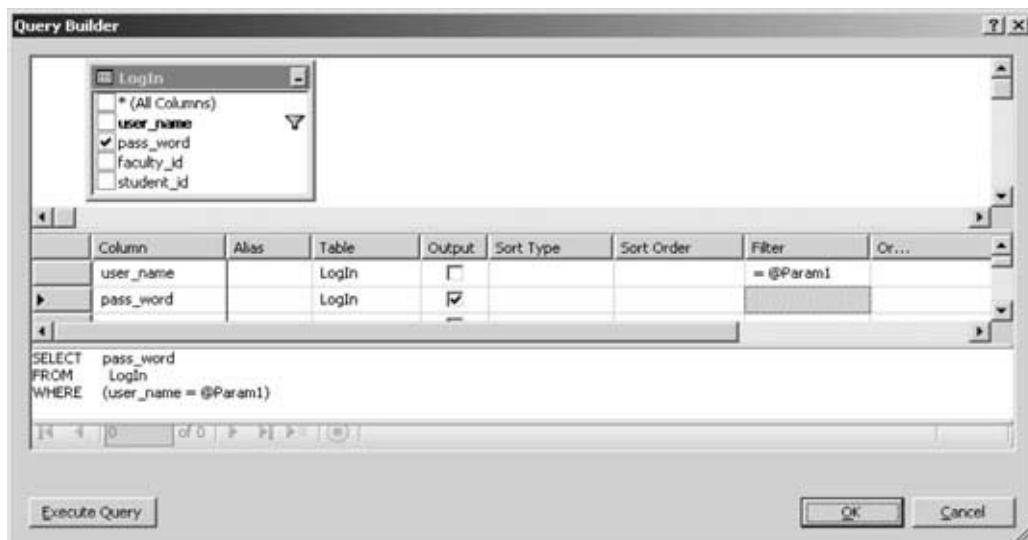


Figure 4.39. The finished Query Builder.

The next window is used to confirm your terminal SQL statement. Click Next to go to the next window.

This window asks you to choose a function name for your query. Change the default name to a meaningful name such as PassWordQuery, then click the Next button. A successful Wizard Result will be displayed if everything is fine. Click the Finish button to complete this configuration.

Now let's do our coding for the LogIn form. For the purpose of testing, we need to add another temporary button with name = cmdPW and Text = Password to the LogIn form. Open the Solution Explorer window, select and open the LogIn form, double-click the Password button to open its event procedure, and enter the following code into the event procedure, as shown in Figure 4.40.

	<b>cmdPW</b>	<b>Click</b>	
--	--------------	--------------	--

```

Private Sub cmdPW_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdPW.Click
    Dim LogInTableApt As New CSE_DEPTDataSetTableAdapters.LogInTableAdapter
    Dim passWord, Result As String
    A
    LogInTableApt.ClearBeforeFill = True
    passWord = LogInTableApt.PassWordQuery(txtUserName.Text)
    B
    If passWord = String.Empty Then
        MessageBox.Show("No matched username/password found!")
        Exit Sub
    End If
    C
    Result = "The password is " & passWord
    MessageBox.Show(Result)
    D
    E
End Sub

```

Figure 4.40. The code for the cmdLogIn button event procedure.



Figure 4.41. The running status of the LogIn form.

Let's take a closer look at this piece of code.

- 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.
- 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 finds a valid password from the LogIn table based on the username entered by the user, that password will be returned and assigned to the local string variable passWord.
- If this query cannot find any matched password, a blank string will be returned and assigned to the variable passWord. A warning message will be displayed if this situation occurs. The program will be directed to exit this subroutine.
- If the query 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 variable Result.
- A message box is used to display this password.

Now let's run the project to test this query. Click the Start button to run the project. Your running project should match the one shown in Figure 4.41.

Enter a username, such as jking, in the UserName box, and click the PassWord button. The returned password is displayed in a message box, which is shown in Figure 4.42.



Figure 4.42. The returned password.

Well, it looks like fun! Isn't it?

Now you can remove the temporary button PassWord and its event procedure from this LogIn form since we need to continue to develop our project.

In the following sections, we will 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.

## 4.10 CODING FOR THE SELECTION FORM

As discussed in Section 4.8, if the login process is successful, the Selection Form window should be displayed to allow users to continue to the next step. Figure 4.43 shows an opened Selection Form window.

Each item of information in the combo box control is associated with a form window and furthermore with a group of data stored in a data table in the database.

The operation steps for this form are summarized as follows:

1. When this form is opened, three pieces of information will be displayed in a combo box control to allow users to make a selection to browse the information related to that selection.
2. When the user clicks the OK button, the selected form should be displayed to enable the user to browse the related information.

Based on step 1, the coding to display the 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.

Open the Selection Form window and click the View Code button to open its code window. Select the SelectionForm Events item from the Class Name list box and choose the Load item from the Method Name list box to open its SelectionForm\_Load event procedure, and enter the code shown in Figure 4.44 into this event procedure.

Let's see how this piece of code works.

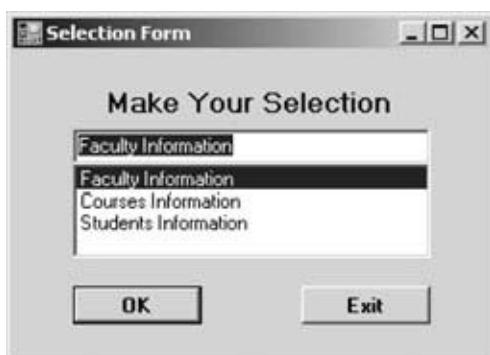


Figure 4.43. The Selection form.

The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says '(SelectionForm Events)'. The dropdown menu shows 'Load' is selected. The code in the editor is:

```

Private Sub SelectForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
    A   ComboBox1.Items.Add("Faculty Information")
    ComboBox1.Items.Add("Courses Information")
    ComboBox1.Items.Add("Students Information")
    B   ComboBox1.SelectedIndex = 0
End Sub

```

Annotations A and B point to the 'Add' method call and the 'SelectedIndex' assignment respectively.

Figure 4.44. The coding for the Selection Form\_Load event procedure.

- 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 has to be enclosed by double quotation marks.
- 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 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 the View Designer button to open the GUI of the SelectionForm object. Then double-click the OK button to open its event procedure, and enter the code shown in Figure 4.45 into this procedure.

Let's see how this piece of code works.

- First, we need to create three new objects based on three classes. You need to note that when you add a new item 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.
- Open the FacultyForm window if the user selected Faculty Information.
- Open the StudentForm window if the user selected Students Information.
- Open the CourseForm window if the user selected Courses Information.

The last coding for this form is the Exit button. Open the GUI of the SelectionForm and double-click the Exit button to open its event procedure. Enter the code shown in Figure 4.46 into this procedure.

The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says 'cmdOK'. The dropdown menu shows 'Click' is selected. The code in the editor is:

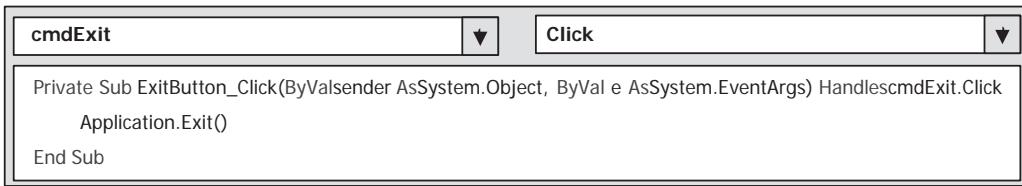
```

Private Sub OKButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdOK.Click
    A   Dim facultyform As New FacultyForm
    Dim studentform As New StudentForm
    Dim courseform As New CourseForm
    B   If ComboBox1.Text = "Faculty Information" Then
        facultyform.Show()
    C   ElseIf ComboBox1.Text = "Students Information" Then
        studentform.Show()
    D   ElseIf ComboBox1.Text = "Courses Information" Then
        courseform.Show()
    End If
End Sub

```

Annotations A, B, C, and D point to the declarations of the three forms and the conditional statements for each case.

Figure 4.45. The coding for the OK button event procedure.



```
cmdExit Click
Private Sub ExitButton_Click(ByVal Valsender As System.Object, ByVal e As System.EventArgs) Handles cmdExit.Click
    Application.Exit()
End Sub
```

Figure 4.46. The coding for the Exit button event procedure.

This coding is very simple. As the user clicks this button, the project is exited and terminated by calling the Exit() method.

Suppose the user selects the first piece of information – Faculty Information. A Faculty form window will be displayed, and it is supposed to be connected to a Faculty data table in the database. If the user selects a faculty name from the ComboBox control and click the Select button on that form (refer to Figure 4.20), all information related to that faculty member should be displayed on that form in five labels and a picture box.

Now let's see how to perform the data binding to bind controls on the Faculty form to the associated columns in the database.

## 4.11 BIND DATA TO THE ASSOCIATED CONTROLS IN THE FACULTY FORM

Open the Faculty form window from the Solution Explorer window and perform the following data bindings.

1. Select the **TitleLabel** by clicking it, then go to the **DataBindings** property, select the **Text** item, and click the drop-down arrow. Expand the following items (Figure 4.47):
  - Other Data Sources
  - Project Data Sources
  - CSE\_DEPTDataSet
  - Faculty

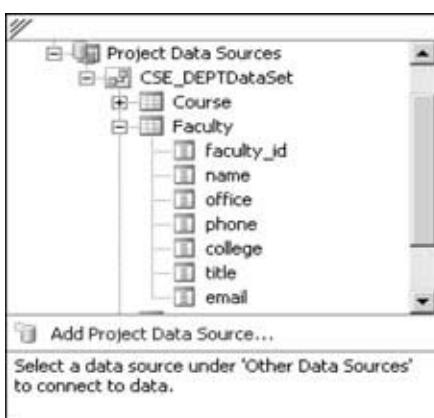


Figure 4.47. The expansion for data binding.



Figure 4.48. The expansion for the office column.

Then select the title column from the Faculty table by clicking it. In this way, you finish the binding between the label control `TitleLabel` on the Faculty form and the title column in the Faculty table. As soon as you finish this data binding, you will immediately find three components displayed under your form: `CSE_DEPTDataSet`, `FacultyBindingSource`, and `FacultyTableAdapter`.

2. Continue to select the next label from the Faculty Information GroupBox, which is the `OfficeLabel`; go to the `DataBindings` property, select the `Text` item, and click the drop-down arrow. This time you will find that a new object, `FacultyBindingSource`, is created. As discussed in Section 4.7, as soon as you finish one data binding, a new object of the data-binding source will be created and served for the form in which the binding source is located. Now we need to use this data-binding source to bind our `OfficeLabel` control. Expand this binding source until you find the Faculty table, then click the `office` column to finish this binding. An example of this expansion is shown in Figure 4.48.
3. In a similar way, you can finish the data binding for the rest of three label controls: `PhoneLabel`, `CollegeLabel`, and `EmailLabel`. The binding relationship is `PhoneLabel` to `phone` column, `CollegeLabel` to `college` column, and `EmailLabel` to `email` column in the Faculty table.

Next, we need to use the `DataSet` Designer to build our data query with the SQL SELECT statement and modify the name of the `FillBy()` method for the `FacultyTableAdapter`.

Open the Data Sources window by clicking the `Data|Show Data Sources` menu item from the menu bar. Right-click anywhere inside that window and select `Edit DataSet` with the `Designer` item to open the `DataSet` Designer Wizard. Locate the Faculty table, then right-click on the last line of the Faculty table and select the `Add|Query` item from the popup menu to open the `TableAdapter Configuration Wizard`.

In the opened wizard, click `Next` to keep the default command type – `Use SQL statements` – and click `Next` again to keep the default query type, `SELECT`, which returns rows, for the next dialog box. Then click the `Query Builder` button to open the `Query Builder` dialog box. In the middle graphical pane, move your cursor to the

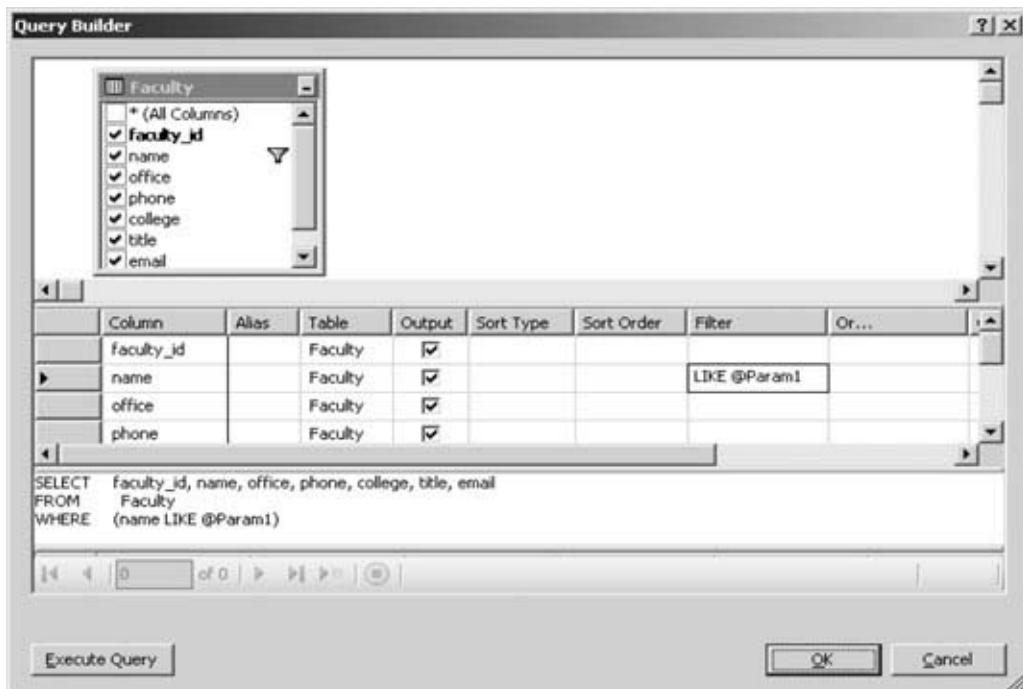


Figure 4.49. An example of the Query Builder.

Filter column along the name line, type a question mark, and press the Enter key on your keyboard. In this way, you add a WHERE clause with a dynamic parameter that is represented by LIKE @Param1 in the SQL Server database. Note that the keyword LIKE is similar to an equal symbol used in the assignment operator in a Microsoft Access query. In a SQL Server data query, LIKE is used instead of the equal symbol.

Your finished Query Builder should match the one shown in Figure 4.49.

Click the OK and Next buttons to modify the name of the FillBy() method. Attach FacultyName to the end of the FillBy, so that the modified name for this FillBy() method is FillByFacultyName() in our application. Click the Next and Finish buttons to complete this configuration.

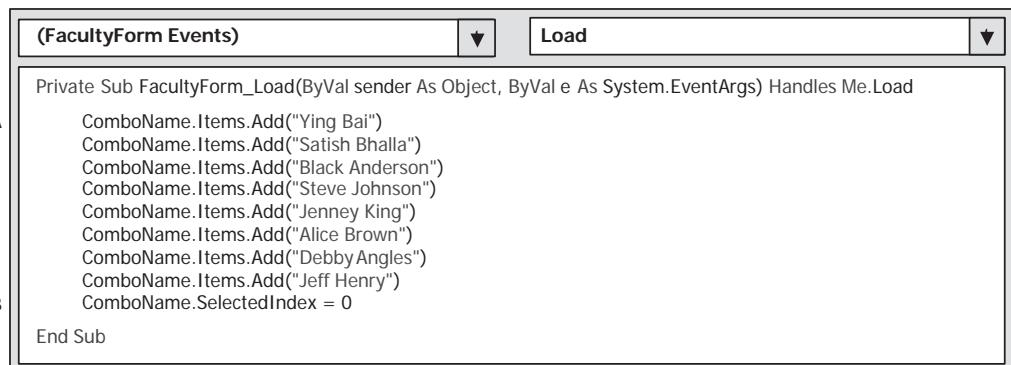
Now let's develop the code for querying the faculty information using this Faculty form with the Faculty data table in the database.

## 4.12 DEVELOP CODE TO QUERY DATA USING THE FILL() METHOD

In this section, we will only do the coding for the Select button to perform a data query and the Back button. All other buttons will be discussed and coded in the following sections.

The pseudocode or the operation sequence of this data query can be described as follows:

- After the project runs, the user has completed the login process and selects the Faculty Information item from the Selection form.



The screenshot shows the Visual Studio code editor with the title bar '(FacultyForm Events)' and a 'Load' button. The code in the editor is:

```

Private Sub FacultyForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    A
    ComboBoxName.Items.Add("Ying Bai")
    ComboBoxName.Items.Add("Satish Bhalla")
    ComboBoxName.Items.Add("Black Anderson")
    ComboBoxName.Items.Add("Steve Johnson")
    ComboBoxName.Items.Add("Jenney King")
    ComboBoxName.Items.Add("Alice Brown")
    ComboBoxName.Items.Add("Debby Angles")
    ComboBoxName.Items.Add("Jeff Henry")
    B
    ComboBoxName.SelectedIndex = 0
End Sub

```

**Figure 4.50.** The coding for the FacultyForm\_Load event procedure.

- The Faculty form will be displayed to allow users to select the desired faculty name from the Faculty Name combo box control.
- Then the user can click the Select button to query the Faculty data table to get all information related to the desired faculty member

The main coding job is performed within the Select button event procedure. But before we can do that coding, we need to add all faculty names into the Faculty Name combo box control. In this way, as the project runs the user can select a desired faculty name from that box. Since these faculty names should be displayed as the project first runs, we need to do this coding in the Form\_Load event procedure.

Select the Solution Explorer button to open the Solution Explorer window, then choose Faculty Form.vb and click the View Code button to open the code window. In the code window, go to the Class Name list box and click the drop-down arrow to select the FacultyForm Events item. Go to the Method Name list box, click the drop-down arrow, and select Load. This will open the FacultyForm\_Load event procedure. Enter the code in Figure 4.50 into this event procedure.

Let's see how this piece of code works.

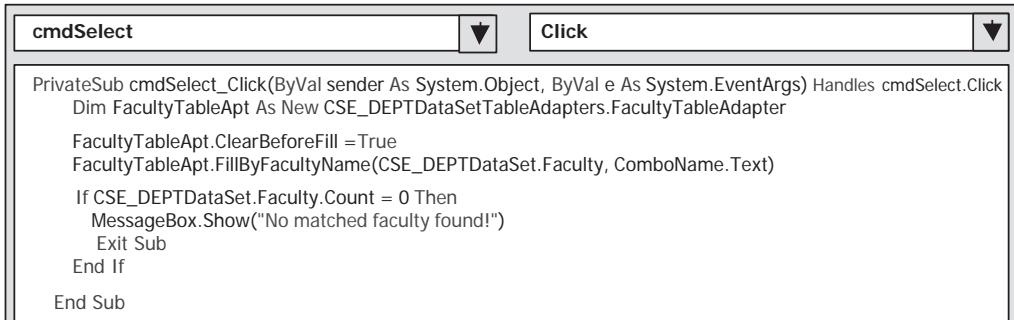
- A. First, we need to use the Add method to add all faculty names into the Faculty Name ComboBox control.
- B. Then we set the SelectedIndex value to 0, which means that the first faculty name, with an index value of 0, has been selected as a default name as the project runs.

Now we need to do the coding for the Select button event procedure.

Click the View Designer button to open the Faculty form's GUI. On the opened Faculty form, double-click the Select button to open its event procedure, then enter the code in Figure 4.51 into this event procedure.

Let's see how this piece of code works.

- A. First, create a new FacultyTableAdapter object, FacultyTableApt. In Visual Basic.NET 2005, 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.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdSelect" and the tab bar says "Click". The code is as follows:

```

A PrivateSub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
B     Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
C     FacultyTableApt.ClearBeforeFill = True
D     FacultyTableApt.FillByFacultyName(CSE_DEPTDataSet.Faculty, ComboName.Text)
E     If CSE_DEPTDataSet.Faculty.Count = 0 Then
F         MessageBox.Show("No matched faculty found!")
G         Exit Sub
H     End If
I     End Sub

```

**Figure 4.51.** The coding for the Select button event procedure.

- B. Clean up the Faculty table before it can be filled by setting the ClearBeforeFill property to True.
- C. Call the method FillByFacultyName() to fill the Faculty table with a dynamic parameter, which is selected by the user from the Faculty Name ComboBox control as the project runs.
- D. By checking the Count property of the Faculty table that is involved in our DataSet, we will know whether this fill is successful or not. If this property is equal to 0, which means that no matched record has been found 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 to require users to handle this problem. The user can either continue to select a correct faculty name or exit the project. If this property is nonzero, it indicates that this fill is successful and a matched faculty name has been found and the Faculty table in our DataSet filled. All information related to the matched faculty name will be displayed in five label controls and a picture box.

As already mentioned, in this section we only perform the coding for the Select and Back buttons. The coding for all other buttons will be provided in the following 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 (Figure 4.52).

Now we have performed the coding for three forms: LogIn, Selection, and Faculty. Let's run our project to test the functionalities of those forms. Click the Start button to run the project. Enter ybai as the username and reback as the password in the two text boxes, then click the LogIn button. The next form, the Selection form, will be displayed. Select Faculty Information and click the OK button



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdBack" and the tab bar says "Click". The code is as follows:

```

PrivateSub cmdBack_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdBack.Click
    Me.Close()
End Sub

```

**Figure 4.52.** The coding for the Back button.

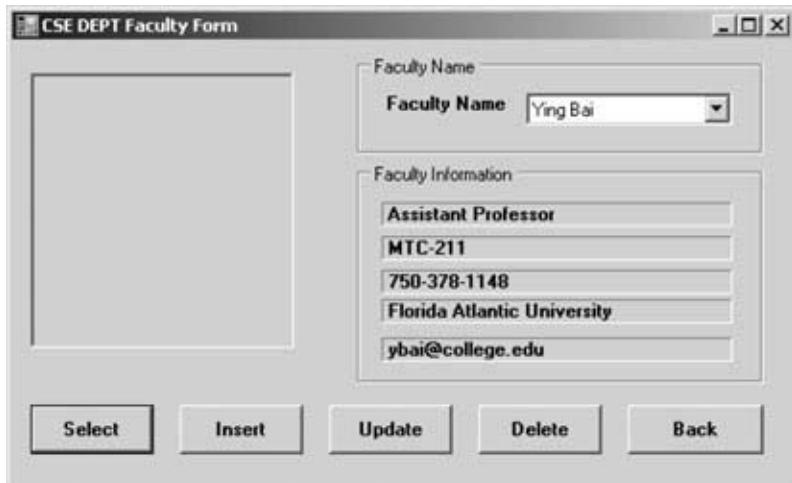


Figure 4.53. The running status of the Faculty form.

to open the Faculty form. Choose Ying Bai from the Faculty Name combo box, and click the Select button. All information about this faculty member will be displayed in five labels inside the Faculty Information group box, which is shown in Figure 4.53.

You can also try to select other faculty to test the project. When you are done, click the Back button and then the Exit button to terminate the project.

One issue you may have found while testing this project is that the faculty picture is not displayed together with the faculty information. We will solve this problem in the next section.

#### 4.13 DISPLAY PICTURES FOR THE FACULTY FORM

Storing 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 computer. We can load this picture into the project to show it as the project runs.

To display the correct faculty photo from the correct location, we need to perform the following steps to configure this operation:

- In order to make this project portable, which means that the project can be executed as an integrated body without any other additional configurations, the best place to save these faculty images is in the folder in which your Visual Basic.NET 2005 executable file is stored. The exact folder is dependent on your output file type. The folder should be **your\_project\_folder\bin\Debug** if your output file is a debug file and **your\_project\_folder\bin\Release** if your output file is a release file. In this application, our output file is a debug file; therefore, save those faculty images in the folder **SelectWizard\bin\Debug**. You do not need to specify the full path for the images' location if you save images in this way as you load them when the project runs.

```

cmdSelect Click
Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim strName As String
    strName = FindName(ComboName.Text)
    If strName = "No Match" Then
        MessageBox.Show("No Matched Faculty Found!")
        Exit Sub
    End If
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage
    PhotoBox.Image = System.Drawing.Image.FromFile(strName)
    FacultyTableApt.ClearBeforeFill = True
    FacultyTableApt.FillByFacultyName(CSE_DEPTDataSet.Faculty, ComboName.Text)
    If CSE_DEPTDataSet.Faculty.Count = 0 Then
        MessageBox.Show("No matched faculty found!")
        Exit Sub
    End If
End Sub

```

Figure 4.54. Add code to select the faculty's image.

- In order to select the correct faculty image based on the faculty name selected by the user, a function should be developed to complete this functionality.
- To display the image, a system method, System.Drawing.ImageFromFile(), is used.

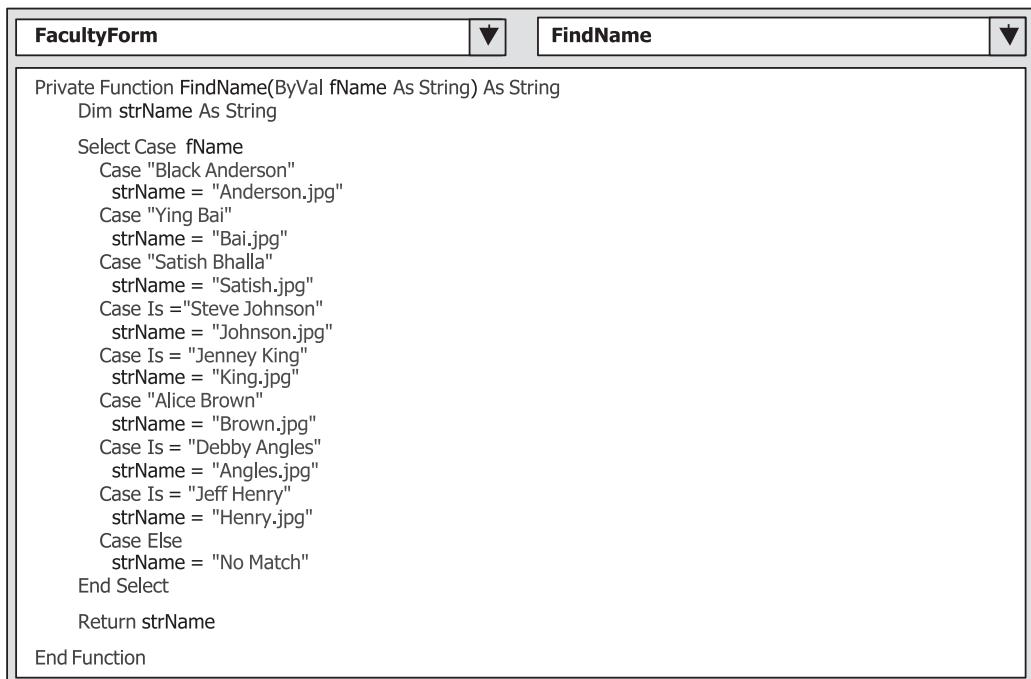
Let's take a look at the code that needs to be added into the Select button event procedure to select and display the matched faculty image.

#### 4.13.1 Modify the Code for the Select Button Event Procedure

Open the Faculty form window and double-click the Select Button to reopen its event procedure. Add the code in Figure 4.54 into this event procedure.

Let's see how this piece of code 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 below to identify and return the matched faculty image based on the input, which 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 propertySizeMode to StretchImage, we allow the image to be enlarged to fill the whole PictureBox. Then the system method is called to load and display the selected image.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "FacultyForm". In the top right corner of the code editor, there are two dropdown arrows labeled "FindName". The code itself is a VB.NET function named "FindName". It uses a Select Case statement to map faculty names to their corresponding image file names. If no match is found, it returns "No Match".

```

Private Function FindName(ByVal fName As String) As String
    Dim strName As String
    Select Case fName
        Case "Black Anderson"
            strName = "Anderson.jpg"
        Case "Ying Bai"
            strName = "Bai.jpg"
        Case "Satish Bhalla"
            strName = "Satish.jpg"
        Case Is = "Steve Johnson"
            strName = "Johnson.jpg"
        Case Is = "Jenney King"
            strName = "King.jpg"
        Case "Alice Brown"
            strName = "Brown.jpg"
        Case Is = "Debby Angles"
            strName = "Angles.jpg"
        Case Is = "Jeff Henry"
            strName = "Henry.jpg"
        Case Else
            strName = "No Match"
    End Select
    Return strName
End Function

```

Figure 4.55. The code for the function FindName.

#### 4.13.2 Create a Function to Select the Matched Faculty Image

Now let's develop a function to select the matched image for the faculty name selected by the user. The input parameter should be a faculty name, and the output should be a matched faculty image.

Keeping the Faculty form selected, click the View Code button from the Solution Explorer window to open its code window. Create a new function, `FindName()`, by entering the code in Figure 4.55 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. The string "No Match" is returned if no matched faculty image is found.

Right now we are ready to test our project. Click the Start button to run the project. Enter `ybai` as the username and `reback` as the password on the LogIn form. Click the LogIn button to open the Selection Form window, select the Faculty Information item, and then click the OK button to open the Faculty form. Select Ying Bai from the Faculty Name combo box, and click the Select button. All information related to this faculty member, including a faculty picture, will be displayed, as shown in Figure 4.56.

One point you need to remember is that you must save all faculty image files in the folder in which your project executable file is located in order to make your project work properly. In this application, this folder is **C:\SelectWizard\SelectWizard\bin\Debug**.



Figure 4.56. The running status of the Faculty form window.

At this point, we have finished designing and building our Faculty form. Next, we will take care of the Course form.



In this example, we saved our faculty image files in the folder in which the project executable file is stored. If you do not want to save your image files in this folder, you must provide the full name for your image files, which includes the full path for the folder and the image file name. For instance, if the image file Bai.jpg is saved in the folder C:\FacultyImage, you must give the full name as the returned string, as in C:\FacultyImage\Bai.jpg.

#### **4.14 BINDING DATA TO ASSOCIATED CONTROLS IN THE COURSE FORM**

The functions of this form are as the follows:

1. This form allows users to find the courses taught by the selected faculty member from the Faculty Name combo box control when users click the Select button. The courses are displayed in the Course list box.
2. The detailed information for each course, such as the course title, course schedule, classroom, credits, and enrollment, can be obtained by double-clicking the desired course in the Course list box, and is displayed in five text box controls.
3. The Back button allows users to return to the Selection form to make another selection to obtain the desired information related to that selection.

In this section, we discuss only two buttons, Select and Back. The coding for the Insert button will be discussed in the following chapters.

For step 1, in order to find the courses taught by the selected faculty member from the Course table, we need first to obtain the selected faculty ID that is associated with the faculty name selected in the Faculty Name combo box control when the user clicks the Select button, because no faculty name is available from the Course table. The only information available in the Course table is the faculty\_id. So first we need 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 taught by the selected faculty member based on the faculty\_id we obtained from the Faculty table.

Now let's do the first job, which is to create a query to obtain the associated faculty\_id from the Faculty table based on the faculty name selected in the Faculty Name combo box in the Course form.

Open the DataSet Designer Wizard and right-click the last line of the Faculty table and select Add|Query to open the TableAdapter Query Configuration Wizard. Keep the default selection Use SQL statements, and click the Next button to go to the next window. Check the radio button in front of SELECT, which returns a single value to choose this query type, and click the Next button to go the next step. Click the Query Builder to build the query.

In the Query Builder dialog box, remove the default query from the text pane, the third pane, by highlighting and right-clicking on it and selecting Delete. Then right-click on the top pane and select the Add Table item to open the Add Table dialog box. Select the Faculty table by clicking on it in the table list, and click Add and then Close to add this table. Select the faculty\_id and the name from the Faculty table by checking them in the top pane, and uncheck the Output check box for the name row in the middle pane since we do not want to query the name but only to use it as the criterion to find the faculty\_id. Then type a question mark in the Filter column for the name row and press the Enter key your keyboard. Your finished query should match the one shown in Figure 4.57.

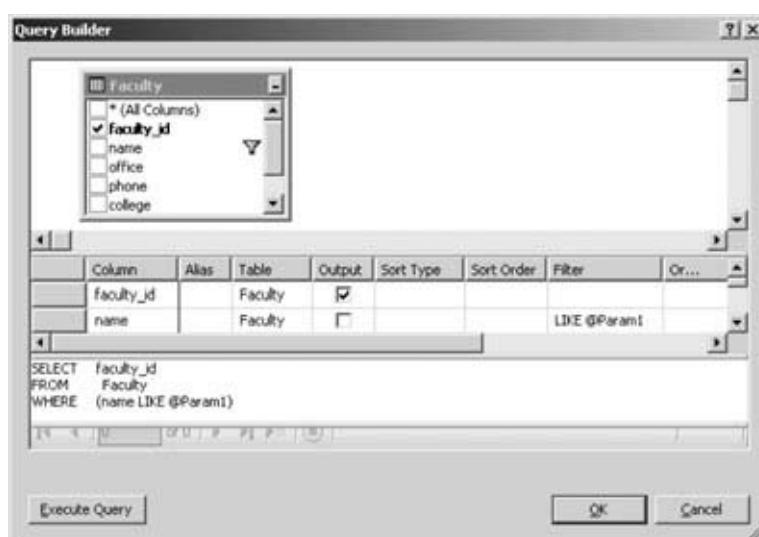


Figure 4.57. The finished query for the faculty\_id.

The SQL statement shown in the text pane is

---

```
SELECT faculty_id FROM Faculty WHERE (name LIKE @Param1)
```

---

Click the OK and the Next buttons to continue. Enter FindFacultyIDByName into the box as the function name, and then click the Next and the Finish buttons to complete this query building.

Now let's continue to build our query to find the courses taught by the selected faculty member from the Course table. Open the DataSet Designer to create the desired query and modify the Fill() method for the CourseTableAdapter.

Open the Data Sources window by clicking the Data>Show Data Sources menu item from the menu bar. Then right-click anywhere inside this window and select the Edit DataSet with Designer item to open the DataSet Designer Wizard. Right-click the last line of the Course table and choose the Configure item to open the TableAdapter Configuration Wizard. Then click the Query Builder to open the Query Builder window, which is shown in Figure 4.58.

Keep the default selections for the top graphical pane even if you only need the course column. You will see why we need to keep these default items later. Go to the Filter column in the faculty\_id row, type a question mark, and press the Enter key on your keyboard. This is equivalent to setting a dynamic parameter for the

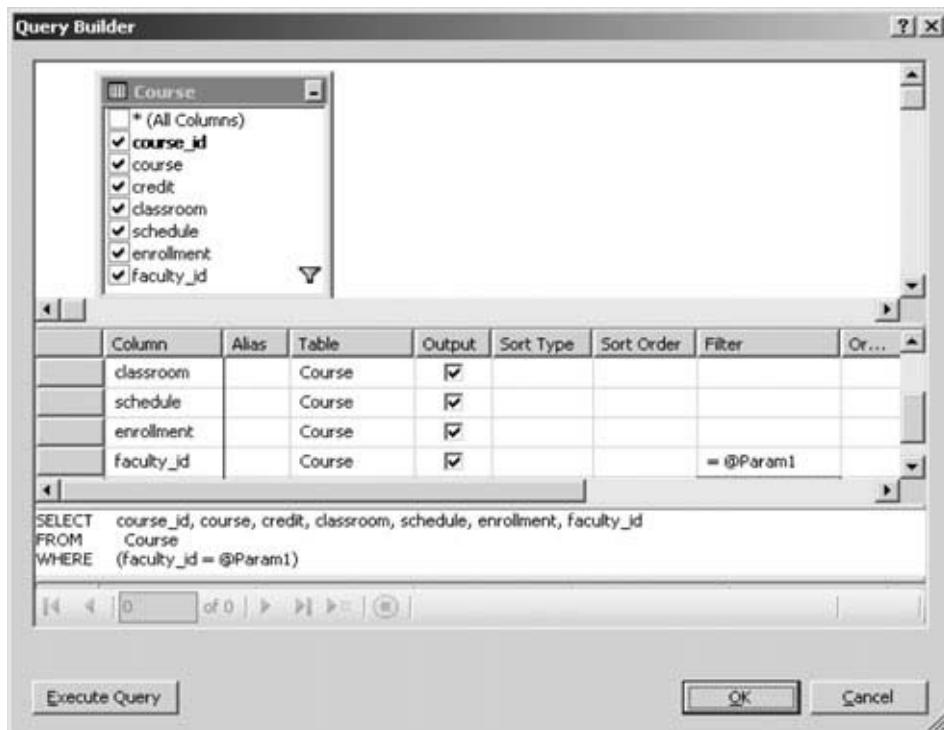


Figure 4.58. The Query Builder.

SQL SELECT statement. The completed SQL statement is displayed in the text pane, and the content of this statement is

---

```
SELECT course_id, course, credit, classroom, schedule, enrollment, faculty_id
FROM Course
WHERE (faculty_id = @Param1)
```

---

The dynamic parameter @Param1 is a temporary parameter and will be replaced by the real parameter faculty\_id as the project runs.

Click OK and then Next to return to the TableAdapter Configuration Wizard to modify the Fill() method. Attach ByFacultyID to the end of the Fill() method to get a modified method: FillByFacultyID(). Then click Next and Finish to complete this configuration.

The next step is to bind the controls from the Course form to the associated data in the Course table in the DataSet. Click the Solution Explorer button, select Course Form.vb from that window, and click the View Designer button to open the Course form's GUI.

First we need to bind the CourseList to the course column in the Course table in the DataSet. Recall from Chapter 2 that there are many records with the same faculty\_id in this Course table. Those multiple records with the same faculty\_id are distinguished by the different courses taught by that faculty member. To bind a list box to the multiple records with the same faculty\_id, we cannot continue to use the binding method we used for labels or text box controls in the previous sections. This is the specialty of binding a list box control. The important point is that the relationship between the list box and the data item in the table is one-to-many, which means that a list box can contain multiple items; in this case, the CourseList can contain many courses. So the binding of a list box control is to bind a list box to a table in the DataSet, or to the Course table in this application.

To do this binding, click the CourseList control from the Course form, go to the DataSource property, and click the drop-down arrow to expand the data source until the Course table is found. Select this table by clicking it. Figure 4.59a shows this expansion.

Continue this binding by going to the DisplayMember property, expanding the Course table to find the course column, and selecting it by clicking this item (Figure 4.59b).

In this way, we set up a binding relationship between the Course list box in the Course form and the Course data table in the DataSet.

To execute step 2, we need to bind the five text box controls in the Course form to five columns in the Course data table in the DataSet. To do this, again keep the Course form opened, and then select the Course ID text box from the Course Information group box control. Go to the DataBindings property, expand to the Text item, and click the drop-down arrow. 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 4.59c, and then choose it by clicking the course\_id column. In this way, a binding is set up between

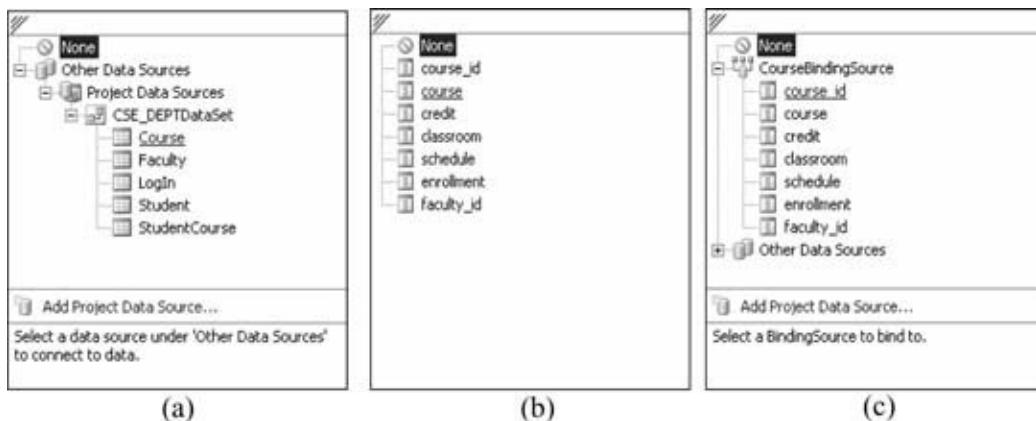


Figure 4.59. The expansion of the data source.

the Course ID text box in the Course form and the course\_id column in the Course table in the DataSet.

In a similar way, set up the data bindings for the other four text box controls: Schedule, Classroom, Credits, and Enrollment.

One point you need to note is the order in which to perform these two bindings. You must first perform the binding for the CourseList control and then perform the binding for the five text boxes.

Now we can answer the question of why we needed to keep the default selections in the top graphical pane when we built our query in the Query Builder (refer to Figure 4.58). The reason is that we need those columns to perform the data binding for our five text box controls here. In this way, each text box control in the Course form is bound with the associated data column in the Course table in the DataSet. After this kind of binding relationship is set up, all data columns in the Course data table in the DataSet will be updated by the data columns in the Course data table in the real database each time a FillByFacultyID() method is executed. At the same time, all five text boxes' content will also be updated since those text box controls have been bound to the data columns in the Course data table in the DataSet.

Now it is time for us to do the coding for this form.

## 4.15 DEVELOP CODE TO QUERY DATA FOR THE COURSE FORM

Based on the analysis of the functionality of the Course form above, when the user selects a faculty name and clicks the Select button, all courses taught by that faculty member should be listed in the Course List list box. So the coding is divided into two parts. The first part is to display all faculty names in the Faculty Name combo box when the project runs, and the second part is to perform the coding for the Select button's click event procedure.

Open the Course form window, and click the View Code button from the Solution Explorer window to open the code window. Click the drop-down arrow from the Class Name combo box to select the CourseForm Events item, then click the

```

(CourseForm Events) ▾ Load ▾
Private Sub CourseForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.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
End Sub

```

**Figure 4.60.** The coding for the CourseForm\_Load event procedure.

drop-down arrow on the Method Name combo box to select Load. In this way, we open the CourseForm\_Load event procedure. Enter the code shown in Figure 4.60 into this event procedure.

This coding is straightforward. The Add method is used to add all faculty names into the combo box. Resetting the SelectedIndex property to 0 selects the first faculty as the default one for the combo box as the project runs.

Open the Course form window by clicking the View Designer button from the Solution Explorer window, and then double-click the Select button to open its event procedure. Enter the code shown in Figure 4.61 into this event procedure.

Let's see how this piece of code works.

- A new course table adapter object is created based on the CourseTableAdapter class that is located at the namespace CSE\_DEPTDataSetTableAdapters.
- A new faculty table adapter object is also created based on the FacultyTableAdapter class. A local string variable, strFacultyID, is declared and is used

```

cmdSelect ▾ Click ▾
A Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
B     Dim CourseTableApt As New CSE_DEPTDataSetTableAdapters.CourseTableAdapter
C     Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
D     Dim strFacultyID As String
E
F     FacultyTableApt.ClearBeforeFill = True
G     strFacultyID = FacultyTableApt.FindFacultyIDByName(ComboName.Text)
H     If strFacultyID = String.Empty Then
I         MessageBox.Show( "No matched faculty_id found!")
J         Exit Sub
K     End If
L     CourseTableApt.FillByFacultyID(CSE_DEPTDataSet.Course, strFacultyID)
M     If CSE_DEPTDataSet.Course.Count = 0 Then
N         MessageBox.Show( "No Matched Courses Found!")
O         Exit Sub
P     End If
Q
End Sub

```

**Figure 4.61.** The coding for the Select button click event procedure.

- to hold the returned faculty\_id when our query FindFacultyIDByName() is executed later.
- C. Before the query FindFacultyIDByName() is executed, the faculty table adapter is cleaned up.
  - D. The query FindFacultyIDByName() is called with an argument, which 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 value returned from calling the query FindFacultyIDByName() 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. 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 the real parameter strFacultyID now (refer to Figure 4.58).
  - G 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 is filled into the Course table in the 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 taught by the selected faculty member will be filled into the Course table and loaded into the Course List list box control in the Course form, and furthermore, the detailed course information including the course ID, course schedule, classroom, credits, and enrollment for the course selected in the Course List list box will be displayed in the five text box controls since these five text box controls have been bound to those related columns in the Course table.

Return to the Course form window by clicking the View Designer button from the Solution Explorer window, then double-click the Back button to open its event procedure and enter the code Me.Close() into this event procedure.

That's it! The coding is done.

Let's test our project by running it. Click the Start button to run the project. Enter ybai and reback as the username and password on the LogIn form, and then select Course Information from the Selection combo box. Click OK to open the Course form, which is shown in Figure 4.62.

On the Course form, select the default faculty name, Ying Bai, and click the Select button to fill and load all courses taught by this faculty member into the Course table in the DataSet as well as the Course List list box in this form.

The filled courses are displayed in the Course List list box, which is also shown in Figure 4.62.

Now let's go one step further by clicking a course from the Course List list box. Immediately the detailed information about that selected course, including the course ID, schedule, classroom, credits, and enrollment, will be displayed in the five text box controls. This makes sense since those five text box controls have been bound to the five associated columns in the Course table in our DataSet. As you click on one of the courses in the Course List list box, you equivalently select and pick up one course record from the Course table. Recall that the Course List list box

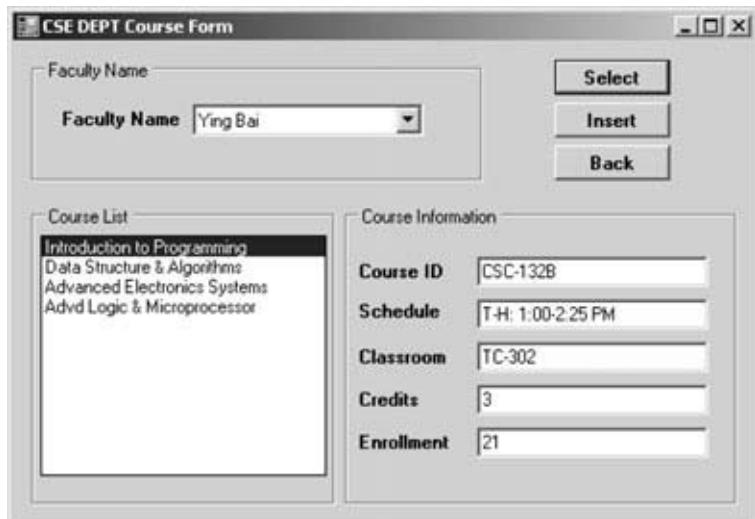


Figure 4.62. The running status of the Course form.

was bound to the Course table in our DataSet by using the CourseBindingSource when we performed this data binding in Section 4.14. For the selected course record, five columns of that record have been bound to the five text box controls in the form, so the data related to those five columns will also be reflected in these five text box controls. These relationships can be represented and illustrated by connections in Figure 4.63.

It is very interesting, isn't it?

Yes! This is the power provided by Visual Basic.NET 2005. By using Design Tools and Wizards in Visual Basic.NET 2005, it is very easy and fun to develop professional database programming in the Visual Basic.NET environment.

The last form, which is the Student form, will be left as homework for students to finish developing the data connection and operation between the Student form and the Student table as well as the StudentCourse table. For your reference, a completed project named SampleWizards Project, which contains the coding for the Student form, has been developed and is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 4**. You can

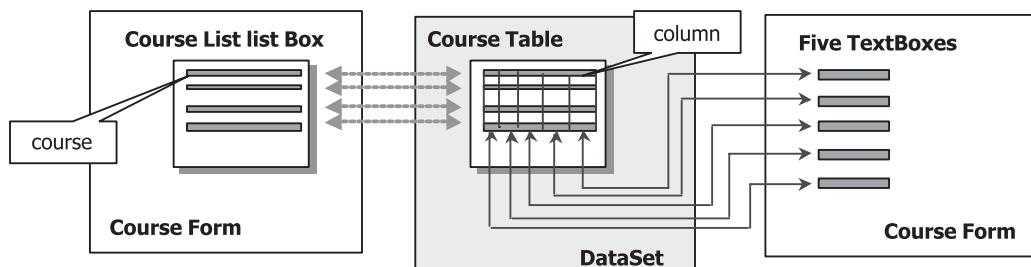


Figure 4.63. The relationships between the Course List list box, the Course Table, and the text boxes.

get it from that folder under the solution named SampleWizards Solution. The database used for that project is Microsoft Access.

The completed coding for this SelectWizard project, including the source code, GUI designs, Data Source, and Query Builders, can be found in this folder.

## PART II DATA QUERY WITH RUNTIME OBJECTS

Unlike the sample data-driven application program we developed in Part I, in which quite a few of the design tools and wizards provided by Visual Basic.NET 2005 are utilized to help us finish 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 2.0 objects directly by writing Visual Basic.NET 2005 code without using Visual Basic.NET 2005 design-time wizards, tools, or autogenerated code. All data-driven objects are created and implemented during the period when the project runs. In other words, all those objects are created dynamically.

The shortcoming of using Visual Basic.NET 2005 tools and wizards to create data connections is that the autogenerated connection code related to tools and wizards is embedded into your programs, and this connection code is machine dependent. Once the connection information in your programs is compiled, it cannot be modified. In other words, your programs cannot be distributed to and run on other platforms.

Compared with tools and wizards, there are some advantages of using runtime objects to make the data operations for your Visual Basic.NET 2005 project. One of the most important advantages is that they provide programmers more flexibility in creating and implementing connection objects and data operation objects related to ADO.NET and allow them to use different methods to access and manipulate data from the data source and the database. But everything has both a good and a bad side, and this is true here. The flexibility also brings some complex issues. For example, you have to create and use different data providers and different commands to access the different databases by using 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 discussion was provided in Chapter 3, we will have a more detailed discussion of this topic in this section so that readers get a clear picture of this issue.

### 4.16 INTRODUCTION TO RUNTIME OBJECTS

Runtime objects can be defined as objects or instances used for data connections and operations in a data-driven application that are created and implemented while your project runs; in other words, objects are created and utilized dynamically. To understand what kind of objects are most commonly used in an application, let's first discuss the most useful classes provided by ADO.NET.

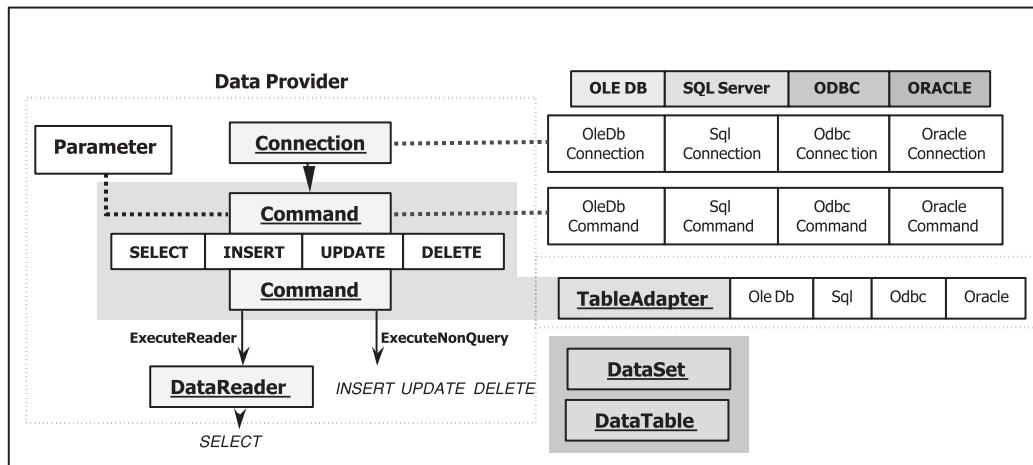


Figure 4.64. Classes provided by ADO.NET.

According to Chapter 3, the ADO.NET architecture can be divided into three components: Data Provider, DataSet, and DataTable. These three components are directly related to different associated classes, as shown in Figure 4.64.

Data Provider contains four components:

1. Connection
2. Command
3. DataReader
4. TableAdapter

All components of the Data Provider are Data Provider-dependent components, which means that all these components, namely, Connection, Command, TableAdapter (TableAdapter), and DataReader, are identified and named based on the real data provider or database used. For example, the Data Provider used for an SQL Server database must be identified and named by a prefix such as Sql. For example,

- Data Connection component: SqlConnection
- Data Command component: SqlCommand
- DataAdapter (TableAdapter): SqlDataAdapter (SqlDataAdapter)
- DataReader components: SqlDataReader

The same definition is needed for all three other 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 4.64, four data providers are commonly used in database programming in Visual Basic.NET 2005. You must create the correct connection object based on your real database by using the specific prefix.

However, 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 the DataSet and the DataTable can be filled by using the DataAdapter or TableAdapter components.

ADO.NET provides different classes to allow users to develop a professional data-driven application by using the different methods. Among those methods, two popular methods will be discussed in detail in this part.

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 methods of a DataAdapter, such as SelectCommand, InsertCommand, UpdateCommand, or DeleteCommand. As shown in Figure 4.64, 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 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 to your applications. For instance, you cannot access different data tables separately to perform multiple specific data operations.

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 and 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; you can also perform data manipulations by calling the ExecuteNonQuery() method to the desired DataTable.

In this section, we provide three sample projects to illustrate these two methods; AccessSelectRTOBJECT, SQLSelectRTOBJECT, and OracleSelectRTOBJECT, which are associated with Microsoft Access, Microsoft SQL Server, and Oracle databases, respectively.

To understand these two methods better, 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.

### 4.16.1 Procedure of Building a Data-Driven Application Using Runtime Objects

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 the 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 to be filled with data.
6. Call the suitable command/object such as the SelectCommand or the Fill() or the DataReader to make a data query.
7. Fill the data to the bound controls on the Visual Basic.NET 2005 form.
8. Release the TableAdapter, Command, DataTable, and the DataReader used.
9. Close the database Connection object if no more database operations are needed.

Now let's first develop a sample project to access the data using runtime objects for a Microsoft Access database.

#### **4.17 BUILD A MICROSOFT ACCESS DATABASE PROJECT – ACCESSSELECTRTOBJECT**

The Microsoft Access database file used in this sample project is CSE\_DEPT.mdb and is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **database\Access**.

First, we need to create a Visual Basic.NET 2005 project named AccessSelectRTObject with five form windows: LogIn, Selection, Faculty, Course, and Student. Refer to Section 4.3.1 to build these form windows.

Open this project after all forms are developed. Let's begin to develop a data-driven application starting with the LogIn form.

##### **4.17.1 Query Data Using Runtime Objects for the LogIn Form**

Open the LogIn form from the Solution Explorer window by clicking the View Designer button, as shown in Figure 4.65.

Two buttons are added into this form. The TabLogIn button is used to trigger the associated click event procedure to execute the first data query method, the DataSet-TableAdapter method. The ReadLogIn button is used to trigger the associated event procedure to run the data query in the second method, the DataReader method.

Now click the View Code button to open its code window to begin coding.



Figure 4.65. The LogIn form window.

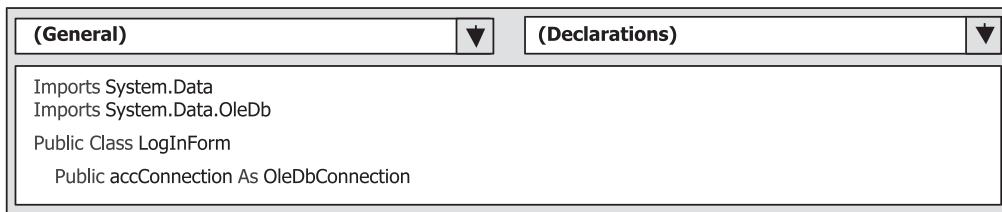


Figure 4.66. The declaration of the namespace for the OleDb Data Provider.

#### 4.17.1.1 Declare the Runtime Objects

As 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 in the top line of your code window to allow Visual Basic.NET 2005 to know that you want to use this specified Data Provider. Enter the code in Figure 4.66 at the top of this code window.

The Imports System.Data provides a reference to the namespace that will be used in this project. As discussed in the last section and in Chapter 3, both the DataSet and the DataTable components are located at this namespace, and these components will be used in this project later, so you must provide the reference to that namespace to allow the Visual Basic.NET 2005 to access it.

The following code is used to declare a new global instance of the OleDbConnection classes.

- **Public accConnection as OleDbConnection:** Create a global connection object accConnection based on the class OleDbConnection since this connection will be used for the whole application. Although declared as a module level object, the accessing mode **Public** can be accessed by all event procedures defined in all different forms from the project.

The first job you need to do after a new instance of the data connection object is declared is to connect your project with the database you selected.

#### 4.17.1.2 Connect to the Data Source with the Runtime Object

Because the connection job is the first thing you need to do before you can perform any data queries, 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 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 the drop-down arrow to select the Load method to open the LogInForm\_Load event procedure, and enter the code shown in Figure 4.67 into this event procedure.

Let's see how this piece of code works.

- A. A connection string should be created first based on the procedure listed in Section 4.16.1. The connection string is used to indicate the connection information, including the name of the data provider, the location, the name of the database, and the username and password to access the database selected. In

```

(LogInForm Events) Load

A Private Sub LogInForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
B     Dim strConnectionString As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
C         "Data Source=C:\database\CSE_DEPT.mdb;"
D     accConnection = New OleDbConnection(strConnectionString)
E
F     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
End Sub

```

**Figure 4.67.** The coding for the LogInForm\_Load event procedure.

our case, no username and password are utilized for our database, so those two items are missed from this connection string. You would need to add these two pieces of information if your database did utilize the items. The database driver for Microsoft Access is Microsoft Jet OLEDB 4.0, and our database file is named CSE\_DEPT.mdb and located in the **C:\database** directory.

- B. By using the keyword **New**, we initialize a new instance of the connection class **OleDbConnection** with the connection string we built in step A.
- C. A **Try...Catch** block is utilized here to catch any mistakes caused by establishing 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 strategy is to avoid an unnecessary system debug process and simplify the procedure.
- D. An **OleDbExceptionError** message will be displayed if an error related to the **OleDb** connection occurs.
- E. An **InvalidOperationExceptionError** message will be displayed if an invalid operation error is encountered.
- F. This step confirms that our database connection is successful. If not, an error message is displayed and the project is exited.

After a database connection is successfully made, we need to use the connection to access the database to perform our data query job.

#### 4.17.1.3 Coding for Method 1: Using DataSet-TableAdapter to Query Data

In this section, we will create and use runtime objects to query the data by using the **DataSet-TableAdapter** method.

Open the LogIn form window by clicking the View Designer button, and then double-click the TabLogIn button to open its event procedure. Enter the code shown in Figure 4.68 into this event procedure.

Let's see how this piece of code works.

	TabLogIn	▼	Click	▼
--	----------	---	-------	---

```

Private Sub TabLogIn_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TabLogIn.Click
    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)"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim LogInTableAdapter As New OleDbDataAdapter
    Dim accDataTable As New DataTable
    Dim accCommand As New OleDbCommand
    Dim selForm As New SelectionForm

    B
    accCommand.Connection = accConnection
    accCommand.CommandType = CommandType.Text
    accCommand.CommandText = cmdString
    C
    accCommand.Parameters.Add("@Param1", OleDbType.Char).Value = txtUserName.Text
    accCommand.Parameters.Add("@Param2", OleDbType.Char, 8).Value = txtPassWord.Text
    D
    LogInTableAdapter.SelectCommand = accCommand
    LogInTableAdapter.Fill(accDataTable)
    E
    If accDataTable.Rows.Count > 0 Then
        MessageBox.Show("LogIn is successful")
        'SelForm.Show()
        'Me.Hide()
    G
    Else
        MessageBox.Show("No matched username/password found!")
    End If
    H
    accDataTable.Dispose()
    accDataTable = Nothing
    accCommand.Dispose()
    accCommand = Nothing
    LogInTableAdapter.Dispose()
    LogInTableAdapter = Nothing
End Sub

```

Figure 4.68. The coding for the TabLogIn button.

- 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 tricky issue exists when you break a long query string into multiple substrings, 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 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.



An SQL statement should be represented by a string in a single line in Visual Basic.NET 2005. If the statement is too long, it can be broken into multiple substrings, but the original format of the SQL statement cannot be modified, which means that a space is required between each clause in the SQL statement.

- 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 parameters that will be entered by the user via two text boxes, txtUserName and txtPassWord, respectively, when the project runs. To add these two parameters, the Add() method in the Parameters collection is called. The Add() method can be overloaded. 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 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 user's 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 success message is displayed if this property's value is greater than 0, which means that some data has been filled into the LogIn table. Note that the following two lines of code will be used later for our actual project. The purpose of these two lines of code is to display the next form window, the Selection form, and hide the current form window, the LogIn form, if the login process is successful. But currently, in order to test our project, a message box is used. Later on we will use these two lines of code to replace the message box in our final code.
- 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 coding for the second method.

#### **4.17.1.4 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 the ReadLogIn button to open its event procedure. Enter the code shown in Figure 4.69 into this event procedure.

Let's see how this piece of code works.

Most of the code in the top section is identical to that in the TabLogIn button event procedure, with two exceptions. First, a DataReader object is created to replace the TableAdapter to perform the data query. Second, the DataTable is

	ReadLogIn	▼	Click	▼
--	-----------	---	-------	---

```

Private Sub ReadLogIn_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ReadLogIn.Click
    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)"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim accCommand As New OleDbCommand
    Dim accDataReader As OleDbDataReader
    Dim selForm As New SelectionForm

    accCommand.Connection = accConnection
    accCommand.CommandType = CommandType.Text
    accCommand.CommandText = cmdString
    accCommand.Parameters.Add("@Param1", OleDbType.Char).Value = txtUserName.Text
    accCommand.Parameters.Add("@Param2", OleDbType.Char, 8).Value = txtPassWord.Text
    accDataReader = accCommand.ExecuteReader

A    If accDataReader.HasRows = True Then
        MessageBox.Show("LogIn is successful")
        'SelForm.Show()
        'Me.Hide()
B    Else
        MessageBox.Show("No matched username/password found!")
    End If

C    accCommand.Dispose()
    accCommand = Nothing
    accDataReader.Close()
    accDataReader = Nothing
End Sub

```

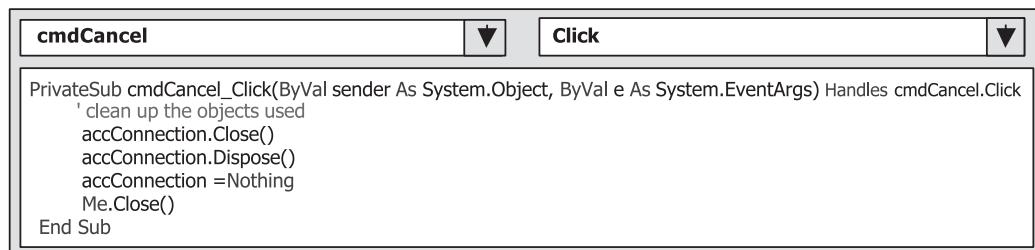
**Figure 4.69.** The coding for the ReadLogIn button event procedure.

removed from this event procedure since we do not need it for our data query in this method.

- 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.
- 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 code in the next two lines will be used later to replace this message box function. But at this moment, we use this message box to test our project.
- An error message will be displayed to require the user to handle this situation if HasRows is False, which means that no data has been received by the DataReader and the login has failed.
- A cleaning job is performed to release all objects we used for this data query.

#### 4.17.1.5 Clean Up the Objects Used

Before we can test our project, we need to finish the coding of the last button, Cancel. The purpose of this coding is to clean up the objects used in this form. Return to the LogIn form window by clicking the View Designer button from the Solution Explorer window, and then double-click the Cancel button to open



```

cmdCancel Click
Private Sub cmdCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdCancel.Click
    ' clean up the objects used
    accConnection.Close()
    accConnection.Dispose()
    accConnection = Nothing
    Me.Close()
End Sub

```

**Figure 4.70.** The coding for the Cancel button event procedure.

its event procedure. Enter the code shown in Figure 4.70 into this event procedure.

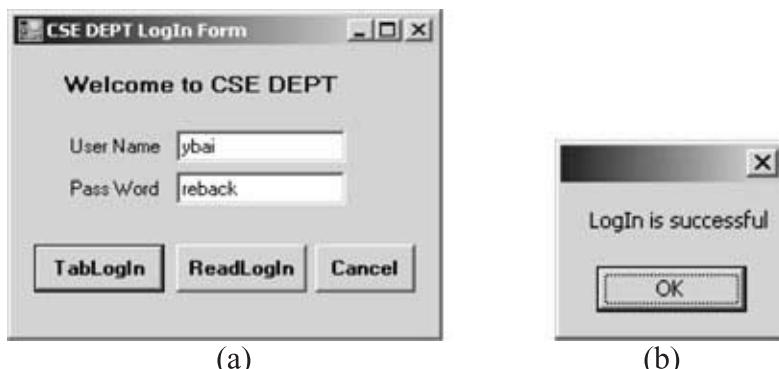
To release the Connection object, a Close() method is called first. Then the Dispose() method and the Nothing property are used to finish this release. Finally, the system method Close() is called to close the whole project. The keyword Me represents the current form window – the LogIn form. Note that the Connection instance would not be released if this Cancel button were not clicked. In the normal case, we still need to use this Connection object for the following data query if the login process is successful.

It is the time for us to test our project. Click the Start button to begin our project. Enter ybai and reback as the username and the password into the two text boxes on the LogIn form window, then click the TabLogIn button (Figure 4.71a). A success message is displayed, which means that our data query is successful (Figure 4.71b).

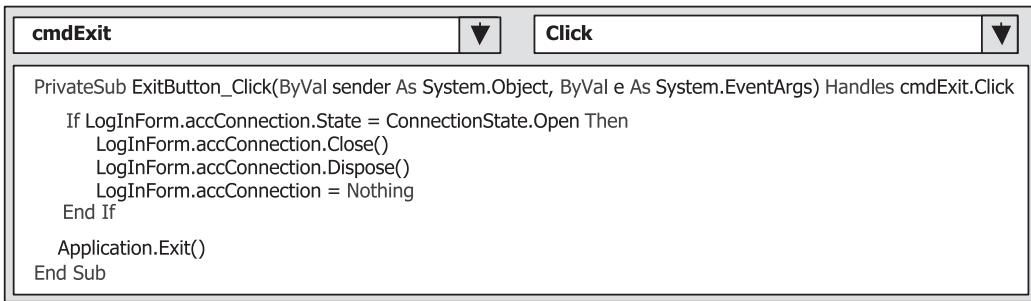
Click OK in the message box, and then click the ReadLogIn button to test our data query by using the DataReader method. Similarly, a success message is displayed, shown in Figure 4.71b. You can try to enter other correct usernames and passwords to test the project and even enter a wrong username or password to see what will happen.

Click the Cancel button to terminate the project.

Our project is successful! But before we can move on, we need to replace the success message with the two lines of replacement coding, selForm.Show() and Me.Hide(), for both event procedures, TabLogIn and ReadLogIn. By using these two lines, the next form, Selection, will be displayed and the current form, LogIn, will disappear from the screen.



**Figure 4.71.** The running status of the project.



The screenshot shows the Microsoft Visual Studio code editor with a single event procedure defined. The procedure is named `ExitButton_Click` and is associated with the `cmdExit` button. The code itself handles the disposal of a connection object and exits the application.

```

Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdExit.Click
    If LogInForm.accConnection.State = ConnectionState.Open Then
        LogInForm.accConnection.Close()
        LogInForm.accConnection.Dispose()
        LogInForm.accConnection = Nothing
    End If
    Application.Exit()
End Sub

```

**Figure 4.72.** The coding for the cmdExit button event procedure.

Before we can move on to the Faculty form, we need to add a small piece of code to the Selection form.

### 4.17.2 Coding for the Selection Form

The first coding we need to do for this form is that of the Exit button. The issue is that before we can exit this project, we must make sure that the Connection instance has been released. Otherwise it may cause some trouble when you run the project again in the future.

Open the Exit button event procedure and enter the code shown in Figure 4.72 into this procedure.

The coding for this event procedure is straightforward. Since the Connection instance is created in the LogIn form window with the Public accessing mode, it can be accessed by all event procedures in all forms in the current application. After an inspection of the connection state is executed, the Connection object is released using the `Dispose()` and `Close()` methods if the Connection instance is still open. Finally, the project is exited by calling a system method, `Application.Exit()`.

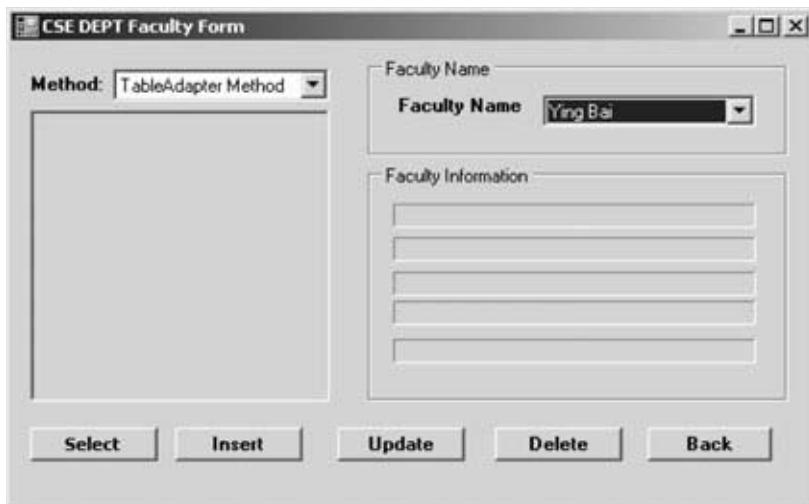
For the coding in the Form\_Load and OK button event procedures, refer to Figures 4.44 and 4.45 in section 4.10.

Now we can move on to the next form, Faculty.

### 4.17.3 Query Data Using Runtime Objects for the Faculty Form

The Faculty form window has been modified a little, and a new combo box has been added to this form to allow the user to make a selection between two methods, `TableAdapter` or `DataReader`, to perform the data query job. Open the Faculty form window by clicking the View Designer button from the Solution Explorer window, and add one more combo box above the faculty photo, as shown in Figure 4.73. Name this combo box `ComboMethod` and set its `DropDownStyle` property to `DropDownList`. Add a label to the left of this combo box, and make its `Text` property “Method:”.

Now open the code window of the Faculty form by clicking the View Code button from the Solution Explorer window. First let’s create some form-level variables and do the coding for the `FacultyForm_Load()` event procedure to put our initialization code in it. Click the drop-down arrow on the Class Name combo box and choose the `FacultyForm Events` item, then click the drop-down arrow on the



**Figure 4.73.** The modified Faculty form window.

Method Name combo box and select the Load item to open this event procedure. Enter the code in Figure 4.74 into this code window.

Let's see how this piece of code 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).

```
(FacultyForm)
Load

A Imports System.Data
Imports System.Data.OleDb

B Public Class FacultyForm
    Private FacultyLabel(6) As Label           'Faculty table has 7 columns

    Private Sub FacultyForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        C If LogInForm.accConnection.State <> ConnectionState.Open Then
            MessageBox.Show("Database has not been opened!")
            Exit Sub
        End If
        D ComboBoxName.Items.Add("Ying Bai")
        ComboBoxName.Items.Add("Satish Bhalla")
        ComboBoxName.Items.Add("Black Anderson")
        ComboBoxName.Items.Add("Steve Johnson")
        ComboBoxName.Items.Add("Jenney King")
        ComboBoxName.Items.Add("Alice Brown")
        ComboBoxName.Items.Add("Debby Angles")
        ComboBoxName.Items.Add("Jeff Henry")
        ComboBoxName.SelectedIndex = 0
        E ComboBoxMethod.Items.Add("TableAdapter Method")
        ComboBoxMethod.Items.Add("DataReader Method")
        ComboBoxMethod.SelectedIndex = 0
    End Sub
End Class
```

**Figure 4.74.** The coding for the FacultyForm\_Load event procedure.

- B. In order to hold the retrieved data, an object array is declared as a form-level or module-level label array. Since the array index is 0-based and there are 7 columns in our faculty data table, the array size is selected as 6. A little trick for the size of this array is that the first index of this array is 0, not 1. So in all we have 7 label objects defined with an index from 0 to 6.
- C. Before we can perform any data queries, we must make sure that the connection from our project to the database is still active, which means that the connection is still open. Since we created our Connection instance in the LogIn form with the accessing mode of Public, our connection instance accConnection is a module-level object, but it can be accessed by all other event procedures in all form windows from this project. To access this Connection instance, one must begin with the name of the class in which this connection object is created. 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 a 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. Two 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.

Next, we need to do the coding for the Select button.

As the project runs, the user can choose either the TableAdapter or the DataReader method from the ComboMethod combo box to perform the data query job. Then the user needs to click the Select button to begin the data query.

Click the drop-down arrow on the Class Name combo box and select the cmdSelect item, then click the drop-down arrow on the Method Name combo box and select the Click item to open the cmdSelect click event procedure. Enter the code shown in Figure 4.75 into this event procedure.

Let's see how this piece of code 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 object accCommand will be used for both methods.
- B. The Command instance is initialized with the Connection object that is created in the LogIn form, 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 is a dynamic parameter entered by the user as the project runs, to the Parameters collection that is a property of the Command object. In this way, we finish building the Command object with our desired data query statement. One point to be noted is that the first argument of this Add() method must be identical to the column name in the Faculty table; also, it must be identical to the dynamic parameter we defined in the query string cmdString2.

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, name, office, phone, college, title, email FROM Faculty"
    Dim cmdString2 As String = "WHERE name=@Param1"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim FacultyTableAdapter As New OleDbDataAdapter
    Dim accCommand As New OleDbCommand
    Dim accDataReader As OleDbDataReader
    Dim accDataTable As New DataTable

    accCommand.Connection = LogInForm.accConnection
    accCommand.CommandType = CommandType.Text
    accCommand.CommandText = cmdString
    accCommand.Parameters.Add("@Param1", OleDbType.Char).Value = ComboName.Text
    Call ShowFaculty(ComboName.Text)

    If ComboMethod.Text = "TableAdapter Method" Then
        FacultyTableAdapter.SelectCommand = accCommand
        FacultyTableAdapter.Fill(accDataTable)
    End If

    If accDataTable.Rows.Count > 0 Then
        Call FillFacultyTable(accDataTable)
    Else
        MessageBox.Show("No matched faculty found!")
    End If

    accDataTable.Dispose()
    accDataTable = Nothing
    FacultyTableAdapter.Dispose()
    FacultyTableAdapter = Nothing

    Else
        accDataReader = accCommand.ExecuteReader
        If accDataReader.HasRows = True Then
            Call FillFacultyReader(accDataReader)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        accDataReader.Close()
        accDataReader = Nothing
    End If

    accCommand.Dispose()
    accCommand = Nothing
End Sub

```

Figure 4.75. The coding for the cmdSelect click event procedure.

- D. A subroutine ShowFaculty() 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 is chosen, which means that the user wants to perform the data query with the TableAdapter method to fill the Faculty table, 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 or not. If this table filling is fine, a subroutine FillFacultyTable() is executed with the filled Faculty table as the argument to fill the label 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 selects the second method, the DataReader method, to perform this data query, the ExecuteReader() method is called to run the completed command object with our desired query statement. The returned data is assigned to the DataReader object.
- J. By checking the HasRows property, we can determine whether this query is successful or not. If this property is True, which means that the DataReader did receive the data from this query, a subroutine FillFacultyReader() is called with the DataReader as an argument to fill the label controls on the Faculty form window.
- K. Otherwise, an error message is displayed to show the user that no matched faculty name has been found from this query.
- L. The DataReader object is released.
- M. The Command object is also released.

Now let's take a look at the coding for two subroutines. The first subroutine is FillFacultyTable(), which is shown in Figure 4.76.

Let's see how this piece of code 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 data from each row. The integer pos1 is used as a loop counter later to retrieve the data from the DataTable and assign them to the associated bound label control on the Faculty form.
- B. Next, we need to initialize the module-level object array FacultyLabel by creating seven instances of the Label control. Recall that we have seven columns in our Faculty table, so this label array size is 7 (from 0 to 6).

	<b>FacultyForm</b>	▼
	<b>FillFacultyTable</b>	▼
<pre> Private Sub FillFacultyTable(ByVal FacultyTable As DataTable)     Dim pos1 As Integer     Dim column As New DataColumn     Dim row As DataRow     For pos2 As Integer = 0 To 6           'Initialize the object array         FacultyLabel(pos2) = New Label()     Next pos2     Call MapFacultyTable(FacultyLabel)     For Each row In FacultyTable.Rows         For Each column In FacultyTable.Columns             FacultyLabel(pos1).Text = row(column)             pos1 = pos1 + 1         Next     Next End Sub </pre>		

Figure 4.76. The coding for the subroutine FillFacultyTable.

```

FacultyForm MapFacultyTable

Private Sub MapFacultyTable(ByRef fLabel AsObject)
    fLabel(2) = OfficeLabel
    fLabel(3) = PhoneLabel
    fLabel(4) = CollegeLabel
    fLabel(5) = TitleLabel
    fLabel(6) = EmailLabel
    'The order must be identical with the order in
    'the query string – cmdString1
End Sub

```

**Figure 4.77.** The coding for the MapFacultyTable subroutine.

- C. Then a MapFacultyTable() subroutine is called to set the correct mapping relationship between each label object in the label array and the data retrieved from the DataTable.
- D. A double For...Next loop is utilized to assign the data read out from the DataTable to the mapped label 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 executed only once and the inner loop is executed seven times. Because the distribution order of the label controls on the Faculty form and the column order in the query string (cmdString1) are not identical, we need this MapFacultyTable() subroutine to align them.

A clearer picture of the MapFacultyTable() subroutine will be obtained after we do a detailed analysis of this item. The detailed coding for this subroutine is shown in Figure 4.77.

The order of the labels after the equal symbols is the order of the queried columns in the query string cmdString1 (columns 0 and 1 are faculty\_id and name), but the order of the five label controls on the Faculty form window is different. By performing this assignment, the five label controls on the Faculty form window have one-to-one relations with the queried columns in the Faculty table.

Now let's begin the coding for another subroutine, FillFacultyReader(). This subroutine is used to retrieve the queried data from the DataReader and distribute it to the five label controls in the Faculty form window. The coding for this subroutine is shown in Figure 4.78.

Let's see how this piece of code works.

- A. A loop counter intIndex is declared.
- B. Seven instances in the object array, or label array, are created and initialized. These seven objects are mapped to seven columns in the Faculty table in the database.
- C. The subroutine MapFacultyTable() is called to set up the correct mapping between the five label controls in the Faculty form window and five columns in the Faculty table in the database.
- D. A While loop is executed as long as the loop condition Read() method is True, which means that valid data is read out from the DataReader. This method will return a False if no valid data can be read out from the

The screenshot shows a Microsoft Visual Studio code editor window. The title bar indicates the project is 'FacultyForm' and the current file is 'FillFacultyReader'. The code itself is as follows:

```

PrivateSub FillFacultyReader(ByVal FacultyReader As OleDbDataReader)
    Dim intIndex As Integer
    For intIndex = 0 To 6           'Initialize the object array
        FacultyLabel(intIndex) = New Label()
    Next intIndex
    Call MapFacultyTable(FacultyLabel)
    While FacultyReader.Read()
        For intIndex = 0 To FacultyReader.FieldCount - 1
            FacultyLabel(intIndex).Text = FacultyReader.Item(intIndex).ToString()
        Next intIndex
    End While
End Sub

```

Figure 4.78. The coding for the subroutine FillFacultyReader.

DataReader, which means that all data has been read out. In this application, in fact, this While loop is only executed once since we have only one row (one record) read out from the DataReader.

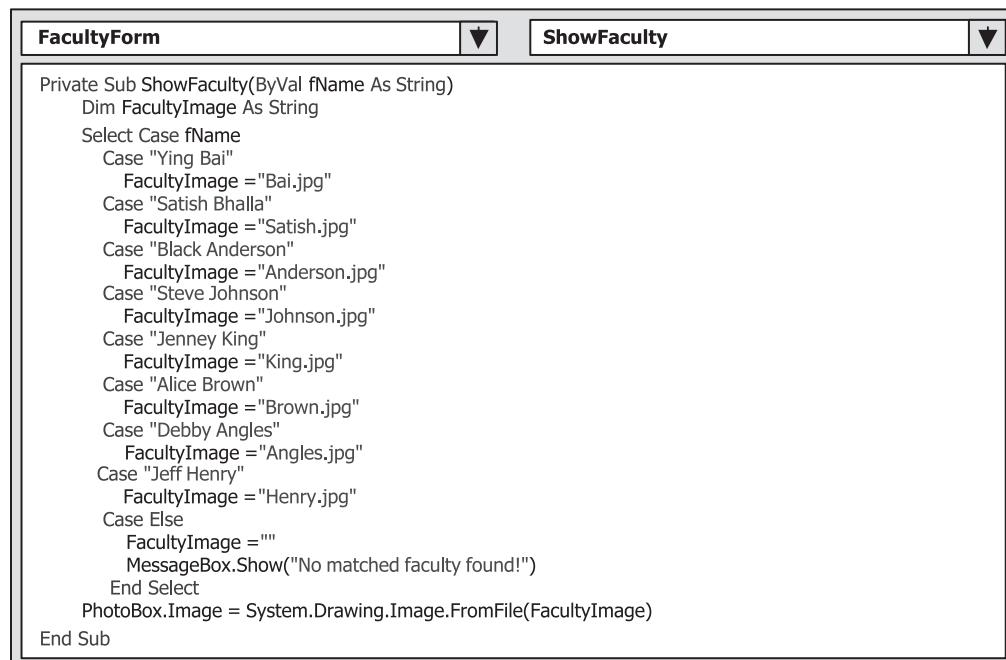
- E. A For...Next loop is utilized to pick up the data read out from the DataReader object, and the data is assigned to the associated label controls in the Faculty form window. Note that the first two columns (faculty id and name in the Faculty data table) are not used for this application. The Item property with the index is used here to identify the data from the DataReader.

The next subroutine for which we need to do coding is ShowFaculty().

This subroutine is used to identify the faculty photo based on the faculty name and to display the selected faculty photo in the PhotoBox control in the Faculty form window. The coding for this part is very similar to that in Section 4.13.2. 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 pay attention to is the location where the faculty images should be located. Of course you can store the faculty image files in any folder on your computer or in any server that is connected to a network with your computer. The point is that you must provide the full name of the faculty image, including the drive and path as well as the faculty image file name, to the system drawing method to perform the displaying. A convenient way to do this is to save the faculty image files in the folder in which your Visual Basic executable file is located. If you do this, you do not need to give the full location of the faculty image files, but only the name of the faculty image file. For example, in this application, our Visual Basic executable file is in the folder **Chapter 4\AccessSelectRTOObject Solution\AccessSelectRTOObject Project\bin\Debug**. So all faculty image files should be saved to this folder. The coding for this subroutine is shown in Figure 4.79.

Let's see how this piece of code works.

- A. A local string variable, FacultyImage, is declared and is used to hold the faculty image file.



```

FacultyForm ShowFaculty

Private Sub ShowFaculty(ByVal fName As String)
    Dim FacultyImage As String
    Select Case fName
        Case "Ying Bai"
            FacultyImage = "Bai.jpg"
        Case "Satish Bhalla"
            FacultyImage = "Satish.jpg"
        Case "Black Anderson"
            FacultyImage = "Anderson.jpg"
        Case "Steve Johnson"
            FacultyImage = "Johnson.jpg"
        Case "Jenney King"
            FacultyImage = "King.jpg"
        Case "Alice Brown"
            FacultyImage = "Brown.jpg"
        Case "Debby Angles"
            FacultyImage = "Angles.jpg"
        Case "Jeff Henry"
            FacultyImage = "Henry.jpg"
        Case Else
            FacultyImage = ""
            MessageBox.Show("No matched faculty found!")
    End Select
    PhotoBox.Image = System.Drawing.Image.FromFile(FacultyImage)
End Sub

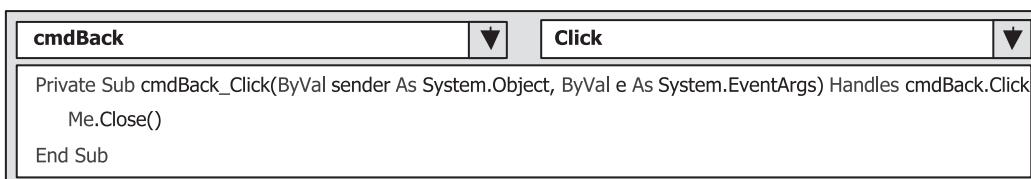
```

Figure 4.79. The coding for the subroutine ShowFaculty.

- B. A 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 executable file is located, only the faculty image file name is needed for the system drawing method to display the faculty image.
- C. If no matched faculty image file is found, a blank string is assigned to the FacultyImage variable and an error message is displayed.
- D. The system drawing method is executed to draw the faculty image based on the selected faculty image file name.

The last thing we need to do is the coding for the Back button event procedure. This coding is very easy. The current form window, the Faculty form, should disappear from the project if this button is clicked. The method Close() is suitable for this coding, shown in Figure 4.80.

At this point, we have finished the coding for the Faculty form. Let's test our project now. Click the Start button to run the project. Enter ybai and reback as the username and password in the LogIn form window, and then click the LogIn button



```

cmdBack Click

Private Sub cmdBack_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdBack.Click
    Me.Close()
End Sub

```

Figure 4.80. The coding for the cmdBack button event procedure.



Figure 4.81. The running status of the Faculty form.

to open the Selection form window. Make the default selection and click the OK button to open the Faculty form window, shown in Figure 4.81.

Select either the first or the second method from the Method combo box, choose the desired faculty member from the Faculty Name combo box, and click the Select button to perform the information query for the selected faculty member. Five label controls that are bound to the associated columns in the Faculty table are updated with the queried information, and the selected faculty image is also displayed in the PhotoBox control, which is shown in Figure 4.81. You can try selecting a different method by clicking the drop-down arrow on the Method combo box. Yes, the project works fine!

Our next job is to do the coding for the Course form.

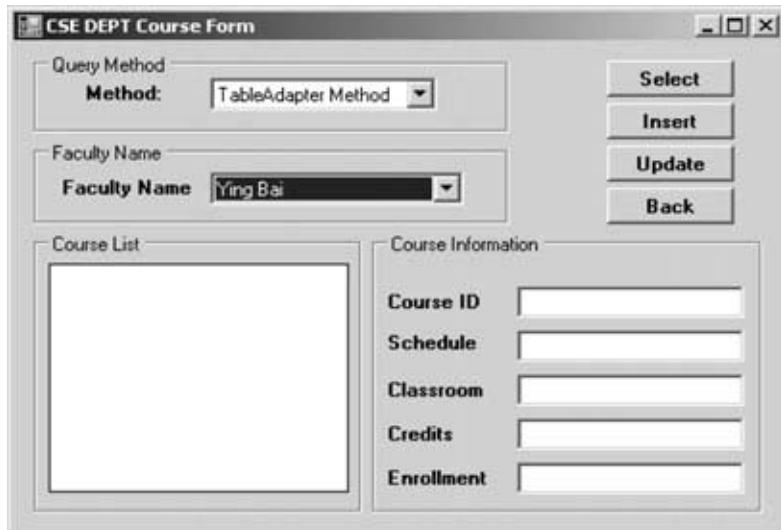
As we did for the Faculty form window, first we need to add a Method combo box to allow the user to select the data query method: TableAdapter or DataReader. You can copy this Method combo box from the Faculty form window and paste it into the Course form window.

#### 4.17.4 Query Data Using Runtime Objects for the Course Form

Open the Course form window by clicking the Course Form.vb item and the View Designer button from the Solution Explorer window. Copy both the Method label and the Method combo box from the Faculty form window and paste them onto the top of the Course form with a group box, which is shown in Figure 4.82.

The Text property of the group box is “Query Method”. Also add one more button and name it cmdUpdate with the Text property equaling “Update”, as shown in Figure 4.82. We will use this button to perform the data update query in the following section.

Let’s first do the coding for the Form\_Load event procedure. Open the code window by clicking the View Code button in the Solution Explorer window. Click the drop-down arrow on the Class Name combo box and select the CourseForm



**Figure 4.82.** The modified Course form window.

Events, and then click the drop-down arrow on the Method Name combo box and select the Load item to open its Form\_Load event procedure. Enter the code shown in Figure 4.83 into this event procedure.

Let's see how this piece of code works.

- This coding fragment is very similar to one we did for the Faculty form. The only difference is that the Label array has been replaced by a TextBox array

```
(CourseForm) ▾ Load ▾
Imports System.Data
Imports System.Data.OleDb
Public Class CourseForm
    Private CourseTextBox(4) As TextBox
    Private CourseID() As String
    Private Sub CourseForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Me.Load
        If LogInForm.accConnection.State <> ConnectionState.Open Then
            MessageBox.Show("Database has not been opened!")
            Exit Sub
        End If
        ComboBoxName.Items.Add("Ying Bai")
        ComboBoxName.Items.Add("Satish Bhalla")
        ComboBoxName.Items.Add("Black Anderson")
        ComboBoxName.Items.Add("Steve Johnson")
        ComboBoxName.Items.Add("Jenney King")
        ComboBoxName.Items.Add("Alice Brown")
        ComboBoxName.Items.Add("Debby Angles")
        ComboBoxName.Items.Add("Jeff Henry")
        ComboBoxName.SelectedIndex = 0
        ComboBoxMethod.Items.Add("TableAdapter Method")
        ComboBoxMethod.Items.Add("DataReader Method")
        ComboBoxMethod.SelectedIndex = 0
    End Sub

```

**Figure 4.83.** The coding for the CourseForm\_Load event procedure.

since we used five text box controls to display the detailed course information that is related to the selected faculty name from the Faculty Name combo box. The Course table has seven columns, but we only need five of them (refer to Figure 4.82), so the size of this TextBox array is 5, and each element or TextBox control in this array is indexed from 0 to 4.

- B. Recall that in the first project, SelectWizard, the contents displayed in the Course List box were the names of all the courses taught by the selected faculty member. Also, the detailed information related to the course name selected in the Course List box will be displayed in five text boxes when the user clicks one course name. Note that the names of the courses in the Course table are not unique, which means that one course may have two sections with the same name. For example, both CSC-132A and CSC-132B have the same course name, Introduction to Programming. An error may occur if one tries to use the course name as a criterion to query all detailed information related to that course since the course name is not unique in the Course table. In order to solve this problem, we must use the course\_id, which provides a unique value in the Course table, as a criterion to query the detailed information such as course name, credit, classroom, schedule, and enrollment for that course. Each course name displayed in the Course List box has a unique course\_id associated with it. So we need to use a form-level dynamic string array, CourseID(), to save the course\_id that is related to each course name when it is selected. This CourseID array will be used in the next event procedure, CourseList\_SelectedIndexChanged, which will be triggered when the user clicks the course name from the CourseList box. Since the number of courses taught by each faculty member is different and cannot be known at this moment, we declare this string array as a dynamic one – a blank parenthesis that has no dimension follows the array name CourseID.
- C. Before we can perform any data queries, we need to check whether a valid connection is still open. Since we created the connection instance in the LogIn form with the Public accessing mode, the class name LogIn must precede the connection object.
- D. This code is used to initialize the Faculty Name and Method Name combo boxes; the first item is selected as the default item for both combo boxes.

The next coding job is for the Select button. After the user selects the desired data query method from the Method combo box and the faculty name from the Faculty Name combo box, the Select button is used to trigger the event procedure to retrieve all courses taught by the selected faculty member.

One point you need to note is that two queries are required in this event procedure because there is no faculty name column available in the Course table. So we must first query the Faculty table to find the faculty\_id that is related to the faculty name selected by the user. The second query is used to find all course\_id and related courses taught by the selected faculty member from the Course table. The queried course names are displayed in the Course List box, and the detailed course information for each course can be displayed in five text boxes when the user clicks the associated course name in the Course List box.

Now return to the Course form window by clicking the View Designer button, and double-click the Select button to open its event procedure. Enter the code shown in Figure 4.84 into this procedure.

The coding of this part is very similar to what we did for the Select button event procedure in the Faculty form. Let's see how this piece of code works.

- A. The first query string is used to find the faculty\_id based on the faculty name from the Faculty table. The second query string is for the Course table. The course table has seven columns, but we only need five of them. There are six query items related to six columns: course\_id, course, credit, classroom, schedule, and enrollment. The course column contains the course names that will be displayed in the Course List box, and the other five items will be displayed in five text boxes as the detailed information for the selected course. The faculty\_id is still used as the criterion to query the desired course information for the selected faculty member. Other necessary instances are also created in this part to aid the data query task. Two TableAdapters, two Command objects, and two DataTable objects are declared to facilitate the two queries.
- B. Then the first Command object, accCmdFaculty, is initialized by assigning it with the connection instance, command type, and query string. The dynamic parameter Param1 is obtained from the Faculty Name combo box, in which the selected faculty name is entered by the user as the project runs. The completed Command object accCmdFaculty is assigned to the SelectCommand property of the FacultyTableAdapter, which is ready to make the 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.
- D. If the Fill is successful, one row of data is returned (exactly 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, rowFaculty. Since we only need the first column or the unique column from this queried row (refer to query string strFaculty), which is rowFaculty(0), this column is the faculty\_id, and it is assigned to the local string variable strFacultyID, which will be used later.
- E. Next, the Course Command object accCmdCourse is initialized, and the dynamic parameter faculty\_id is replaced by the real faculty\_id obtained above.
- F. As for the Faculty form, the user can choose between two methods, TableAdapter and DataReader. If the user selects 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.
- G. If this fill is successful, the Count property of the Course table should be greater than 0, which means that the table is filled at least by one row. The

**cmdSelect**

**Click**

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim strFaculty As String = "SELECT faculty_id FROM Faculty WHERE name=@Param1"
    Dim strCourseA As String = "SELECT course_id, course, credit, classroom, schedule, enrollment FROM Course "
    Dim strCourseB As String = "WHERE faculty_id=@Param2"
    Dim cmdString As String = strCourseA & strCourseB
    Dim CourseTableAdapter As New OleDbDataAdapter
    Dim FacultyTableAdapter As New OleDbDataAdapter
    Dim accCmdFaculty, accCmdCourse As New OleDbCommand
    Dim accCourseTable, accFacultyTable As New DataTable
    Dim accDataReader As OleDbDataReader
    Dim strFacultyID As String
    Dim rowFaculty As DataRow

    accCmdFaculty.Connection = LogInForm.accConnection           'Initialize faculty Command object
    accCmdFaculty.CommandType = CommandType.Text
    accCmdFaculty.CommandText = strFaculty
    accCmdFaculty.Parameters.Add("@Param1", OleDbType.Char).Value = ComboName.Text
    FacultyTableAdapter.SelectCommand = accCmdFaculty
    FacultyTableAdapter.Fill(accFacultyTable)                   'Execute the first query
    If accFacultyTable.Rows.Count = 0 Then
        MessageBox.Show("No matched faculty_id found!")
        Exit Sub
    End If
    rowFaculty = accFacultyTable.Rows.Item(0)                  'Get faculty_id
    strFacultyID = rowFaculty(0)

    accCmdCourse.Connection = LogInForm.accConnection          'Initialize course Command object
    accCmdCourse.CommandType = CommandType.Text
    accCmdCourse.CommandText = cmdString
    accCmdCourse.Parameters.Add("@Param2", OleDbType.Char).Value = strFacultyID

    If ComboMethod.Text = "TableAdapter Method" Then
        CourseTableAdapter.SelectCommand = accCmdCourse
        CourseTableAdapter.Fill(accCourseTable)                 'Execute the second query
        If accCourseTable.Rows.Count > 0 Then
            Call FillCourseTable(accCourseTable)
        Else
            MessageBox.Show("No matched course found!")
        End If
    Else
        accFacultyTable.Dispose()
        accFacultyTable = Nothing
        accCourseTable.Dispose()
        accCourseTable = Nothing
        CourseTableAdapter.Dispose()
        CourseTableAdapter = Nothing
    End If

    accDataReader = accCmdCourse.ExecuteReader
    If accDataReader.HasRows = True Then
        Call FillCourseReader(accDataReader)
    Else
        MessageBox.Show("No matched course found!")
    End If
    accDataReader.Close()
    accDataReader = Nothing
End If

accCmdFaculty.Dispose()
accCmdFaculty = Nothing
accCmdCourse.Dispose()
accCmdCourse = Nothing
CourseList.SelectedIndex = 0
End Sub

```

Figure 4.84. The coding for the cmdSelect button event procedure.

subroutine FillCourseTable() will be called with the filled table as the argument to fill the Course List box control with the course\_id on the Course form. Otherwise, if this Count is equal to 0, which means that no row or record has been filled into the Course table, an error message will be displayed for this situation.

- H. Some necessary cleaning jobs are performed to release objects we used for this data query.
- I. If the user selected the DataReader method, the ExecuteReader() method is called to perform a read-only operation to the database.
- J. If the HasRows property of the DataReader is True, which means that the DataReader did receive some data, the FillCourseReader() subroutine is executed with the DataReader as the argument to fill the Course List box control in the Course form window. An error message will be displayed if the HasRows property is not True.
- K. Finally, the DataReader and the Command objects are released.
- L. This coding is very important and is used to select the first course name as the default one from the Course List box. This coding is used to trigger the CourseList\_SelectedIndexChanged() event procedure to display detailed information for the selected course name in five text boxes. Without this default course selected, no detailed course information would be displayed when the CourseList\_SelectedIndexChanged() event procedure is executed for the first time.

Now let's take a look at the coding for two subroutines used in this part: FillCourseTable() and FillCourseReader(). These two subroutines are used to fill the Course List box control in the Course form window by using the queried data. Figure 4.85 shows the coding for these two subroutines.

Let's see how this piece of code works.

- A. A local DataRow object is created. In order to access the DataTable to pick up data, one must scan the table by using either DataRow or DataColumn objects, and in most cases, one needs both of them. The DataRow and the DataColumn classes are included in the DataRowCollection and the DataColumnCollection classes, respectively. Although these two objects work like integers, one must use these two objects to access the DataTable to set, retrieve, and manipulate the data to and from the DataTable. In this application, we only need to pick up the data located in the first column (course\_id) and the second column (course) of the Course table, so we do not need to use the DataColumn object.
- B. A local integer variable, pos, is declared here and is used as a loop counter later to store the course\_id value into the CourseID array.
- C. Recall that the CourseID is a form-level dynamic string array and its dimension cannot be determined when it is declared since different faculty members teach different numbers of courses. But this number can be determined after this Course table query. The total number of rows returned from this query is the number of courses taught by the selected faculty member. So now the dynamic string array CourseID can be redefined

```

A Private Sub FillCourseTable(ByVal CourseTable As DataTable)
B     Dim row As DataRow
C     Dim pos As Integer
D     ReDim CourseID(CourseTable.Rows.Count) ' redefine the CourseID array with the definite demission
E     CourseList.Items.Clear()
F     For Each row In CourseTable.Rows
G         CourseList.Items.Add(row(1))           ' the 2nd column is course- strCourseA
H         CourseID(pos) = row(0)             ' the 1st column is course_id- strCourseA
I         pos = pos + 1
J     Next
K End Sub
L Private Sub FillCourseReader(ByVal CourseReader As OleDbDataReader)
M     Dim strCourse As String
N     Dim pos As Integer
O     ReDim CourseID(9)                  ' redefine the CourseID array with the definite demission
P     CourseList.Items.Clear()
Q     While CourseReader.Read()
R         strCourse = CourseReader.GetString(1)          ' the 2nd column is course
S         CourseList.Items.Add(strCourse)
T         CourseID(pos) = Convert.ToString(CourseReader.GetString(0)) ' the 1st column is course_id
U         pos = pos + 1
V     End While
W End Sub

```

**Figure 4.85.** The coding for two subroutines.

by using the number of courses (returned rows), which is equal to the CourseTable.Rows.Count property.

- D. Before we can fill the Course List box, a cleaning job is needed. This cleaning is very important; multiple repeated courses would be displayed in this list box if you forgot to clean it up first.
- E. A For Each loop is used to scan all rows of the filled Course table. Recall that we filled six columns' data from the Course table in the database to this Course table in the DataTable object starting with the first column, course\_id, and the second column, course (refer to the query string strCourseA defined in the cmdSelect button event procedure, Figure 4.84). Now we need to pick up both the first column's data, course\_id (column index = 0), and the second column's data, course (column index = 1), for each returned row or record of the Course table. Then the Add() method of the ListBox control is used to add each retrieved second column's data (row(1) = course) into the Course List box and save each retrieved first column's data (row(0) = course\_id) into the CourseID array with an appropriate index (pos).
- F. The loop counter pos, which also works as an index for the CourseID array, is updated by 1.
- G. For the FillCourseReader() subroutine, a local string variable strCourse is created. This variable can be considered an intermediate variable that is used to temporarily hold the queried data from the Course table. A local integer variable, pos, is created and works as a loop counter later for the While loop and the index for the CourseID array.

- H. The dynamic string array CourseID is redefined here by a definite dimension number, 9. Since the maximum number of courses taught by any faculty member in this department is less than 10, a dimension of 10 is defined to allow the string array to store up to 10 course\_id. This number can be modified based on your real project.
- I. Similarly, we need to clean up the Course List box before it can be filled.
- J. A While loop is used to retrieve each second column's data (GetString(1)), whose column index is 1 and data value is the course. The queried data is first assigned to the intermediate variable strCourse and then added into the Course List box by using the Add() method. Also, each retrieved first column's data (GetString(0)), whose column index is 0 and data value is the course\_id, is stored into the string array CourseID with an appropriate index (pos). The index pos is updated by 1 for the next loop.

Next, we need to take care of the coding for the CourseList\_SelectedIndexChanged() event procedure. The function of this procedure is to display the information related to the selected course from the Course List box, which includes the course ID, classroom, schedule, credit, and enrollment, and these pieces of information will be displayed in five text box controls on the Course form window. This event procedure is triggered when the user clicks a course from the Course List box.

Open the Course form window by clicking the View Designer button from the Solution Explorer window, and then double-click the Course List box to open its CourseList\_SelectedIndexChanged() event procedure. Enter the code shown in Figure 4.86 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 see how this piece of code works.

- A. The query string is defined with five 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 that information in five text box controls, one of which is the course\_id. This course\_id is stored in the form-level string array CourseID that we created before. In order to make this query neat, we still add the course\_id column into the query string even if it is available from the CourseID array. Also, some necessary objects are created and the command object is initialized here. The dynamic parameter now is the course\_id.
- B. The dynamic parameter course\_id is now replaced by the real parameter CourseID(CourseList.SelectedIndex). Recall that we stored the associated course\_id into the string array CourseID in the FillCourseTable and FillCourseReader event procedures (refer to Figure 4.85). The value of the CourseList.SelectedIndex is the position number of each course displayed in the Course List box, and this number is identical to the order number in which we added the courses into the Course List box and saved the associated course\_id into the array CourseID in the FillCourseTable and FillCourse Reader event procedures (Figure 4.85).

	<b>CourseList</b>	<b>SelectedIndexChanged</b>
A	<pre>Private Sub CourseList_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) _ Handles CourseList.SelectedIndexChanged</pre>	
B	<pre>    Dim cmdString1 As String = "SELECT course_id, credit, classroom, schedule, enrollment FROM Course "     Dim cmdString2 As String = "WHERE course_id=@Param1"     Dim cmdString As String = cmdString1 &amp; cmdString2     Dim CourseTableAdapter As New OleDbDataAdapter     Dim accCommand As New OleDbCommand     Dim accDataReader As OleDbDataReader     Dim accDataTable As New DataTable     accCommand.Connection = LogInForm.accConnection     accCommand.CommandType = CommandType.Text     accCommand.CommandText = cmdString     accCommand.Parameters.Add("@Param1", OleDbType.Char).Value = CourseID(CourseList.SelectedIndex)</pre>	
C	<pre>If ComboMethod.Text = "TableAdapter Method" Then</pre>	
D	<pre>    CourseTableAdapter.SelectCommand = accCommand     CourseTableAdapter.Fill(accDataTable)     If accDataTable.Rows.Count &gt; 0 Then         Call FillCourseTextBox(accDataTable)     Else</pre>	
E	<pre>        MessageBox.Show("No matched course information found!")     End If</pre>	
F	<pre>    accDataTable.Dispose()     accDataTable = Nothing     CourseTableAdapter.Dispose()     CourseTableAdapter = Nothing</pre>	
G	<pre>Else     accDataReader = accCommand.ExecuteReader     If accDataReader.HasRows = True Then         Call FillCourseReaderTextBox(accDataReader)     Else</pre>	
H	<pre>        MessageBox.Show("No matched course information found!")     End If</pre>	
	<pre>    accDataReader.Close()     accDataReader = Nothing     End If     accCommand.Dispose()     accCommand = Nothing</pre>	
	<pre>End Sub</pre>	

**Figure 4.86.** The coding for the CourseList SelectedIndexChanged event procedure.

- C. If the user selects 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 has been stored in the Course table. Next, the FillCourseTextBox() subroutine is executed with the DataTable as the argument to fill five text box 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 objects used, which include the Data-Table and the TableAdapter.
- F. If the user selected the DataReader method, the ExecuteReader() method is executed to perform a read-only operation to retrieve the information related to the course selected in the Course List box.

	<b>CourseForm</b>		<b>FillCourseTextBox</b>	
--	-------------------	--	--------------------------	--

```

Private Sub FillCourseTextBox(ByVal CourseTable As DataTable)
    Dim pos1 As Integer
    Dim row As DataRow
    Dim column As DataColumn
    For pos2 As Integer = 0 To 4           'Initialize the object array
        CourseTextBox(pos2) = New TextBox
    Next pos2
    Call MapCourseTable(CourseTextBox)
    For Each row In CourseTable.Rows
        For Each column In CourseTable.Columns
            CourseTextBox(pos1).Text = row(column)
            pos1 = pos1 + 1
        Next
    Next
End Sub

Private Sub FillCourseReaderTextBox(ByVal CourseReader As OleDbDataReader)
    Dim intIndex As Integer
    For intIndex = 0 To 4                 'Initialize the object array
        CourseTextBox(intIndex) = New TextBox
    Next intIndex
    Call MapCourseTable(CourseTextBox)
    While CourseReader.Read()
        For intIndex = 0 To CourseReader.FieldCount - 1
            CourseTextBox(intIndex).Text = CourseReader.Item(intIndex).ToString
        Next intIndex
    End While
End Sub

```

**Figure 4.87.** The coding for two subroutines.

- G. If this read-only operation is successful, the HasRows property of the DataReader will be True, and the subroutine FillCourseReaderTextBox() is called to fill five text box 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 objects used for this data query.

The code for two subroutines, FillCourseTextBox() and FillCourseReaderTextBox(), is shown in Figure 4.87.

Let's see how this piece of code works.

- A. As mentioned in the coding 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 – is created and initialized here. Any object or object array should be created by using the New operator. Here five text box objects are created and can be mapped to five text box controls in the Course form window. We use these five text box objects later to display the detailed information for the course selected in the Course List box.
- C. The subroutine MapCourseTable() is executed to set up a one-to-one relation between each text box control on the Course form window and each

queried column in the query. This step is necessary since the distribution order of the five text box controls on the Course form differs from 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 executed only once since we query only one record's (one row's) course information based on the selected course\_id from the Course data table. The inner loop is executed exactly five times to pick up five pieces of course-related information, the course title, classroom, credit, schedule, and enrollment. Then the retrieved information is assigned to each text box control in the text box array, which will be displayed in that text box control.
- E. For the subroutine FillCourseReaderTextBox(), a loop counter intIndex is first created and is used for the loop of creation of the text box object array and the loop of retrieving data from the DataReader later.
- F. This loop is used to create the text box object array and perform the initialization for those objects.
- G. The functionality of this step is the same as described in step C.
- H. A While loop and a For...Next loop are used to pick up all five pieces of course-related information from the DataReader one by one. The Read() method is used as the While loop condition. A returned True means that 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 a FieldCount of 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 item is converted to a string and assigned to the associated text box control in the text box object array.

The coding for the subroutine MapCourseTable() is shown in Figure 4.88.

This coding is straightforward. The order of the text boxes after the equal operators is the column order of the query string cmdString1. By assigning each text box control on the Course form window to its partner, the text box in the text box object array, in this order, a one-to-one relationship is built, and the data retrieved from the DataReader can be mapped exactly to the associated text box control and can be displayed there.

The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "CourseForm" and "MapCourseTable". The code is as follows:

```
Private Sub MapCourseTable(ByRef fCourse As Object)
    fCourse(0) = txtID
    fCourse(1) = txtCredits
    fCourse(2) = txtClassRoom
    fCourse(3) = txtSchedule
    fCourse(4) = txtEnroll
End Sub
```

A note in the code states: "The order must be identical with the column order in the query string cmdString1 in CourseList\_SelectedIndexChanged procedure".

Figure 4.88. The coding 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 4.89. The coding for the cmdBack button event procedure.

The last coding job is for the cmdBack button event procedure. This coding is very simple and is shown in Figure 4.89.

Now let's test our project by clicking the Start button. Enter the username and password as we did before, and select Course Information from the Selection form window to open the Course form window, shown in Figure 4.90.

Select any method you want by clicking the drop-down arrow on the Method combo box, and then select your desired faculty name from the Faculty Name combo box. Click the Select button, and all courses taught by that faculty member will be displayed in the Course List box, shown in Figure 4.90. Then select any course by clicking on it in the Course List box, and all detailed information related to the selected course will be displayed in five text box controls.

What fun!

There is another version of the Course form, in which the contents displayed in the Course List box are not the queried courses, but the course\_id. By clicking on each course\_id in the CourseList box, the detailed information related to that selected course\_id will be displayed in five text boxes, which include course title, credit, classroom, schedule, and enrollment. You can find this version of the Course form with the detailed coding in a project available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 4\Access-SelectRTOBJECT\_Course**.

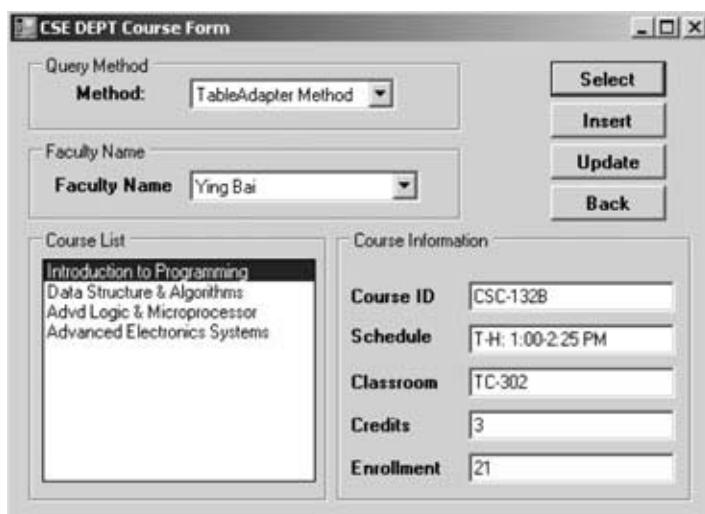


Figure 4.90. The running status of the Course form window.

#### 4.17.5 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 students' information to be reviewed in detail. This information includes 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 Course List box and all other information is displayed in five text boxes when the Select button is clicked.

The coding for this form is a little different since two data tables are utilized for this form, Student and StudentCourse. The first table contains the student's general information, and the second contains all courses taken by the student. So two TableAdapters need to be created and used. Also, two different data queries are needed. The first one is used to retrieve the general student information from the Student table, and the second is used to pick up information on the courses taken by the student from the StudentCourse table.

In order to save space, only one query method, the TableAdapter query method, is provided in this section. The DataReader query method has been left as homework.

The coding job is divided into two major event procedures: StudentForm\_Load and cmdSelect\_Click. The first is used to initialize the Student form and display all the students' names in a combo box so that they can be selected by the user. The second executes the data queries to pick up the selected student's information and display it in the associated text box controls and the list box control.

You need to include the necessary data library paths, which include Imports System.Data and Imports System.Data.OleDb, at the top of the code window of this form before you can use any component in ADO.NET.

##### 4.17.5.1 Coding for the Student Form\_Load Event Procedure

The coding for this event procedure is shown in Figure 4.91. Let's take a look at the coding.

```
(StudentForm Events) ▾ Load ▾
Imports System.Data
Imports System.Data.OleDb
Public Class StudentForm
Private StudentTextBox(5) As TextBox
    'We query 6 columns from the Student table
Private Sub StudentForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ComboBox1.Items.Add("Erica Johnson")
    ComboBox1.Items.Add("Ashly Jade")
    ComboBox1.Items.Add("Holes Smith")
    ComboBox1.Items.Add("Andrew Woods")
    ComboBox1.Items.Add("Blue Valley")
    ComboBox1.SelectedIndex = 0
End Sub
```

Figure 4.91. The Student Form\_Load event procedure.

- A. A text box array, StudentTextBox(), is created here, and this object array is used to set up a bridge between the six text boxes in the Student form and six query columns in the query string strStudent: student\_id, gpa, credits, major, school year, and email.
- B. Five students' names are added into the Student combo box. As the project runs, the user can select any student by clicking the name to review the detailed information for the selected student in six text boxes and all courses taken by that student in the Course List box.
- C. The first student's name is selected as the default by setting the SelectedIndex property value as 0.

#### 4.17.5.2 Coding for the Student Select Button Event Procedure

As the project runs, the user selects a student's name from the student name combo box and clicks the Select button. Detailed information about the selected student is displayed in six text boxes, and this information is located in the Student table in the CSE\_DEPT database. Also, all courses taken by that student are displayed in the Course List box, and this information is stored in the StudentCourse table. So this event procedure needs to perform two queries with two different tables.

The coding for this event procedure is shown in Figure 4.92. Let's take a close look at this piece of code.

- A. The query string for the Student table is declared, and the string contains six columns in the Student data table, which are student\_id, gpa, credits, major, school year, 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 the ampersand operator is used to concatenate them to form a complete query string.
- B. The second string, or the query string for the StudentCourse table, is created and only one column is queried, which is course\_id. The query criterion is the student\_id.
- C. All data operation objects are created here, such as the Student and the StudentCourse TableAdapters, Commands, and DataTables. The string variable strName is used to hold the student's photo returned by calling the function FindName().
- D. The FindName() function is called to get the appropriate student photo based on the student's 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 function 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, accCmd-Student, is a reference (ByRef), which is equivalent to a memory address or a pointer variable in C++, from the function BuildCommand() protocol later. So when the function is done, the built command object is still stored in that reference and can be used without a problem. The dynamic parameter "name" is replaced by the real student name obtained from the student

A	cmdSelect
B	Click
C	<pre>PrivateSub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click     Dim strStudent1 As String = "SELECT student_id, gpa, credits, major, schoolYear, email FROM Student "     Dim strStudent2 As String = "WHERE name=@Param1"     Dim strStudent As String = strStudent1 &amp; strStudent2     Dim strStudentCourse As String = "SELECT course_id FROM StudentCourse WHERE student_id=@Param2"     Dim StudentTableAdapter As New OleDbDataAdapter     Dim StudentCourseTableAdapter As New OleDbDataAdapter     Dim accCmdStudent, accCmdStudentCourse As New OleDbCommand     Dim accStudentTable, accStudentCourseTable As New DataTable     Dim strName As String      strName = FindName(ComboName.Text)     If strName = "No Match" Then         MessageBox.Show("No Matched Student Found!")         Exit Sub     End If     PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage     PhotoBox.Image = System.Drawing.Image.FromFile(strName)     Call BuildCommand(accCmdStudent, strStudent)           'Initialize the Student Command object     accCmdStudent.Parameters.Add("@Param1", OleDbType.Char).Value = ComboName.Text     StudentTableAdapter.SelectCommand = accCmdStudent     StudentTableAdapter.Fill(accStudentTable)             'Execute the first query     If accStudentTable.Rows.Count &gt; 0 Then         Call FillStudentTextBox(accStudentTable)     Else         MessageBox.Show("No matched student found!")     End If     Call BuildCommand(accCmdStudentCourse, strStudentCourse) 'Initialize the StudentCourse Command object     accCmdStudentCourse.Parameters.Add("@Param2", OleDbType.Char).Value = txtID.Text     StudentCourseTableAdapter.SelectCommand = accCmdStudentCourse     StudentCourseTableAdapter.Fill(accStudentCourseTable)   'Execute the second query     If accStudentCourseTable.Rows.Count &gt; 0 Then         Call FillCourseList(accStudentCourseTable)     Else         MessageBox.Show("No matched course_id found!")     End If     accStudentTable.Dispose()     accStudentTable = Nothing     accStudentCourseTable.Dispose()     accStudentCourseTable = Nothing     StudentTableAdapter.Dispose()     StudentTableAdapter = Nothing     StudentCourseTableAdapter.Dispose()     StudentCourseTableAdapter = Nothing     accCmdStudent.Dispose()     accCmdStudent = Nothing     accCmdStudentCourse.Dispose()     accCmdStudentCourse = Nothing End Sub</pre>
D	
E	
F	
G	
H	
I	
J	

Figure 4.92. The coding for the Student Select button event procedure.

name combo box, and the completed Command object is assigned to the SelectCommand property of the TableAdapter.

- G. The Fill() method is called to fill the Student table. By checking the Count property, we can determine 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 six text boxes in the Student form with the detailed student information, such as student\_id,

- gpa, credits, major, school year, and email, which is stored in the filled Student table. Otherwise, an error message is displayed.
- H. To make the second query to the StudentCourse table to find all courses taken by the selected student, the BuildCommand() is called again to initialize 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 text box txtID. The completed StudentCourse Command object is assigned to the SelectCommand property of the StudentCourse TableAdapter.
  - 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 the courses (exactly matching course.id) stored in the filled StudentCourse table into the Course List 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. A cleaning job is performed to release all objects used by this event procedure.

Now let's do the coding for the subroutine BuildCommand(). The coding is shown in Figure 4.93.

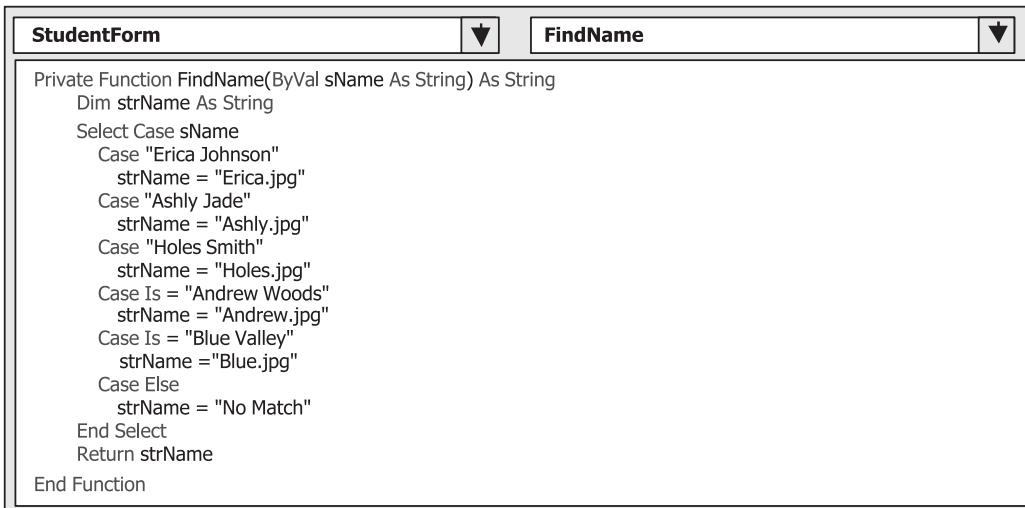
This coding is straightforward. The different properties of the Command class such as the Connection string, Command type, and Command text are assigned to the Command object. The only point to note is the data type of the first argument, cmdObj, which is a reference (ByRef), as mentioned above (refer to step F in Figure 4.92). A reference in Visual Basic.NET is equivalent to a memory address or a pointer in C++, and the argument cmdObj is called passing by reference. When the argument is passing in this mode, both the input and the built or output object cmdObj will be stored at the same address when this subroutine is completed, and we can use this built cmdObj without the need to return this cmdObj object from the subroutine.

The function FindName() is similar to the one in the Faculty form, and the coding for this function is shown in Figure 4.94.

A Select Case structure is used to select the desired student's photo based on the input, the 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 on a server, but you must provide the full name for these photos and assign it to the strName variable to be returned. The full name includes the machine name, driver name, and folder name, as well as the photo file name.

StudentForm	▼	BuildCommand	▼
<pre>Private Sub BuildCommand(ByRef cmdObj As OleDbCommand, ByVal cmdString As String)     cmdObj.Connection = LogInForm.accConnection     cmdObj.CommandType = CommandType.Text     cmdObj.CommandText = cmdString End Sub</pre>			

Figure 4.93. The BuildCommand subroutine.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "StudentForm". The code is for a function named "FindName". It uses a Select Case statement to map student names to photo file names. If no match is found, it returns "No Match".

```

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

```

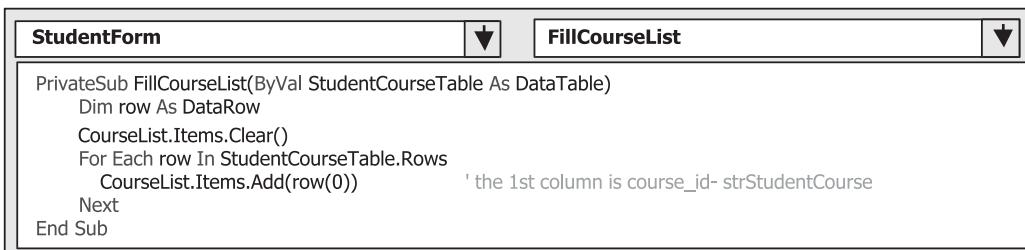
**Figure 4.94.** The FindName function.

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 executable file AccessSelectRTOBJECT.exe is located in the folder **C:\Chapter 4\AccessSelectRTOBJECT\bin\Debug**. When saving all student photos in this folder, you don't need to provide the full name for those photos; you only need to provide the photo name and assign it to the variable strName. That is much easier!

For other functions or subroutines used in this form, such as FillCourseList(), FillStudentTextBox(), and MapStudentTextBox(), the coding is similar to what we developed in the Course form. For your convenience, we list them here again with some simple explanations.

First is the coding for the FillCourseList() subroutine. The coding of this subroutine is shown in Figure 4.95.

The function of this subroutine is to fill the Course List box with all courses taken by the selected student. Those queried courses are stored in the StudentCourse table and 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 can be used to hold each row or record queried from the StudentCourse table. After the Course List box is cleared, a For Each loop is executed to pick up each row from the StudentCourse table, and the first column,



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "StudentForm". The code is for a subroutine named "FillCourseList". It starts by clearing a list box named "CourseList". Then it loops through each row in a DataTable named "StudentCourseTable", adds the first column value (course\_id) to the list box, and adds a comment indicating the first column is course\_id. Finally, it ends the subroutine.

```

PrivateSub FillCourseList(ByVal StudentCourseTable As DataTable)
    Dim row As DataRow
    CourseList.Items.Clear()
    For Each row In StudentCourseTable.Rows
        CourseList.Items.Add(row(0))      ' the 1st column is course_id- strStudentCourse
    Next
End Sub

```

**Figure 4.95.** The coding for the FillCourseList subroutine.

**StudentForm**

**FillStudentTextBox**

```
Private Sub FillStudentTextBox(ByVal StudentTable As DataTable)
    Dim pos1 As Integer
    Dim row As DataRow
    Dim column As DataColumn
    For pos2 As Integer = 0 To 5
        StudentTextBox(pos2) = New TextBox           'Initialize the textbox array
    Next pos2
    Call MapStudentTextBox(StudentTextBox)
    For Each row In StudentTable.Rows
        For Each column In StudentTable.Columns
            StudentTextBox(pos1).Text = row(column)
            pos1 = pos1 + 1
        Next
    Next
End Sub
```

Figure 4.96. The coding for the FillStudentTextBox subroutine.

which is `row(0)` or the `course_id`, is added into the Course List box by executing the `Add` method.

The next one is the subroutine `FillStudentTextBox()`, and the coding for this subroutine is shown in Figure 4.96.

The function of this piece of coding is to fill six text boxes in the Student form with six columns' data obtained from the Student table: `student_id`, `gpa`, `credits`, `major`, `school_year`, 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 relationship between the `StudentTextBox` array and six text boxes in the Student form. A nested `For Each` loop is executed to pick up each column's data from the queried row. Exactly one row of data that matches the selected student name is obtained from the Student table; therefore, the outer loop is only executed once. The reason for using a double loop is that both the `DataRow` and the `DataTable` are classes, and in order to pick up data from any `DataTable`, one must use the objects `row` and `column`, which are instances or objects of the `DataRow` and `DataColumn`, as the index to access each row or column of `DataTable` instead of using an integer. The local integer variable `pos1` works as an index for the `StudentTextBox` array.

The coding for the subroutine `MapStudentTextBox()` is shown in Figure 4.97.

**StudentForm**

**MapStudentTextBox**

```
Private Sub MapStudentTextBox(ByRef sTextBox As Object)
    sTextBox(0) = txtID          'The order must be identical with the
    sTextBox(1) = txtGPA         'order in the query string - strStudent
    sTextBox(2) = txtCredits
    sTextBox(3) = txtMajor
    sTextBox(4) = txtSchoolYear
    sTextBox(5) = txtEmail
End Sub
```

Figure 4.97. The coding for the subroutine `MapStudentTextBox`.

The purpose of this coding is to set up a one-to-one relationship between each text box control in the StudentTextBox array and each column data in our first query string, strStudent. Each text box control in the StudentTextBox array is related to an associated text box control in the Student form, such as student\_id, gpa, credits, major, school year, and email. Since the distribution order of the text boxes in the StudentTextBox array may differ from the order of the column data in our first query, a correct order relationship can be set up after calling this subroutine.

Another important point to note is the data type of the argument TextBox, which is a nominal reference variable of the StudentTextBox array. A reference data type (ByRef) must be used for this argument in order to use the modified text box controls in the StudentTextBox array when this subroutine returns to our main procedure.

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 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 that all student photo files have been stored in the same folder in which your Visual Basic.NET executable file is located. Click the Start Debugging button to run the project. Enter a suitable user-name and password, such as jhenry and test, for the LogIn form, and click the Students Information item from the Selection form to open the Student form window, which is shown in Figure 4.98.

Select a student name, such as Ashly Jade, from the Student Name combo box, and then click the Select button. All courses taken by this student are shown in the Course List box and the detailed information about this student is displayed in six text boxes.

The completed project, including the graphical user interfaces – five form windows – and related coding, is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 4\AccessSelectRTOBJECT**.



Figure 4.98. The running status of the Student form.

Next, we want to show readers how to develop a professional data-driven project using runtime objects for an SQL Server database.

## **4.18 BUILD AN SQL SERVER DATABASE PROJECT – SQLSELECTRTOBJECT**

In the previous section you learned how to build a data-driven application using runtime objects for the Microsoft Access database. Microsoft Access is a very good candidate when a small group of users with small amounts of data is 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 different data providers to access different databases, and 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.

### **4.18.1 Migrating from Access to SQL Server and Oracle Databases**

Basically similar runtime objects and structures are used to develop a data-driven project that can access 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 different prefixes to specify what kind of data provider is utilized for certain databases. The prefix “Sql” would be used if an SQL Server data provider is utilized, for example, SqlConnection, SqlCommand, SqlDataAdapter, and SqlDataReader. The same thing is done for the Oracle data provider.

So the difference between the data-driven applications that can access the different databases is in the data provider-dependent components. Among them, the Connection String is a big issue. Different data providers need to use different connection strings to make a connection to the associated database.

Normally, a Connection String is composed of five parts:

- Provider
- Data Source
- Database
- User ID
- Password

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

---

```
Connection = New xxxConnection("Provider = MyProvider;" &_
"Data Source = MyServer;" &_
"Database = MyDatabase;" & _
```

---

```
"User ID = MyUserID;" &_
"Password = MyPassWord;"
```

---

where **xxx** would be replaced by the selected data provider, such as OleDb, Sql, or Oracle, in your real application. You need to use the real parameter values implemented in your applications to replace the nominal values, such as MyServer, MyDatabase, MyUserID, and MyPassWord.

The Provider parameter indicates the database driver you selected. If you installed a local SQL server and client such as the SQL Server 2005 Express on your computer, the provider should be *localhost*. If you are using a remote SQL Server instance, you need to use that remote server's network name. If you are using the default named instance of SQLX on your computer, you need to use *.\SQLEXPRESS* as the value for your provider parameter. Similar values can be used for the Oracle server database.

The Data Source parameter indicates the name of the network computer on which your SQL server or Oracle server is installed and running. The Database parameter indicates your database name. The User ID and Password parameters are used for security issues for your database. In most cases, the default Windows NT Security Authentication is utilized.

You can also use OLE DB as the SQL Server database provider. A sample connection string to connect to a SQL Server database using the OLE DB data provider can be expressed as follows:

---

```
Connection = New OleDbConnection("Provider = SQLOLEDB;" &_
"Data Source = MyServer;" &_
"Database = CSE_DEPT;" &_
"User ID = MyUserID;" &_
"Password = MyPassWord;"
```

---

You need to use the real parameter values implemented in your applications to replace the nominal values such as MyServer, MyUserID, and MyPassWord in your application.

When you want to connect to an SQL Server database using *SqlClient*, the connection string is a little different from those strings shown above. The Provider parameter should be replaced by the Server parameter, and the User ID and Password parameters should be replaced by the Integrated Security parameter. A sample connection string to be used to connect to an SQL Server database using the *SqlClient* is

---

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

---

where the value for the Data Source parameter is *Computer Name|SQL Server 2005 Express name* since we installed the Express version of the SQL 2005 Server on our local computer. Also, we installed the SQL 2005 Client on the same computer to make it work as both a server and a client.

When you build a connection string to be used by an Oracle database using the OLE DB 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 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 Oracle is installed and running. This information is included in the tnsnames.ora file, too.

A sample connection string to connect to an Oracle database using the OLEDB provider can be expressed as follows:

---

```
Connection = New OleDbConnection("Provider = MSDAORA;" &_
    "Data Source = MySID;" &_
    "User ID = MyUserID;" &_
    "Password = MyPassWord;")
```

---

You need to use the real parameter values implemented in your applications to replace the nominal values such as MySID, MyUserID, and MyPassWord in your application.

To build a connection string to be used by an Oracle database using the Oracle-Client, you should know that most of parameters are included in the tnsnames.ora file, and an Oracle connection string is inseparable from an 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 the following:

---

```
OraDb = New OracleConnection("Data Source = OraDb;" +
    "User ID = MyUserID;" +
    "Password = MyPassWord;")
```

---

We discussed the development of data-driven applications using the OLE DB data provider in the last section. In the following sections, we will see how to develop professional data-driven applications connecting to SQL Server or Oracle databases using different data providers. We will discuss the data query first for the SQL Server database and then for the Oracle database.

In this section, we use an SQL Server 2005 Express database and connect it with our example project using the SQL Server data provider. The SQL Server database file used in this sample project is CSE\_DEPT.mdf, which was developed in Chapter 2 and is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **database\SQLServer**. The advantages of using the Express version of SQL Server 2005 include, but are not limited to, the following:

- SQL Server 2005 Express is fully compatible with SQL Server 2005 database and has full functionalities of the latter.
- SQL Server 2005 Express can be easily downloaded from the Microsoft site free of charge.
- SQL Server Management Studio Express 2005 can also be downloaded and installed on your local computer free of charge. You can use this tool to build your database easily and conveniently.
- 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 2005 project named SQLSelectRTOBJECT with five form windows: LogIn, Selection, Faculty, Course, and Student. Because there is a great similarity between this project and the project we developed in the last section, AccessSelectRTOBJECT, you do not need to redevelop all the coding for this one. What you need to do is to create a new project named SQLSelectRTOBJECT and add all five forms from the last project to this one by using the Project|Add Existing Item menu item. The only difference between these two projects that are connected to different databases is in the data provider-dependent objects, and the most important part is the connection string. To save time, in this section we only emphasize the differences in coding between that used in the previous section and that used in this one.

Now create a new Windows-based project and name it SQLSelectRTOBJECT.

#### **4.18.2 Query Data Using Runtime Objects for the LogIn Form**

Open the default Form1 window by double-clicking on Form1.vb in the Solution Explorer window, and then right-click this form and click the Delete item and then Yes in the message box to remove this form from our project. This deletion will

cause a compiling error, which we will fix later. Now we need to add all five form windows from the AccessSelectRTOBJECT project to this project. Go to the Project menu and choose Add Existing Item, and then browse to the AccessSelectRTOBJECT project folder. Press and hold the Ctrl key on your keyboard and click five forms one by one: LogIn Form.vb, Selection Form.vb, Faculty Form.vb, Course Form.vb, and Student Form.vb. Click the Add button to add these forms into our current project.

Now let's fix the error caused by deleting the default form window. Click the Show All Files button at the top of the Solution Explorer window to display all files in the current project, then expand My Project to Application.myapp and finally to the file Application.Designer.vb. Click this file to open it. The code in this file is autogenerated by the system as you create a new project. Move to the bottom of the file and find this line of code:

---

**Me.MainForm = Global.SQLSelectRTOBJECT.Form1**

---

The default main form is Form1 when you create a new Windows-based project. A blue line appears under Form1 since we have deleted it. Replace the default form with our LogIn form by removing Form1 and typing LogInForm in its place. The problem is fixed. Then you need to confirm that your startup form is the LogIn form window. To do that, go to the Project menu and click SQLSelectRTOBJECT Properties to open the project property window. Make sure that LogInForm is located in the Startup form box. If not, select LogInForm as the Startup form.

Open the LogIn code window from the Solution Explorer window by clicking the View Code button to begin our coding.

An easy way to develop the coding for this section is to replace the prefix “acc” that precedes 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, most of the other coding is identical as long as the connection string is modified and matched to the selected database or the Data Provider.

#### 4.18.2.1 Declare the Runtime Objects

As 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 your code window to allow Visual Basic.NET 2005 to know that you want to use this specified Data Provider. Enter the code shown in Figure 4.99 at the top of this code window.

A new instance of the SqlConnection class is created with the accessing mode of Public, which means that we want to use this connection object for our whole project.

The first job you need to do after a new instance of the data connection object is declared is to connect your project with the database you selected.

```
(General) (Declarations)
Imports System.Data
Imports System.Data.SqlClient
Public Class LogInForm
    Public sqlConnection As SqlConnection
```

Figure 4.99. The declaration of the namespace for the SQL Server Data Provider.

#### 4.18.2.2 Connect to the Data Source with the Runtime Object

Since the connection job is the first thing you need to do before you can perform any data queries, you need to do the connection job in the first event procedure, Form\_Load(), to allow the connection to be made first as your project runs.

In the code window, click the drop-down arrow on the Class Name combo box and select 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, which is shown in Figure 4.100. Enter the code shown in Figure 4.100 into this event procedure.

Let's see how this piece of code works.

- A. A SqlConnection String is created first, and the connection string is used to connect your project with the SQL Server database selected. Note that this connection string is different from the one we created in the last section. The Server parameter is assigned a value of localhost, which means that the SQL Server is installed on our local computer. The Data Source parameter is used to indicate the server name. In this case, since we installed the server on our local computer, we need to use the local computer's name (Susan) followed by the server name (SQLEXPRESS). To identify your computer's name, right-click the My Computer icon on your desktop screen, select and open

```
(LogInForm Events) Load
Private Sub LogInForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    Dim sqlString As String = "Server=localhost;" +
        "Data Source=Susan\SQLEXPRESS;Database=CSE_DEPT;" +
        "Integrated Security=SSPI"
    sqlConnection = New SqlConnection(sqlString)
    Try
        sqlConnection.Open()
        Catch SqlExceptionErr As SqlException
            MessageBox.Show(SqlExceptionErr.Message, "SQL Error")
        Catch InvalidOperationExceptionErr As InvalidOperationException
            MessageBox.Show(InvalidOperationExceptionErr.Message, "SQL Error")
        End Try
    If sqlConnection.State <> ConnectionState.Open Then
        MessageBox.Show("Database connection is Failed!")
        Exit Sub
    End If
End Sub
```

Figure 4.100. The coding for the database connection.

the System Properties window, then click the Network Identification tab, and you can find your computer's full name. The database we used for this project is the 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 does utilize those two items.

- B. A new instance of SqlConnection class is created with the connection string as an argument.
- C. A Try...Catch block is utilized here to catch any mistakes caused by opening this connection. The advantage of using this kind of strategy is to avoid an unnecessary system debug process or simplify it.
- D. This step is used to confirm that our database connection is successful. If not, an error message is displayed and the project is exited.

After a database connection is successfully made, we need to use this connection to access the SQL Server database to perform our data query job.

#### **4.18.2.3 Coding for Method 1: Using the TableAdapter to Query Data**

In this section, we create and use runtime objects to query data from the SQL Server database by using the TableAdapter method.

Open the LogIn form window by clicking the View Designer button, and then double-click the TabLogIn button to open its event procedure. Enter the code shown in Figure 4.101 into this event procedure.

Let's see how this piece of code works.

- A. Since the query string applied in this application is relatively long, we break it into two substrings, cmdString1 and cmdString2. Then we combine these two substrings to form a complete query string. One point you need to pay attention to is the relational operator applied in the SQL Server database, which is different from 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 we used another method 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, paramUserName and paramPassWord, separately. The parameter's name must be identical to the name of dynamic parameter in the SQL statement string. The values of the two parameters should be equal to the contents of the two associated text box controls, which will be entered by the user as the project runs.

```

Private Sub TabLogIn_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TabLogIn.Click
    Dim cmdString1 As String = "SELECT user_name, pass_word, faculty_id, student_id 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 LogInTableAdapter As New SqlDataAdapter
    Dim sqlDataTable As New DataTable
    Dim sqlCommand As New SqlCommand
    Dim selForm As New SelectionForm

    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)
    LogInTableAdapter.SelectCommand = sqlCommand
    LogInTableAdapter.Fill(sqlDataTable)

    If sqlDataTable.Rows.Count > 0 Then
        selForm.Show()
        Me.Hide()
    Else
        MessageBox.Show("No matched username/password found!")
    End If

    sqlDataTable.Dispose()
    sqlDataTable = Nothing
    sqlCommand.Dispose()
    sqlCommand = Nothing
    LogInTableAdapter.Dispose()
    LogInTableAdapter = Nothing
End Sub

```

Figure 4.101. The coding for the TabLogIn button event procedure.

- D. Now 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.

The rest of the coding is identical to that developed in the last section, and a detailed explanation was given in the last section, too.

Now let's take a look at the coding for the second method.

#### 4.18.2.4 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 the ReadLogIn button to open its event procedure. Enter the code in Figure 4.102 into this event procedure.

Most of the code in the top section is identical to that of the TabLogIn button 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

	ReadLogIn	▼	Click	▼
--	-----------	---	-------	---

```

Private Sub ReadLogIn_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ReadLogIn.Click
    A Dim cmdString1 As String = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "
    B Dim cmdString2 As String = "WHERE (user_name LIKE @name) AND (pass_word LIKE @word)"
    C Dim cmdString As String = cmdString1 & cmdString2
    D Dim paramUserName As New SqlParameter
    E Dim paramPassWord As New SqlParameter
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader
    Dim selForm As New SelectionForm
    paramUserName.ParameterName = "@name"
    paramUserName.Value = txtUserName.Text
    paramPassWord.ParameterName = "@word"
    paramPassWord.Value = txtPassWord.Text
    sqlCommand.Connection = sqConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add(paramUserName)
    sqlCommand.Parameters.Add(paramPassWord)
    sqlDataReader = sqlCommand.ExecuteReader
    If sqlDataReader.HasRows = True Then
        selForm.Show()
        Me.Hide()
    Else
        MessageBox.Show("No matched username/password found!")
    End If
    sqlCommand.Dispose()
    sqlCommand = Nothing
    sqlDataReader.Close()
    sqlDataReader = Nothing
End Sub

```

**Figure 4.102.** The coding for the ReadLogIn button event procedure.

removed from this event procedure since we do not need it for our data query in this method.

Let's see how this piece of code works.



When using the Parameters collection to add dynamic parameters, the order of the dynamic parameters is very important. The order in which you add the dynamic parameters into the Parameters collection must be identical to the order in which the nominal parameters are placed in the SQL statement string.

- As we did in the coding for the method 1, the comparison operator LIKE is used to replace the equal operator in the second query string. This is a requirement of the data query format for the SQL Server database.
- Two SqlParameter objects are created and 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.
- 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 to 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 coding is identical to the coding we did in the last section.

The coding for the Cancel command button event procedure is also identical to the coding we did in the last section with no modification.

#### 4.18.3 Coding for the Selection Form

Most coding in this form is identical to that 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 was used and all data provider-dependent objects were preceded by the prefix “acc”, such as in accConnection. In this project we use a SQL Server database, so the connection object should be preceded by the prefix “sql”. When the Exit button is clicked, we need to check whether the connection object has been closed and released. Since the connection object is created in the LogIn class, the connection object should be preceded by the class name LogIn. The only modification you need to do is to change the prefix “acc” to “sql” for the connection instance, shown in Figure 4.103.

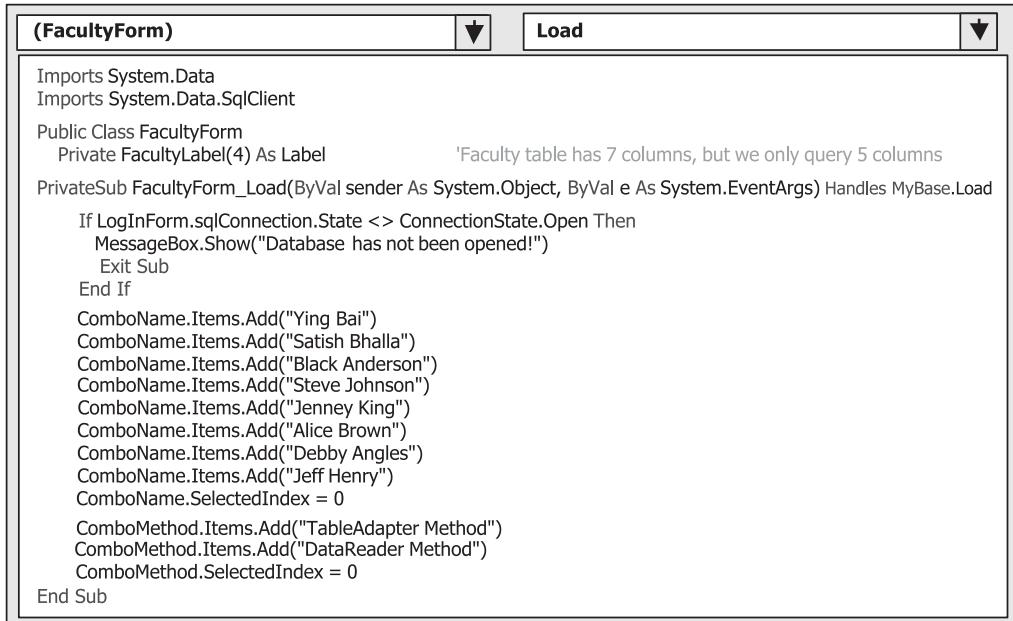
#### 4.18.4 Query Data Using Runtime Objects for the Faculty Form

First, let's take a look at the coding for the Form\_Load() event procedure. The differences between this coding and that in the last project are as follows:

- A. The namespace of the Data Provider is different. System.Data.OleDb is utilized for the last project since an Access database was used. Since we use the SQL Server database in this section, change the namespace to System.Data.SqlClient, which is shown in Figure 4.104.

cmdExit	▼	Click	▼
<pre>Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdExit.Click     If LogInForm.sqlConnection.State = ConnectionState.Open Then         LogInForm.sqlConnection.Close()         LogInForm.sqlConnection.Dispose()         LogInForm.sqlConnection = Nothing     End If     Application.Exit() End Sub</pre>			

Figure 4.103. The modified coding for the Exit button event procedure.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says '(FacultyForm)'. There are two dropdown arrows on the right side of the title bar. The code itself is as follows:

```

Imports System.Data
Imports System.Data.SqlClient

Public Class FacultyForm
    Private FacultyLabel(4) As Label
    'Faculty table has 7 columns, but we only query 5 columns

    Private Sub FacultyForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        If LogInForm.sqlConnection.State <> ConnectionState.Open Then
            MessageBox.Show("Database has not been opened!")
            Exit Sub
        End If
        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("DataReader Method")
        ComboMethod.SelectedIndex = 0
    End Sub

```

Figure 4.104. The modified coding for the FacultyForm\_Load() event procedure.

- B. The FacultyLabel object array used in the last project has seven elements, which are mapped to seven columns in the Faculty table. In this project, a modification is made when we perform the data query. Instead of querying all seven columns, we query only five columns of data, from the third to the last column. Therefore the size of the object array FacultyLabel is reduced to 5. Since the first element's index in the object array is 0, the upper bound index should be 4 for an object array containing five elements.
- C. The prefix of the connection object is changed to "sql" since an SQL Server Data Provider is utilized in the project.

Next is the coding for the Select button event procedure. Open the Faculty form window by clicking the View Designer button from the Solution Explorer window, then double-click the Select button to open its event procedure. Make the modifications shown in Figure 4.105 to the original code. The differences are as follows:

- A. The query string is modified from querying seven to five columns since the first two columns are id and name, and we do not need these. Five labels on the Faculty form window are TitleLabel, OfficeLabel, PhoneLabel, CollegeLabel, and EmailLabel, which are mapped to the five columns in the Faculty table.
- B. A SqlParameter object is created and is used later to hold the dynamic parameter's name and value.
- C. Two Data Provider-dependent objects are created, and they are sqlCommand and sqlDataReader. The sqlDataTable is a Data Provider-independent object.

	cmdSelect	▼	Click	▼
--	-----------	---	-------	---

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString1 As String = "SELECT office, phone, college, title, email FROM Faculty"
    Dim cmdString2 As String = "WHERE name LIKE @facultyName"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim paramFacultyName As New SqlParameter
    Dim FacultyTableAdapter As New SqlDataAdapter
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader
    Dim sqlDataTable As New DataTable

    paramFacultyName.ParameterName = "@facultyName"
    paramFacultyName.Value = ComboName.Text
    sqlCommand.Connection = LogInForm.sqlConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add(paramFacultyName)
    Call ShowFaculty(ComboName.Text)

    If ComboMethod.Text = "TableAdapter Method" Then
        FacultyTableAdapter.SelectCommand = sqlCommand
        FacultyTableAdapter.Fill(sqlDataTable)
        If sqlDataTable.Rows.Count > 0 Then
            Call FillFacultyTable(sqlDataTable)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        sqlDataTable.Dispose()
        sqlDataTable = Nothing
        FacultyTableAdapter.Dispose()
        FacultyTableAdapter = Nothing
    Else
        sqlDataReader = sqlCommand.ExecuteReader
        If sqlDataReader.HasRows = True Then
            Call FillFacultyReader(sqlDataReader)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        sqlDataReader.Close()
        sqlDataReader = Nothing
    End If
    sqlCommand.Dispose()
    sqlCommand = Nothing
End Sub

```

**Figure 4.105.** The modified coding for the Select button event procedure.

- D. The SqlParameter object is initialized by assigning it with the parameter's name and value.
- E. The SqlCommand object is initialized by assigning it with three values.

The following code is similar to what we developed in the same event procedure for the last project. The only modification you need to perform is to change the prefix “acc” that precedes each object to “sql”, namely, accCommand to sqlCommand, accDataTable to sqlDataTable, and accDataReader to sqlDataReader.

Yes, it's that easy!

For three subroutines, FillFacultyTable(), FillFacultyReader(), and MapFacultyTable(), only small modifications are made. For the first two subroutines, the modification is to change the upper bound index from 6 to 4 since we reduced the number of queried columns from seven to five. Also change the nominal argument's type

```

FacultyForm FillFacultyTable

Private Sub FillFacultyTable(ByVal FacultyTable As DataTable)
    Dim pos1 As Integer
    Dim column As New DataColumn
    Dim row As DataRow
    For pos2 As Integer = 0 To 4                                'Initialize the object array
        FacultyLabel(pos2) = New Label()
    Next pos2
    Call MapFacultyTable(FacultyLabel)
    For Each row In FacultyTable.Rows
        For Each column In FacultyTable.Columns
            FacultyLabel(pos1).Text = row(column)
            pos1 = pos1 + 1
        Next
    Next
End Sub

```

**Figure 4.106.** The modification coding for the subroutine FillFacultyTable().

from OleDbDataReader to SqlDataReader for the FillFacultyReader() subroutine. Steps A in Figures 4.106 and 4.107 and B in Figure 4.107 show these modifications.

The modification made for the subroutine MapFacultyTable() is to shift the order index of the Label object array. In the last project, we queried seven columns from the Faculty table starting from column 0, which is the faculty\_id, and column 1, which is the name in the Faculty table. But we do not need these two columns in our project and we want to display the faculty information starting from column 2, which is the office in the Faculty table. So we assign the OfficeLabel to the second element in the Label object array. In this project, we query only five columns starting from column 2 in the Faculty table, which is the office column, so we assign the OfficeLabel to the 0th element in the Label object array. Refer to Figure 4.108 for this modification.

The coding for the subroutine ShowFaculty() and the Back button event procedures is identical to that of the last project, with no modification. Refer to Figures 4.79 and 4.80 in Section 4.17.3 for the detailed coding and explanations.

```

FacultyForm FillFacultyReader

Private Sub FillFacultyReader(ByVal FacultyReader As SqlDataReader)
    Dim intIndex As Integer
    For intIndex = 0 To 4                                'Initialize the object array
        FacultyLabel(intIndex) = New Label()
    Next intIndex
    Call MapFacultyTable(FacultyLabel)
    While FacultyReader.Read()
        For intIndex = 0 To FacultyReader.FieldCount - 1
            FacultyLabel(intIndex).Text = FacultyReader.Item(intIndex).ToString
        Next intIndex
    End While
End Sub

```

**Figure 4.107.** The modification coding for the subroutine FillFacultyReader().

<b>FacultyForm</b>	▼
<pre>Private Sub MapFacultyTable(ByRef fLabel As Object)     fLabel(0) = OfficeLabel     fLabel(1) = PhoneLabel     fLabel(2) = CollegeLabel     fLabel(3) = TitleLabel     fLabel(4) = EmailLabel End Sub</pre>	
▼	▼

Figure 4.108. The modification coding for the subroutine MapFacultyTable().

#### 4.18.5 Query Data Using Runtime Objects for the Course Form

Now, let's do the coding for the CourseForm\_Load() event procedure.

Basically, the coding of this event procedure is similar to what we did for the same event procedure in the last project. The only modifications are as follows (Figure 4.109):

- The Data Provider namespace is modified. The System.Data.SqlClient namespace is used to replace System.Data.OleDb since we used an SQL Server Data Provider in this section.
- The size of the text box object array is reduced to 5 since we only need to query five columns from the Course table and display them on the five associated text box controls in the Course form window. Also, a dynamic string array, CourseID, is declared here to hold the course\_id for each course

<b>(CourseForm)</b>	▼
<pre>A Imports System.Data Imports System.Data.SqlClient  B Public Class CourseForm     Private CourseTextBox(4) As TextBox      'we only query 5 columns from the Course table     Private CourseID() As String            'create a dynamic CourseID string array      C Private Sub CourseForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load         If LogInForm.sqlConnection.State &lt;&gt; ConnectionState.Open Then             MessageBox.Show("Database has not been opened!")             Exit Sub         End If         ComboBoxName.Items.Add("Ying Bai")         ComboBoxName.Items.Add("Satish Bhalla")         ComboBoxName.Items.Add("Black Anderson")         ComboBoxName.Items.Add("Steve Johnson")         ComboBoxName.Items.Add("Jenney King")         ComboBoxName.Items.Add("Alice Brown")         ComboBoxName.Items.Add("Debby Angles")         ComboBoxName.Items.Add("Jeff Henry")         ComboBoxName.SelectedIndex = 0         ComboBoxMethod.Items.Add("TableAdapter Method")         ComboBoxMethod.Items.Add("DataReader Method")         ComboBoxMethod.SelectedIndex = 0     End Sub</pre>	
▼	▼

Figure 4.109. The modified coding for the CourseForm\_Load() event procedure.

selected from the Course list box later. We need this course\_id to perform the query to get the detailed information for each course.

C. The prefix “acc” is replaced by “sql” for the connection object.

The next coding is for the Select button event procedure. This coding is similar to what we did in the same event procedure in the last project. But one important improvement made in this coding 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 faculty member selected in the Course form. The reason is that 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 joint query is a desired method to complete this functionality.

#### 4.18.6 Retrieve Data from Multiple Tables Using Joined Tables

Parts of the Faculty and Course data tables in the CSE\_DEPT database are shown in Table 4.8.

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 Course tables is one-to-many. What we want to do is to pick up all courses from the Course table based on the selected faculty name that is located in the Faculty table, but no faculty name is available in the Course table. An efficient way to do this is to use a query with

**Table 4.8. Parts of Faculty and Course Data Tables**

<b>Faculty Table</b>		
<b>faculty_id</b>	<b>Name</b>	<b>Office</b>
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
<b>Course Table</b>		
<b>course</b>	<b>faculty_id</b>	<b>classroom</b>
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

two joined tables, which means that we need to perform a query by joining two different tables, Faculty and Course, to pick up the course information. To join these two tables, we need to use the primary key and the foreign key, faculty\_id, to set up this relationship. In other words, we want to obtain all courses, that is, all course names, from the Course table based on the faculty name in the Faculty table. But in the Course table, we have only course names and the associated faculty\_id information available. Similarly, in the Faculty table, we have only faculty names and the associated faculty\_id information available. As a result, we cannot set up a direct relationship between the faculty name in the Faculty table and the course name 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 using the primary and foreign keys.

A SQL statement with two joined tables, Faculty and Course, can be represented as follows:

---

```
SELECT Course.course FROM Course, Faculty  
WHERE (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.name  
LIKE @name)
```

---

The @name is a dynamic parameter and will be replaced by the real faculty name as the project runs.

One point to note is that the syntax of this SQL statement is defined in the ANSI 89 standard and is relatively out of date. Microsoft will not support this out-of-date syntax in the future, so it is highly recommended to use a new syntax for this SQL statement, which is defined in the ANSI 92 standard and looks like this:

---

```
SELECT Course.course FROM Course JOIN Faculty  
ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.name  
LIKE @name)
```

---

Now let's use this inner join method to develop our query for this Course form. The modified code is shown in Figure 4.110.

Let's take a look at the modifications in detail:

- A. The joined table query string is declared at the beginning of this event procedure. A concatenate operator is used to join two subquery strings together to form a complete query statement. 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 the course name, as the identifier to pick up each course's detailed information from the Course table when the user clicks and selects the course names from the Course list box. It is not convenient for the user to use the course\_id as the identifier to get the

cmdSelect	▼	Click	▼
<pre> Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click     Dim cmdString1 As String = "SELECT Course.course_id, Course.course FROM Course JOIN Faculty "     Dim cmdString2 As String = "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.name LIKE @name)"     Dim cmdString As String = cmdString1 &amp; cmdString2     Dim CourseTableAdapter As New SqlDataAdapter     Dim sqlCommand As New SqlCommand     Dim sqlDataReader As SqlDataReader     Dim sqlDataTable As New DataTable     sqlCommand.Connection = LogInForm.sqlConnection     sqlCommand.CommandType = CommandType.Text     sqlCommand.CommandText = cmdString     sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = ComboName.Text     If ComboMethod.Text = "TableAdapter Method" Then         CourseTableAdapter.SelectCommand = sqlCommand         CourseTableAdapter.Fill(sqlDataTable)         If sqlDataTable.Rows.Count &gt; 0 Then             Call FillCourseTable(sqlDataTable)         Else             MessageBox.Show("No matched course found!")         End If         sqlDataTable.Dispose()         sqlDataTable = Nothing         CourseTableAdapter.Dispose()         CourseTableAdapter = Nothing     Else         sqlDataReader = sqlCommand.ExecuteReader         If sqlDataReader.HasRows = True Then             Call FillCourseReader(sqlDataReader)         Else             MessageBox.Show("No matched course found!")         End If         sqlDataReader.Close()         sqlDataReader = Nothing     End If     sqlCommand.Dispose()     sqlCommand = Nothing     CourseList.SelectedIndex = 0 End Sub </pre>			

Figure 4.110. The modified coding for the Select button event procedure.

detailed information for that course. So we pick the course\_id with the course name together in this joined table query, and we will use that course\_id later. The comparison operator LIKE is used to replace the equal symbol for the criteria in the ON clause in the definition of the query string; this is required for SQL Server database operation.

- B. Some new SQL objects are created such as the CourseTableAdapter, Command, DataReader, and DataTable. All these objects should be prefixed by the keyword “sql” to indicate that those components are related to the SQL Server database.
- C. The sqlCommand object is initialized with the connection string, command type, command text, and command parameter. The parameter’s name must be identical to the dynamic nominal name @name, which is defined in the query string and is located right 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 code is similar to what 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 courses into the Course list box.
- E. Otherwise, the DataReader method is selected by the user, the ExecuteReader() method is executed to read back all courses, and the FillCourseReader() subroutine is called to fill the courses into the CourseList box.
- F. Finally, some cleaning jobs are performed to release objects used for this query.

The subroutine FillCourseTable() and the Back button event procedure have nothing to do with any object used in this project, so no coding modification is needed. The subroutine FillCourseReader() needs only one small modification, which is to change the nominal argument's type from the OleDbDataReader to SqlDataReader since now we are using an SQL Server data provider. The detailed explanations for the subroutines FillCourseTable() and FillCourseReader() (Figure 4.85) can be found in Section 4.17.4.

Next, we need to take care of the coding for the CourseList\_SelectedIndexChanged() event procedure.

All detailed information related to the selected course from the Course list box should be displayed in five text box controls when the user clicks and selects a course from the Course list box control. The coding for this event procedure is similar to what we did in the same event procedure in the last project, with the following modifications (Figure 4.111):

- A. The query string is created with five queried columns: course\_id, credit, classroom, schedule, and enrollment. The query criterion is course\_id, which was obtained from the FillCourseTable() and FillCourseReader() subroutines. 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 equal symbol for the criteria in the WHERE clause in the definition of the query string; this is required for SQL Server database operation.
- B. Some new SQL data objects are created. These objects are used to perform the data operations between the database and our project. All of these objects should be prefixed by the keyword “sql” since in this project we used an SQL Server data provider.
- C. The sqlCommand object is initialized with the connection string, command type, command text, and command parameter. The parameter's name must be identical to the dynamic nominal name @courseid, which is defined in the query string, right after the LIKE comparator in the WHERE clause. The parameter's value is the element's value in the CourseID array we obtained from the FillCourseTable() and FillCourseReader() subroutines. The index of the CourseID array is the SelectedIndex of the CourseList box,

CourseList	▼	SelectedIndexChanged	▼
------------	---	----------------------	---

```

Private Sub CourseList_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles CourseList.SelectedIndexChanged
    Dim cmdString1 As String = "SELECT course_id, credit, classroom, schedule, enrollment FROM Course "
    Dim cmdString2 As String = "WHERE course_id LIKE @courseid"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim CourseTableAdapter As New SqlDataAdapter
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader
    Dim sqlDataTable As New DataTable
    sqlCommand.Connection = LogInForm.sqlConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add("@courseid", SqlDbType.Char).Value = CourseID(CourseList.SelectedIndex)
    If ComboMethod.Text = "TableAdapter Method" Then
        CourseTableAdapter.SelectCommand = sqlCommand
        CourseTableAdapter.Fill(sqlDataTable)
        If sqlDataTable.Rows.Count > 0 Then
            Call FillCourseTextBox(sqlDataTable)
        Else
            MessageBox.Show("No matched course information found!")
        End If
        sqlDataTable.Dispose()
        sqlDataTable = Nothing
        CourseTableAdapter.Dispose()
        CourseTableAdapter = Nothing
    Else
        sqlDataReader = sqlCommand.ExecuteReader
        If sqlDataReader.HasRows = True Then
            Call FillCourseReaderTextBox(sqlDataReader)
        Else
            MessageBox.Show("No matched course information found!")
        End If
        sqlDataReader.Close()
        sqlDataReader = Nothing
    End If
    sqlCommand.Dispose()
    sqlCommand = Nothing
End Sub

```

**Figure 4.111.** The modified coding for the CourseList\_SelectedIndexChanged procedure.

which is equivalent to the course name selected by the user as the project runs.

- D. If the TableAdapter method is selected by the user, the Fill() method is called to fill the Course table, and the FillCourseTextBox() subroutine is executed to fill five text boxes to display the detailed course information for the course selected in the Course list box.
- E. Otherwise, the DataReader method is selected. The ExecuteReader() method is executed to read back the detailed information for the selected course, and the FillCourseReaderTextBox() subroutine is called to fill that information into five text boxes.
- F. Finally, a cleaning job is performed to release 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. The subroutine FillCourseReaderTextBox() needs a small modification,

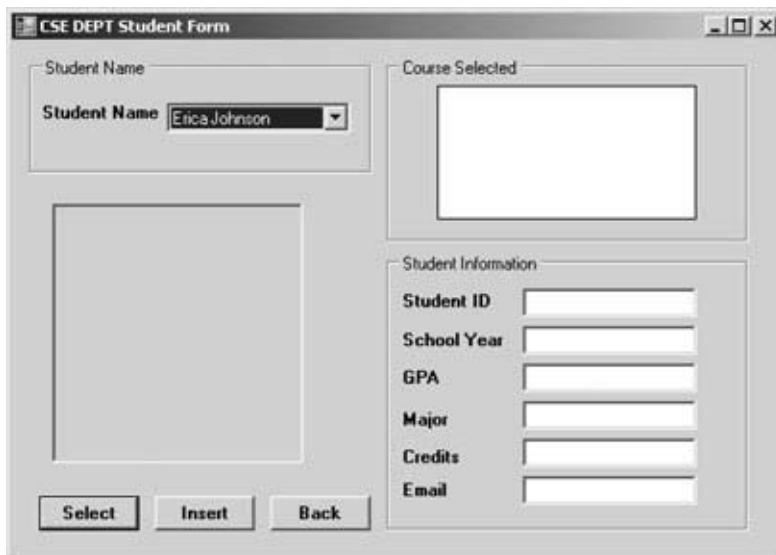


Figure 4.112. The Student form window.

which is to change the nominal argument's type from OleDbDataReader to SqlDataReader since an SQL Server data provider is utilized in this project. For the detailed line-by-line explanations of the subroutines FillCourseTextBox(), FillCourseReaderTextBox(), and MapCourseTable() (Figures 4.87 and 4.88), refer to Section 4.17.4.

Do not forget to copy all faculty image files to the folder in which your Visual Basic executable file is located before you run this project. In this application, it is the **Debug** folder of the project.

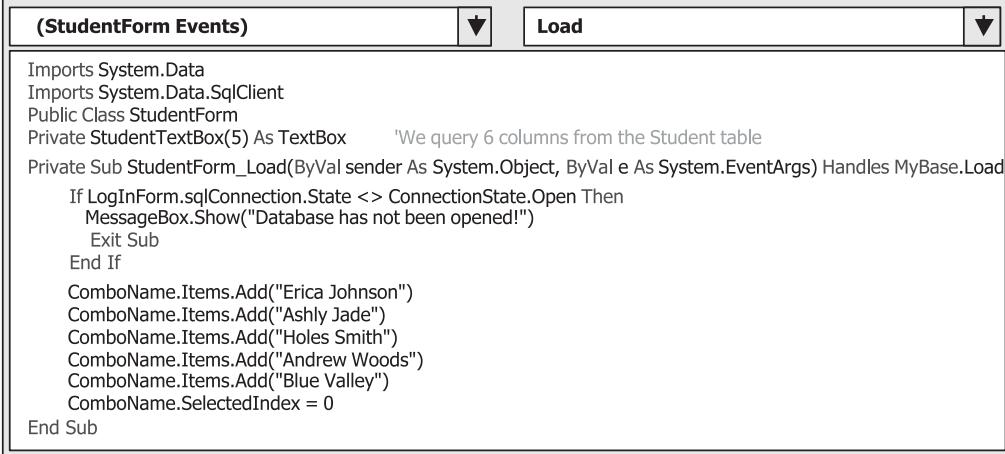
#### 4.18.7 Query Data Using Runtime Objects for the Student Form

Now we finally come to the coding for the Student form window. The Student form window is shown again in Figure 4.112 for your convenience. The function of this form is to pick up all information related to the selected student, such as the student ID, GPA, credits, major, school year, and email, and display it in six text boxes when the Select button is clicked by the user. Also, the courses taken by that student are displayed in the Course list box. Apparently this function needs to make two queries to two different tables, the Student and the StudentCourse tables, respectively.

The coding for this form is similar to what we did for the Faculty form, with one important difference, which is the query type. In order to improve the querying efficiency and simplify the coding, two stored procedures are developed and implemented in this section. By using stored procedures, query coding can be significantly simplified and integrated, and the efficiency of the data query can also be improved.

Let's start with the Form\_Load event procedure.

The coding for this event procedure is shown in Figure 4.113.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says '(StudentForm Events)'. Below it, there are two dropdown menus and a 'Load' button. The code itself is as follows:

```

A Imports System.Data
Imports System.Data.SqlClient
Public Class StudentForm
B Private StudentTextBox(5) As TextBox      'We query 6 columns from the Student table
Private Sub StudentForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
C     If LogInForm.sqlConnection.State <> ConnectionState.Open Then
        MessageBox.Show("Database has not been opened!")
        Exit Sub
    End If
D     ComboName.Items.Add("Erica Johnson")
     ComboName.Items.Add("Ashly Jade")
     ComboName.Items.Add("Holes Smith")
     ComboName.Items.Add("Andrew Woods")
     ComboName.Items.Add("Blue Valley")
     ComboName.SelectedIndex = 0
End Sub

```

**Figure 4.113.** The coding for the Form\_Load event procedure.

- 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 text box array, StudentTextBox(), is declared and is used to hold the detailed student information as the project runs, and this information will be displayed in six text boxes in the Student form later.
- C. The database connection is checked before we perform any data operations between the project and the related database.
- D. All sampled students' names are added into the Student Name combo box, and the default student's name is the first one in this combo box.

Next, let's take a look at the coding for the Select button event procedure. As mentioned at the beginning of this section, when this Select button is clicked by the user, six pieces of student information are displayed in six related text boxes, and the courses taken by that student are displayed in the Course list box. Two queries are needed for this operation. In order to save time and space, two stored procedures are developed for these two queries. Let's go a little deeper into the stored procedure.

#### 4.18.8 Query Data Using Stored Procedures

Stored procedures are nothing more than functions or procedures applied in any project developed in any programming language. This means that stored procedures can be considered functions or subroutines, and they can be called easily with any arguments and can also return any data with a 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 the coding 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 increased and the data query's speed can be significantly improved. 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.

In general, 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. These 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.

#### **4.18.8.1 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 as follows:

```
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
```

For SQL Server database, the name of the stored procedure is always prefixed by the keyword “dbo.” A sample stored procedure, StudentInfo, is shown below:

```
CREATE PROCEDURE dbo.StudentInfo
{
    @StudentName  VARCHAR(50)
}
AS
SELECT student_id FROM Student
WHERE name LIKE @StudentName
RETURN
```

The parameters declared inside the braces are either input or output parameters used for this stored procedure, and an @ symbol must be prefixed to the parameter in the SQL Server database. Any argument sent from the calling procedure to this stored procedure should be declared in here. The other variables, which are created by using the keyword DECLARE after the keyword AS, are local variables and can only be used in this stored procedure. The keyword RETURN is used to return the queried data columns.

#### 4.18.8.2 Call the Stored Procedure

When the stored procedure is created, it is ready to be called by your project developed in Visual Basic.NET. You can use any possible way 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 reading result to a DataReader object. The above methods are good for the single-table query, which means that a group of SQL statements defined in that stored procedure are executed for only one data table. If you want to develop a stored procedure that makes multiple queries with multiple data tables, you need to use the ExecuteNonQuery() method.

To call a developed stored procedure from a Visual Basic.NET project, you need to follow the syntax described below, which uses the Fill() method in TableAdapter:

1. Create a Connection object and open it.
2. Create a Command object and initialize it.
3. Create Parameter objects and add them into the Command object, if you have any.
4. Execute the stored procedure by using the Fill() method in the TableAdapter class.

Here is an example of coding that shows how to call a stored procedure named dbo.StudentInfo (assuming a Connection object has been created):

```

A  Dim StudentTableAdapter As New SqlDataAdapter
B  Dim sqlCmdStudent As New SqlCommand
C  Dim sqlStudentTable As New DataTable
D
E  sqlCmdStudent.Connection = LogInForm.sqlConnection
F  sqlCmdStudent.CommandType = CommandType.StoredProcedure
G  sqlCmdStudent.CommandText = "dbo.StudentInfo"
H  sqlCmdStudent.Parameters.Add("@StudentName", SqlDbType.Char).Value = ComboName.Text
I  StudentTableAdapter.SelectCommand = sqlCmdStudent
J
K  StudentTableAdapter.Fill(sqlStudentTable)
L  If sqlStudentTable.Rows.Count > 0 Then
M      Collect the retrieved data columns.....
N  Else
O      MessageBox.Show("No matched student found!")
P  End If

```

- A. Some useful data components are declared here, such as the TableAdapter, Command, and DataTable.
- B. The Command object is initialized by assigning the associated components to it. The first component is the Connection object.

- C. In order to execute a stored procedure, the keyword `StoredProcedure` must be used here and assigned to the  `CommandType` property of the `Command` object to indicate that a stored procedure will be called when this `Command` object is executed.
- D. The name of the stored procedure must be assigned to the  `CommandText` property of the `Command` object. This name must be identical to the name you used when you created 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 `SelectCommand` property of the `TableAdapter` to 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 what follows, we use our `Student` form to illustrate how to create two stored procedures and how to call them from a Visual Basic.NET project.

#### **4.18.8.3 Query Data Using Stored Procedures for the Student Form**

First, let's create two stored procedures for the `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 that we want to query all courses taken by the selected student based on the student's name, not the `student_id`, from the `StudentCourse` table. But there is only a `student_id` column available in the `StudentCourse` table and no student name in that table. The student name can only be obtained from the `Student` table. So we first need 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 (to be exact, 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 2005, and open the Server Explorer window by clicking the `View|Server Explorer` menu item. To open our database `CSE_DEPT`, right-click on `Data Connections` in the Server Explorer window and select the `Add Connection` item from the popup menu. In the `Add Connection` dialog box, perform the following actions to connect to our database:

1. Click the `Change` button that is next to the `Data source` box.
2. Browse to and select the Microsoft SQL Server Database File item and click `OK`.
3. Click the `Browse` button to go to our database file folder, `C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data`, select the database file `CSE_DEPT.mdf` by clicking it, and then click the `Open` button.
4. Clicking the `Test Connection` button to confirm this connection.

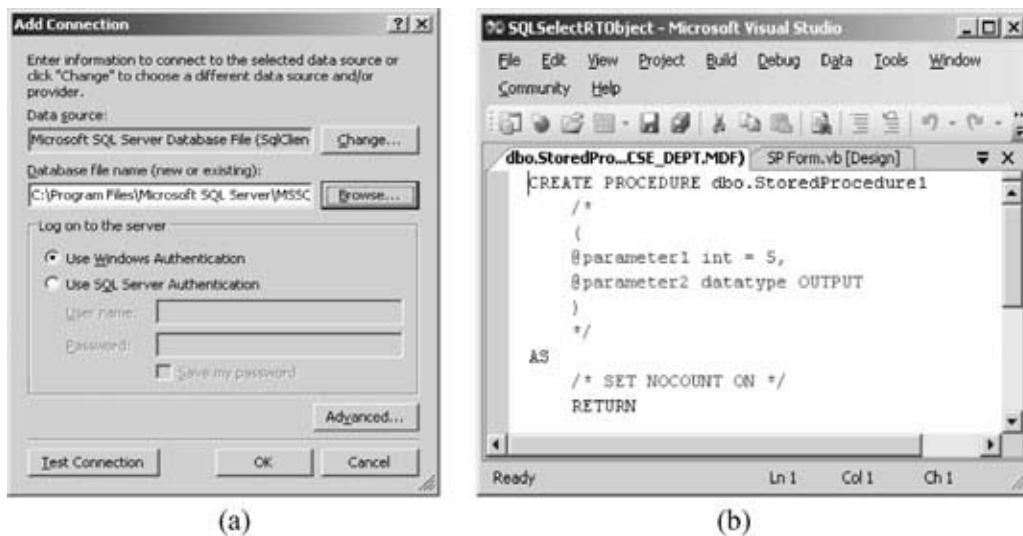


Figure 4.114. The Add Connection dialog box and a new stored procedure window.

Your finished Add Connection dialog box should match the one that is shown in Figure 4.114a.

The Use Windows Authentication radio button is selected since we want to use this security mode as our logon security checking method.

In the opened Server Explorer window, you can see that our database CSE\_DEPT has been connected to the server. Now let's create our first stored procedure.

Right-click on the Stored Procedures folder and select the Add New Stored Procedure item to open a new stored procedure window, which is shown in Figure 4.114b.

The default name for a new stored procedure is dbo.StoredProcedure1, which is located immediately after the keyword CREATE PROCEDURE. The top section of coding, which is commented out by the comment symbol (\* ... \*), is used to create the parameters or parameter list. The bottom section of coding is used to create the SQL statements. To create our first stored procedure, remove the comment symbols and enter the code shown in Figure 4.115 into this procedure.

```

CREATE PROCEDURE dbo.StudentInfo
(
    @StudentName VARCHAR(50)
)
AS
    SELECT student_id, gpa, credits, major, schoolYear, email FROM Student
    WHERE name LIKE @StudentName
    RETURN

```

Figure 4.115. The first stored procedure – dbo.StudentInfo.

```

% SQLSelectRTOObject - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
dbo.StudentC...CSE_DEPT.MDF) dbo.StudentIn...SF_DEPT.MDF*) dbo.StoredPro...CSE_DEPT.MDF)
ALTER PROCEDURE dbo.StudentCourseInfo
(
    @StudentID VARCHAR(50)
)
AS
    SELECT course_id FROM StudentCourse WHERE student_id LIKE @StudentID
    RETURN

```

Ready      ln 1      Col 1      Ch 1      INS

Figure 4.116. The second stored procedure – dbo.StudentCourseInfo.

The @StudentName is our only input parameter to this stored procedure, and this stored procedure will return six pieces of information related to the selected student based on the input student name parameter. Don't forget to modify the stored procedure's name to dbo.StudentInfo. Now click the File|Save StoredProcedure1 menu item to save this stored procedure.

Similarly, we can create our second stored procedure, dbo.Student-CourseInfo, which is shown in Figure 4.116.

The only input parameter to this stored procedure is @StudentID, and this stored procedure will return all courses (to be exact, all course.id) taken by the selected student based on the input parameter student\_id. Click the File|Save StoredProcedure1 menu item to save this stored procedure.

Now we have finished creating our two stored procedures. Next, we need to develop the coding for our Select command button located in the Student form window in the Visual Basic.NET environment to call these two stored procedures.

But wait a moment. Before we continue to develop our Visual Basic.NET coding to call these two stored procedures, is there any way for us to check whether these two stored procedures work fine or not? The answer is yes! The Server Explorer in Visual Studio.NET allows us to debug and test a custom stored procedure by using some popup menu items, as shown in Figure 4.117a.

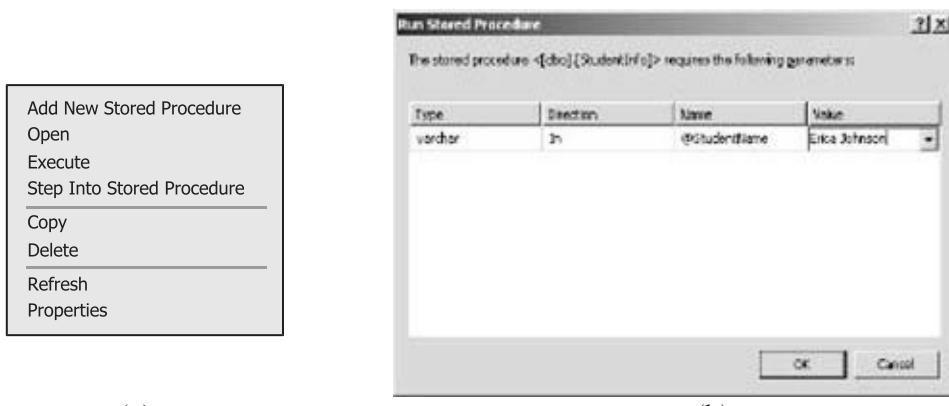


Figure 4.117. The popup menu and the Run dialog box.

The screenshot shows the Microsoft Visual Studio interface with the title bar "00 SQLSelectRTOObject - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has icons for Save, Undo, Redo, Cut, Copy, Paste, Find, Replace, and others. The Server Explorer window is open, showing "dbo.StoredPro...CSE\_DEPT.MDF". The Output window displays the results of running a stored procedure named [dbo].[StudentInfo]. The output shows one row with student\_id 377896 and gpa 3.95. It also shows the message "No rows affected." and "1 row(s) returned". The status bar at the bottom indicates "Ready", "Ln 10", "Col 1", and "Ch 1".

Figure 4.118. The testing result of our first stored procedure.

Open Visual Studio.NET 2005 and open and connect our database CSE\_DEPT to the server from the Server Explorer window. Expand the Stored Procedures folder and right-click that folder. A popup menu will be displayed, which is shown in Figure 4.117a. The functionality for each item is explained below:

1. **Add New Stored Procedure:** Create a new stored procedure. We have already used this item to create our two stored procedures.
2. **Open:** Open an existing stored procedure to allow it to be edited or modified. The name of the modified stored procedure will be prefixed by ALTER.
3. **Execute:** Execute a stored procedure. One can debug and test a developed stored procedure using this item in the Server Explorer environment to make sure that the developed stored procedure works fine.
4. **Step Into Stored Procedure:** Allows 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 the first stored procedure, StudentInfo, and select the EXECUTE item from the popup menu. A Run dialog box is displayed to allow you to enter input parameters if you have any, as shown in Figure 4.117b. Enter one of the sample students' names, Erica Johnson, into the Value box, and then click the OK button to run our stored procedure. The testing result is displayed in the Output dialog box, which is shown in Figure 4.118.

In all, there is one row with six columns returned: student\_id, gpa, credits, major, schoolYear, and email. Click the right arrow on the horizontal scroll bar to view all columns.

Similarly, you can try to run our second stored procedure. You need to enter the student\_id as the input parameter to run it. Of course, you can use the student\_id

```
Running [dbo].[StudentCourseInfo] (@StudentID = J77896)

course_id
-----
CSC-935
CSC-234A
CSC-439
CSC-432
CSE-439
CSC-333A

No rows affected.
(6 row(s) returned)
```

**Figure 4.119.** The testing result for our second stored procedure.

we obtained from our first stored procedure, which is J77896. The testing result is shown in Figure 4.119.

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 the server to open the database if you perform tasks with the Server Explorer such as creating stored procedures, because your server is connected and the database is opened when you create stored procedures. An error message will be displayed if you try to do that since this version only allows one instance of the server to be connected at a time. You have to disconnect that connection first by restarting your computer.

Open the graphical user interface of the Student form by first selecting and clicking on Student Form.vb and then clicking on the View Designer button from the Solution Explorer window. Double-click the Select button to open its event procedure, and enter the code shown in Figure 4.120 into this event procedure.

Let's take a close look at this coding.

- Two stored procedures' names are assigned to the two string variables strStudent and strStudentCourse, respectively. The names must be identical to those that we created in the stored procedures dbo.StudentInfo and dbo.StudentCourseInfo.
- All used data components are declared and created in this section, which include two TableAdapters, two DataTables, two Command objects, and a local string variable strName.

	<b>cmdSelect</b>	▼
	<b>Click</b>	▼

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim strStudent As String = "dbo.StudentInfo"
    Dim strStudentCourse As String = "dbo.StudentCourseInfo"
    Dim StudentTableAdapter As New SqlDataAdapter
    Dim StudentCourseTableAdapter As New SqlDataAdapter
    Dim sqlCommandStudent, sqlCommandStudentCourse As New SqlCommand
    Dim sqlStudentTable, sqlStudentCourseTable As New DataTable
    Dim strName As String

    C
    strName = FindName(ComboName.Text)
    If strName = "No Match" Then
        MessageBox.Show("No Matched Student Found!")
        Exit Sub
    End If

    D
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage
    PhotoBox.Image = System.Drawing.Image.FromFile(strName)

    E
    Call BuildCommand(sqlCmdStudent, strStudent)
    sqlCmdStudent.Parameters.Add("@StudentName", SqlDbType.Char).Value = ComboName.Text

    F
    StudentTableAdapter.SelectCommand = sqlCmdStudent
    StudentTableAdapter.Fill(sqlStudentTable)

    G
    If sqlStudentTable.Rows.Count > 0 Then
        Call FillStudentTextBox(sqlStudentTable)
    Else
        MessageBox.Show("No matched student_id found!")
    End If

    H
    Call BuildCommand(sqlCmdStudentCourse, strStudentCourse)
    sqlCmdStudentCourse.Parameters.Add("@StudentID", SqlDbType.Char).Value = txtID.Text
    StudentCourseTableAdapter.SelectCommand = sqlCmdStudentCourse
    StudentCourseTableAdapter.Fill(sqlStudentCourseTable)

    I
    If sqlStudentCourseTable.Rows.Count > 0 Then
        Call FillCourseList(sqlStudentCourseTable)
    Else
        MessageBox.Show("No matched course_id found!")
    End If

    K
    sqlStudentTable.Dispose()
    sqlStudentTable = Nothing
    sqlStudentCourseTable.Dispose()
    sqlStudentCourseTable = Nothing
    StudentTableAdapter.Dispose()
    StudentTableAdapter = Nothing
    StudentCourseTableAdapter.Dispose()
    StudentCourseTableAdapter = Nothing
    sqlCmdStudent.Dispose()
    sqlCmdStudent = Nothing
    sqlCmdStudentCourse.Dispose()
    sqlCmdStudentCourse = Nothing
End Sub

```

Figure 4.120. The coding for the Select button event procedure.

- C. The `FindName()` function is executed to get the student's photo file based on the student's name. The returned student 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 picture box.
- 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 identical to the name of the stored procedure we developed above.

- F. The unique input parameter to the stored procedure dbo.StudentInfo is 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 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 exactly 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 is filled into the Student table, and the subroutine FillStudentTextBox() is called to fill six text boxes in the Student form with six retrieved columns from the stored procedure. An error message is displayed if this fails.
- 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 as follows:

- 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 StudentID, which is obtained from the calling of the first stored procedure and stored in the student ID text box 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 exactly 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 is filled into the StudentCourse table, and the subroutine FillCourseList() is called to fill the Course list box in the Student form with all courses (course\_id) retrieved from the stored procedure. An error message is displayed if this fill fails.

The screenshot shows a Microsoft Word document window titled "StudentForm". In the top right corner, there are two dropdown menus: "FindName" and another one partially visible. The main content area contains the following VB.NET code:

```

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 Is = "Andrew Woods"
            strName = "Andrew.jpg"
        Case Is = "Blue Valley"
            strName = "Blue.jpg"
        Case Else
            strName = "No Match"
    End Select
    Return strName
End Function

```

**Figure 4.121.** The coding for the subroutine FindName.

- K. A cleaning job is performed to release all data objects used in this event procedure.

The coding for the subroutine FindName() is identical to what we developed in Section 4.17.5.2. For your convenience, this coding is shown again in Figure 4.121.

In order to pick up the correct student's image file from this subroutine, one point you need to note is that you must store all 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 4\SQLSelectRTOBJECT\bin\Debug**. If you place those student image files in another folder, you must provide the 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.

The coding for the BuildCommand() subroutine is shown in Figure 4.122.

The coding for the subroutine FillStudentTextBox() is shown in Figure 4.123. In all, we need six pieces of information for the selected student, so a six-dimension text box array StudentTextBox(5) is used (the index is based on 0).

The coding for the subroutine MapStudentTextBox() is shown in Figure 4.124.

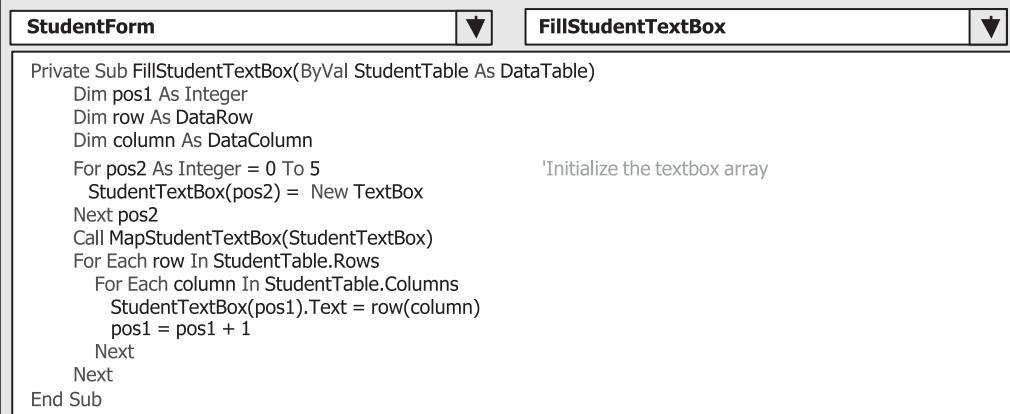
The screenshot shows a Microsoft Word document window titled "StudentForm". In the top right corner, there are two dropdown menus: "BuildCommand" and another one partially visible. The main content area contains the following VB.NET code:

```

Private Sub BuildCommand(ByRef cmdObj As OleDbCommand, ByVal cmdString As String)
    cmdObj.Connection = LogInForm.sqlConnection
    cmdObj.CommandType = CommandType.StoredProcedure
    cmdObj.CommandText = cmdString
End Sub

```

**Figure 4.122.** The coding for the subroutine BuildCommand.



The screenshot shows a Microsoft Visual Studio code editor window. The title bar says "StudentForm" and the tab bar says "FillStudentTextBox". The code is as follows:

```

Private Sub FillStudentTextBox(ByVal StudentTable As DataTable)
    Dim pos1 As Integer
    Dim row As DataRow
    Dim column As DataColumn
    For pos2 As Integer = 0 To 5
        StudentTextBox(pos2) = New TextBox           'Initialize the textbox array
    Next pos2
    Call MapStudentTextBox(StudentTextBox)
    For Each row In StudentTable.Rows
        For Each column In StudentTable.Columns
            StudentTextBox(pos1).Text = row(column)
            pos1 = pos1 + 1
        Next
    Next
End Sub

```

**Figure 4.123.** The coding for the subroutine FillStudentTextBox.

The purpose of this coding is to set a one-to-one relationship between each element in the text box array and each associated text box control in the Student form.

The coding for the subroutine FillCourseList() is shown in Figure 4.125.

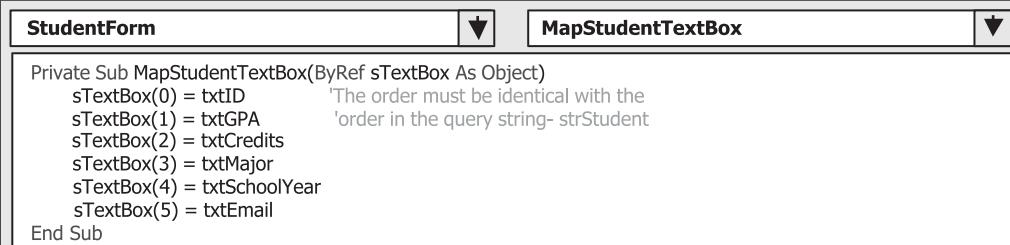
Finally, don't forget to develop code for the Back command button. Enter Me.Close() into this event procedure.

Now we can begin to test the calling of those two stored procedures from our Visual Basic.NET project. Click the Start Debugging button to run the project, enter the username and password, and select the Student Information item to open the Student form window, which is shown in Figure 4.126.

Select Ashly Jade from the Student Name combo box and click the Select button. All information related to this student and the courses is displayed in six text boxes and the Course list box, which is shown in Figure 4.126.

The calling of these two stored procedures was successful!

Some readers may find that these two stored procedures are relatively simple, and each procedure only contains one SQL statement. Let's dig a little deeper and develop some sophisticated stored procedures and try to call them from our Visual Basic.NET project. Next, we will develop a stored procedure that contains more SQL statements.



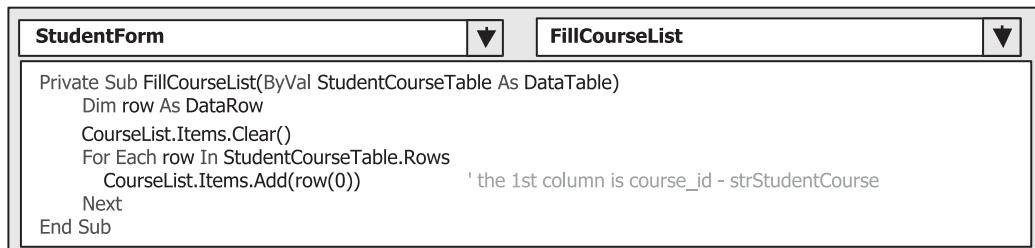
The screenshot shows a Microsoft Visual Studio code editor window. The title bar says "StudentForm" and the tab bar says "MapStudentTextBox". The code is as follows:

```

Private Sub MapStudentTextBox(ByRef sTextBox As Object)
    sTextBox(0) = txtID          'The order must be identical with the
    sTextBox(1) = txtGPA         'order in the query string- strStudent
    sTextBox(2) = txtCredits
    sTextBox(3) = txtMajor
    sTextBox(4) = txtSchoolYear
    sTextBox(5) = txtEmail
End Sub

```

**Figure 4.124.** The coding for the subroutine MapStudentTextBox.



```

StudentForm FillCourseList
Private Sub FillCourseList(ByVal StudentCourseTable As DataTable)
    Dim row As DataRow
    CourseList.Items.Clear()
    For Each row In StudentCourseTable.Rows
        CourseList.Items.Add(row(0))      ' the 1st column is course_id - strStudentCourse
    Next
End Sub

```

Figure 4.125. The coding for the subroutine FillCourseList.

#### 4.18.9 Query Data Using More Complicated Stored Procedures

What we want to do is to get all courses (that is, 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. Usually we need to 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, click the “plus” symbol icon that is next to CSE\_DEPT database folder to connect to our database if this database was already added into the Server Explorer. Otherwise you need to right-click on the Data Connections folder to add and connect to our database.

Right-click on the Stored Procedures folder and select the Add New Stored Procedure item to open the Add Procedure dialog box, and then enter the code shown in Figure 4.127 into this new procedure.



Figure 4.126. The running status of the Student form.

```

A CREATE PROCEDURE dbo.StudentCourseINTO
B (
C     @stdtName VARCHAR(50)
D ) AS
E     DECLARE @stdtID VARCHAR(50)
F     SET @stdtID = (SELECT student_id FROM Student
G         WHERE name LIKE @stdtName)
H     SELECT course_id FROM StudentCourse
I         WHERE student_id LIKE @stdtID
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

```

Figure 4.127. The new stored procedure – StudentCourseINTO.

Let's have a detailed discussion of this coding.

- A. The stored procedure is named dbo.StudentCourseINTO.
- B. The input parameter is the student name, @stdtName, which is a varying-character length variable, with a maximum of 50 characters. All parameters, no matter whether input or output, must be declared inside braces.
- C. The local variable @stdtID is used to hold the returned query result from the first SQL statement, which 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 surrounded by parentheses to indicate that this whole query will return a single piece of data.
- E. The second SQL statement is executed, and this query is used to retrieve all courses (course\_id) taken by the selected student from the StudentCourse table based on the student\_id (@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 popup menu. In the opened dialog box, enter a student's name, Erica Johnson, and then click the OK button to run our procedure. The running result is shown in Figure 4.128.

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 in this project and name it SPForm. Open the project SQLSelectRTOBJECT and select the Project|Add Windows Form menu item, enter SP Form.vb

The screenshot shows the Microsoft Visual Studio interface with the title bar "SQLSelectRTOObject - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has various icons for file operations. The Server Explorer window is open, showing "dbo.StudentC...CSE\_DEPT.MDF". The Output window is active, displaying the following text:

```

Running [dbo].[StudentCourseINTO] ( @stdtName = Erica Johnson ).

course_id
-----
CSC-335
CSC-234A
CSC-439
CSC-432
CSE-439
CSC-333A
No rows affected.
(6 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[StudentCourseINTO].

```

The status bar at the bottom shows "Ready", "Ln 15", "Col 1", "Ch 1", and "INS".

Figure 4.128. The running result of the stored procedure.

into the Name box, and click the Add button to add this new form into our project. Enter SPForm into the name property as the name for this form.

Enlarge the size of this SP Form by dragging the border of the form window, and then open the Student form window. We need to copy all controls on the Student form to our new SP form. On the opened Student form, select the Edit|Select All and Edit|Copy menu items, and then open the SP form and select Edit|Paste to paste all controls we copied from the Student form.

To save time, next we need to copy most coding from the Student code window to our new SP form code window. The only exceptions are the coding for the Select button event procedure, cmdSelect\_Click(), and the coding for the subroutine BuildCommand(). Don't copy these two pieces of coding since we need to develop new coding to test our stored procedure later. To copy all other coding, open the code window of the Student form, select the coding, except the coding for the cmdSelect\_Click() event procedure and subroutine BuildCommand(), and then paste it into our new SP form code window.

Now let's develop the coding for our Select button event procedure. Most of the coding is identical to what we developed for the Student form, such as the subroutines FindName() and FillCourseReader(). Open the Select button click event procedure by double-clicking the Select button from the Designer window, and enter the code shown in Figure 4.129 into this event procedure.

Let's discuss this piece of coding step by step now.

- The name of our stored procedure, dbo.StudentCourseINTO, must be declared first and must be identical to the name we used when we created our stored procedure in the Server Explorer window.
- A Command object and a DataReader object are declared here since we need to use them for our data query operation.

**A**      cmdSelect

**B**      Click

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim strStudentCourse As String = "dbo.StudentCourseINTO"
    Dim sqlCmdStudentCourse As New SqlCommand
    Dim sqlDataReader As SqlDataReader
    Dim strName As String
    strName = FindName(ComboName.Text)
    If strName = "No Match" Then
        MessageBox.Show("No Matched Student Found!")
        Exit Sub
    End If
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage
    PhotoBox.Image = System.Drawing.Image.FromFile(strName)
    sqlCmdStudentCourse.Connection = LogInForm.sqlConnection
    sqlCmdStudentCourse.CommandType = CommandType.StoredProcedure
    sqlCmdStudentCourse.CommandText = strStudentCourse
    sqlCmdStudentCourse.Parameters.Add("@StdName", SqlDbType.Char).Value = ComboName.Text
    sqlDataReader = sqlCmdStudentCourse.ExecuteReader
    If sqlDataReader.HasRows = True Then
        Call FillCourseReader(sqlDataReader)
    Else
        MessageBox.Show("No matched course_id found!")
    End If
    sqlDataReader.Close()
    sqlDataReader = Nothing
    sqlCmdStudentCourse.Dispose()
    sqlCmdStudentCourse = Nothing
End Sub

```

**C**

**D**

**E**

**F**

**G**

**H**

**I**

Figure 4.129. The coding for the Select button event procedure.

- 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.
- 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 Name combo box and should be added into the Parameters collection property of the Command object. You need to note that the nominal name @StdName must be identical to 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 certain rows. The subroutine FillCourseReader() is executed to fill the returned course\_id into the Course list box in that situation. An error message is displayed if this call fails.
- I. The cleaning job is performed to release all objects used in this data query operation.

Before you can test this coding, you need to add one more item into the Selection form code window. Open the code window for the Selection form and enter the following code into the SelectForm\_Load() event procedure:

---

```
ComboSelection.Items.Add("SP Information")
```

---

Then add the following code into the OKButton\_Click() event procedure:

---

```
Dim spform As New SPForm
.....
ElseIf ComboSelection.Text = "SP Information" Then
    spform.Show()
```

---

Now run the project, enter a suitable username and password, and then select SP Information from the Selection form to open the SP Form window. Select a student name from the Student Name combo box and click the Select button. All courses taken by the selected student will be displayed in the Course list box, which is shown in Figure 4.130.

The testing result for our stored procedure is very good; is that correct? The answer is yes, but we may do a little more to develop a more sophisticated stored procedure to meet a special requirement. What is this special requirement? We want to develop some nested stored procedures and call a nested stored procedure from our main or parent stored procedure. Sounds complicated. Is it? Yes, but we will try to simplify this complicated issue with the following example.

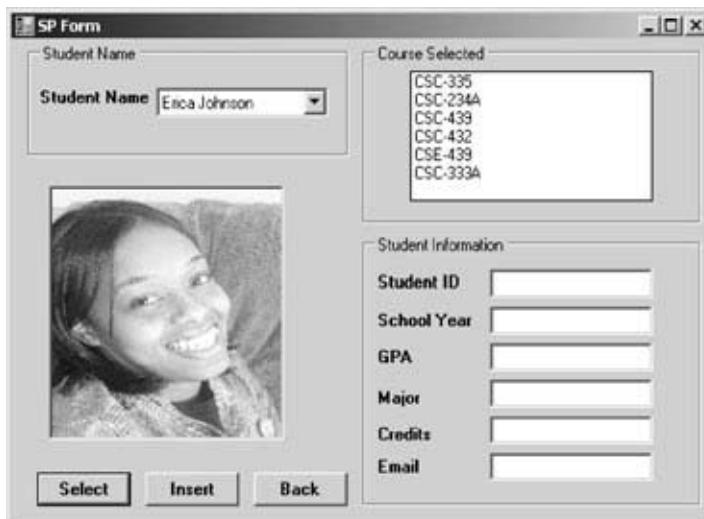


Figure 4.130. The running status of calling stored procedure.

The screenshot shows the Microsoft Visual Studio interface with the title bar "SQLSelectRTOObject - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. Below the menu is a toolbar with various icons. The main window displays a code editor for a stored procedure named "dbo.StudentAndCourse". The code is as follows:

```

A CREATE PROCEDURE dbo.StudentAndCourse
B (
C     @StudentName VARCHAR(50)
D )
E AS
F     DECLARE @StudentID VARCHAR(50)
G     EXEC dbo.StudentInfoID @StudentName, @StudentID OUTPUT
H
I     SELECT course_id FROM StudentCourse
J     WHERE student_id LIKE @StudentID
K
L     RETURN

```

The code editor has line numbers A through L on the left. The status bar at the bottom shows "Ready", "Ln 12", "Col 1", "Ch 1", and "INS".

Figure 4.131. The main stored procedure.

#### 4.18.10 Query Data Using Nested Stored Procedures

So-called nested stored procedures are very similar to subroutines or subqueries, which means that a main stored procedure can be considered as a parent stored procedure, and a substored procedure can be considered as a child stored procedure. The parent stored procedure can call the child stored procedure as it likes. In this section, we try to create two stored procedures, the main and the child stored procedures. The main stored procedure, which is named `StudentAndCourse`, is used to get all courses taken by the selected student from the `StudentCourse` table based on the `student_id`, and the child stored procedure, which is named `StudentInfoID`, is used to get the `student_id` from the `Student` table based on the input student name.

Now open the Server Explorer window and connect to our database `CSE_DEPT`. After the database is connected, right-click on the `Stored Procedures` folder and select the `Add New Stored Procedure` item to open the `Add Procedure` dialog box.

First, let's create our main stored procedure, `dbo.StudentAndCourse`. Enter the code shown in Figure 4.131 into this stored procedure.

The functionality of each line of the coding is as follows:

- The name of our main stored procedure is declared at first, which is `dbo.Student-AndCourse`. This name must be identical to the name of the stored procedure used in our Visual Basic.NET project later.
- The input parameter `@StudentName` is declared here.
- The local variable `@StudentID` is used as an output parameter for the child stored procedure that will return this parameter, `student_id`, to our main procedure.
- The child stored procedure is called to be executed using the command `EXEC`. This calling passes two parameters to the child stored procedure: the input parameter to the child procedure `@StudentName`, and the

The screenshot shows the Microsoft Visual Studio interface with the title bar "SQLSelectRTOObject - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. Below the menu is a toolbar with various icons. The main window displays a SQL script editor containing the following code:

```

A CREATE PROCEDURE dbo.StudentInfoID
(
B     @sName VARCHAR(50),
C     @sID VARCHAR(50) OUTPUT
)
AS
C     SET @sID = (SELECT student_id FROM Student WHERE name LIKE @sName)
D     RETURN

```

The code is annotated with letters A through D pointing to specific parts of the code. The status bar at the bottom shows "Ready", "Ln 9", "Col 1", "Ch 1", and "INS".

Figure 4.132. The child stored procedure.

output parameter @StudentID. The latter must be indicated with the keyword OUTPUT. Later, you must also declare this parameter as an output parameter in the child stored procedure using the keyword OUTPUT to match its definition in this main stored procedure.

- E. After the child stored procedure is executed, it returns the student\_id. Now we can perform our main query to obtain all courses taken by the selected student from the StudentCourse table based on the student.id returned by the child stored procedure.
- F. The retrieved courses are returned to the calling procedure developed in Visual Basic.NET.

Click the File|Save StoredProcedure1 menu item to save our main stored procedure.

Second, let's create our child stored procedure. Right-click on the Stored Procedures folder and select the Add New Stored Procedure item from the popup menu. In the open dialog box, enter the code shown in Figure 4.132 into this new stored procedure.

The functionality for each coding line is explained below:

- A. The name of our child stored procedure, which is dbo.Student-InfoID, is declared first. This name must be identical to the name used by our main stored procedure when this child procedure is called.
- B. Two parameters are declared here; the first, @sName, is the input parameter, and the second, @sID, is the output parameter. The default type of the parameter is INPUT, so the keyword OUTPUT must be attached for the second parameter since it is an output and will be returned to the main stored procedure.
- C. The SQL statement is executed to get the desired student\_id from the Student table based on the input student name. The returned student\_id will be assigned to the output parameter @sID by using the SET command.
- D. The output parameter @sID is returned to the main stored procedure.

The screenshot shows the Microsoft Visual Studio interface with the title bar "SQLSelectRTOBJECT - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has various icons for file operations. The Server Explorer window is open, showing a connection to "dbo.StudentIn... (CSE\_DEPT.MDF)". The Output window is active, displaying the results of a stored procedure execution:

```

Running [dbo].[StudentAndCourse] ( @StudentName = Erica Johnson ).

course_id
-----
CSC-335
CSC-234A
CSC-439
CSC-432
CSE-439
CSC-333A
No rows affected.
(6 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[StudentAndCourse].

```

The status bar at the bottom shows "Ready", "Ln 15", "Col 1", "Ch 1", and "INS".

Figure 4.133. The running result of nested stored procedure.

Click the File|Save StoredProcedure1 menu item to save our child stored procedure.

To test both the main and child stored procedures in the Server Explorer window, right-click the main stored procedure StudentAndCourse item, and then select the Execute item to open the Run Stored Procedure dialog. Enter a student name, such as Erica Johnson, and click the OK button to run both stored procedures. The running result is shown in Figure 4.133.

Our nested stored procedures work fine!

To call these nested stored procedures, we need to develop a Visual Basic.NET project. In order to save time and space, you can use the coding we did for the SP Form window in the last subsection. All coding is the same and the only modifications are to the stored procedure's name declared in the cmdSelect\_Click() event procedure, and the nominal input parameter name in the stored procedure. Change the name of the stored procedure from dbo.StudentCourseINTO to dbo.StudentAndCourse (refer to step A in Figure 4.129), and change the nominal parameter's name from "@StdtName" to "@StudentName" (refer to step G in Figure 4.129). Then you can run this project to get the same result, which is shown in Figure 4.130, as we got from the last project.

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 using the Fill() method of the TableAdapter class to fulfill the same functionality as the DataReader did in this project. We will leave this job as homework for this chapter.

At this point, we have finished developing a data-driven project using runtime objects for the SQL Server database. Now let's go to the last part of this chapter – developing a data-driven application using runtime objects with the Oracle database.

## 4.19 BUILD A SAMPLE ORACLE DATABASE PROJECT – ORACLESELECTRTOBJECT

For convenience, in this section we install Oracle Database 10g Express Edition on our local computer. Whether the Oracle server is installed on a local or a remote computer for this sample project makes no difference. The Oracle database used in this sample project was developed in Chapter 2.

### 4.19.1 Install Oracle Database 10g Express Edition

In this section, we use Oracle Database 10g Express Edition (XE) as our database provider. Refer to Appendix C for detailed procedures to download and install this software on your computer.

Oracle Database 10g XE is an entry-level, small footprint starter database with the following advantages:

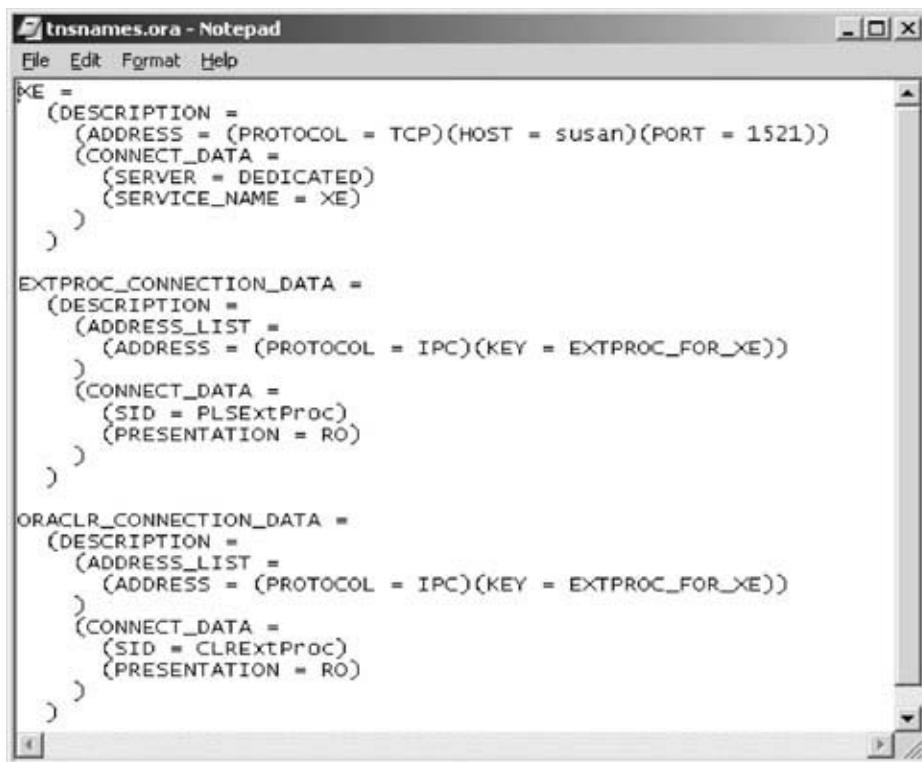
- Free to download and install on your local computer or remote computers
- Free to develop and deploy data-driven applications
- Free to distribute (including ISVs)

Oracle Database 10g the XE is built using the same code base as the Oracle Database 10g Release 2 product line – Standard Edition One, Standard Edition, and Enterprise Edition. It is available on 32-bit Windows and Linux platforms.

Although there are limitations of Oracle Database 10g XE, such as a 4 GB upper bound of the user data and the limit of only a single instance 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:

- Oracle Database 10g XE can be easily upgraded to Standard Edition One, Standard Edition, and Enterprise Edition.
- Any application developed for Oracle Database 10g XE will run completely unchanged on Oracle Database 10g Standard Edition One, Standard Edition, or Enterprise Edition, so the application development investment is guaranteed.
- With Oracle Database 10g XE, independent software vendors have the industry's leading database technology to power their applications. Distributing Oracle Database 10g XE in their applications or products without additional cost offers even greater value to their customers.
- Oracle Database 10g 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 10g 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 10g XE, so that you can connect to the database from the same computer on which you installed the Server component, and then administer the database and develop Visual Studio.NET applications.



The screenshot shows a Windows Notepad window titled "tnsnames.ora - Notepad". The content of the file is as follows:

```

XE =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = susan)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = XE)
    )
  )

EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC_FOR_XE))
    )
    (CONNECT_DATA =
      (SID = PLSExtProc)
      (PRESENTATION = RO)
    )
  )

ORACLR_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC_FOR_XE))
    )
    (CONNECT_DATA =
      (SID = CLRExtProc)
      (PRESENTATION = RO)
    )
  )

```

Figure 4.134. The sample of the file tnsnames.ora.

#### 4.19.2 Configure the Oracle Database Connection String

As we mentioned in Section 4.18.1, 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 Oracle Database 10g 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 you want to create, edit, and manipulate it in the future.

In order to use the database alias defined in the tnsnames.ora file, 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:\oraclexe\app\oracle\product\10.2.0\server\NETWORK\ADMIN after Oracle Database 10g XE is installed. A sample file is shown in Figure 4.134. You can open this file using any text editor such as NotePad, WordPad, or Microsoft 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 Figure 4.134).

Close this file, and now let's create our connection string for Oracle database 10g XE using the database alias XE.

The connection string can be defined as

---

```
Dim oraString As String = "Data Source = XE;" +_
    "User ID = system;" +_
    "Password = reback"
```

---

where reback is the password we used when we installed Oracle Database 10g XE on our computer.

Another way to create the connection string is to copy the top block from the tnsnames.ora file and paste it as the value of the Data Source parameter, which is

---

```
Dim oraString As String = "Data Source = (DESCRIPTION = " +_
    "(ADDRESS = (PROTOCOL = TCP)(HOST = susan)(PORT = 1521))" +_
    "(CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME = XE));" +_
    "User ID = system;Password = reback;"
```

---

In the following sample project, we used the first way to create our connection string. With the connection string ready, we can start to develop our sample project.

#### 4.19.3 Query Data Using Runtime Objects for the LogIn Form

Open Visual Studio.NET 2005 and create a new Windows-based project named OracleSelectRTOBJECT. As we did for the last section, delete the default form Form1 and add five form windows from the last project. Refer to Section 4.18.2 to add those five forms and modify the coding for the file Application.Designer.vb. Make sure that the LogIn form is the start form in this project by checking the Project|OracleSelectRTOBJECT Properties window.

Open the LogIn form window by clicking the View Designer button from the Solution Explorer window. Then we need to open the Oracle Client namespace in which the Oracle Data Provider and associated classes are located, and make this namespace a reference for our project since we need to use those classes for our Oracle database operations later. Right-click on the OracleSelectRTOBJECT from the Solution Explorer window and select the Add Reference item from the popup menu to open the Add reference window. With the .NET tab selected, scroll down the list until you find the item System.Data.OracleClient, click it to select it, and click the OK button to add this reference to our project.

Some readers may have found a discrepancy, which is that we did not perform this reference adding job for our previous projects, either AccessSelectRTOBJECT or SQLSelectRTOBJECT. The reason for this is that the namespaces for those two Data Providers are default namespaces and are added automatically by Visual Basic.NET 2005 as you open a new project.

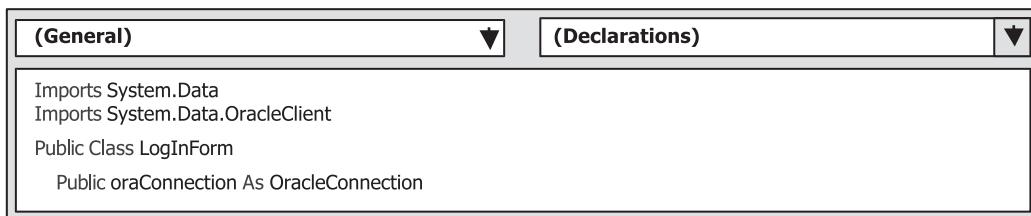


Figure 4.135. The declaration of the namespace for the Oracle Data Provider.

#### 4.19.3.1 Declare the Runtime Objects

As mentioned in Chapter 3, all components related to the Oracle Data Provider supplied by ADO.NET are located at the namespace System.Data.OracleClient. To access the Oracle database, you need to use this Data Provider. You must first declare this namespace at the top of your code window to allow Visual Basic.NET 2005 to know that you want to use this specified Data Provider. Open the code window by clicking the View Code button from the Solution Explorer window and enter the code shown in Figure 4.135 at the top of this code window.

A new instance of the OracleConnection class is created with the accessing mode of Public, which means that we want to use this connection object for our whole project.

The first job you need to do after a new instance of the data connection object is declared is to connect your project with the database you selected.

#### 4.19.3.2 Connect to the Data Source with the Runtime Object

Since the connection job is the first thing you need to do before you can perform any data queries, you need to do the connection in the LogInForm\_Load() event procedure to allow the connection to be made first as your project runs.

In the code window, click the drop-down arrow on the Class Name combo box and select 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. Modify the code in this event procedure as shown in Figure 4.136.

To simplify the coding in this part, change all “sql” prefixes preceding each Data Provider-dependent object, such as sqlConnection and sqlExceptionErr, to “ora” to get a new group of Oracle-related objects such as oraConnection and oraExceptionErr.

Let's see how this piece of code works.

- The Oracle database connection string is defined first. Refer to Section 4.19.2 for a detailed discussion of this connection string. An addition operator (+) can be used to concatenate multiple substrings to form a complete connection string for the Oracle database.
- 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.

	<b>(LogInForm Events)</b>	<b>Load</b>
--	---------------------------	-------------

```

Private Sub LogInForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    Dim oraString As String = "Data Source=XE;" + _
        "User ID=system;" + _
        "Password=reback"
    oraConnection = New OracleConnection(oraString)
    Try
        oraConnection.Open()
    Catch oraExceptionErr As OracleException
        MessageBox.Show(oraExceptionErr.Message, "Oracle Error")
    Catch InvalidOperationExceptionErr As InvalidOperationException
        MessageBox.Show(InvalidOperationExceptionErr.Message, "Oracle Error")
    End Try
    If oraConnection.State <> ConnectionState.Open Then
        MessageBox.Show("Database connection is Failed")
        Exit Sub
    End If
End Sub

```

Figure 4.136. The coding for the LogInForm\_Load() event procedure.

- C. A Try...Catch block is utilized here to try to catch any mistakes caused by opening this connection. The advantage of using this kind of strategy is to avoid an unnecessary system debug process or simplify this debug procedure.
- D. This step is used to confirm that our database connection is successful. If not, an error message is displayed and the project is exited.

After a database connection is successfully made, we need to use this connection to access the Oracle database to perform our data query job. As we did for the previous projects, we will use two methods to perform the data query, the TableAdapter method and the DataReader method.

#### **4.19.3.3 Coding for Method 1: Using the TableAdapter to Query Data**

Open the LogIn form window by clicking the View Designer button, and then double-click the TabLogIn button to open its event procedure. Modify the code in this event procedure as shown in Figure 4.137.



The SELECT statement used for Oracle databases 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 to the parameter to be assigned to the column.

Let's see how this piece of code works.

- A. The SELECT statement for Oracle is basically the same as for the SQL Server database, but a slight difference exists. The difference is the format

**TabLogIn**      **Click**

```

Private Sub TabLogIn_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TabLogIn.Click
    Dim cmdString1 As String = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "
    Dim cmdString2 As String = "WHERE user_name=:UserName AND pass_word=:PassWord"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim oraUserName As New OracleParameter
    Dim oraPassWord As New OracleParameter
    Dim LogInTableAdapter As New OracleDataAdapter
    Dim oraDataTable As New DataTable
    Dim oraCommand As New OracleCommand
    Dim selForm As New SelectionForm

    C oraUserName.ParameterName = "UserName"           "Very important in some applications
    C oraUserName.OracleType = OracleType.Char
    C oraUserName.Value = txtUserName.Text
    C oraPassWord.ParameterName = "PassWord"           "Very important in some applications
    C oraPassWord.OracleType = OracleType.Char
    C oraPassWord.Value = txtPassWord.Text
    oraCommand.Connection = oraConnection
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add(oraUserName)
    oraCommand.Parameters.Add(oraPassWord)
    LogInTableAdapter.SelectCommand = oraCommand
    LogInTableAdapter.Fill(oraDataTable)

    D If oraDataTable.Rows.Count > 0 Then
        selForm.Show()
        Me.Hide()
    Else
        MessageBox.Show("No matched username/password found!")
    End If

    E oraDataTable.Dispose()
    oraDataTable = Nothing
    oraCommand.Dispose()
    oraCommand = Nothing
    LogInTableAdapter.Dispose()
    LogInTableAdapter = Nothing

    F End Sub

```

Figure 4.137. The coding for the TabLogIn button event procedure.

of assigning the parameter in the WHERE clause. In both Microsoft Access and SQL Server, the @ symbol is prefixed to the parameter and an equal symbol or the word LIKE is used to assign a parameter to the column. In Oracle, an equal symbol is still used, but a colon must be prefixed to 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).

- 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.
- Two dynamic parameters are assigned to the OracleParameter objects, paramUserName and paramPassWord, separately. The parameter's name must be identical to the name of the dynamic parameter in the SQL statement string. The parameter's type (OracleType.Char) must be indicated clearly,

although it may work without this indication. The Values of the two parameters should be equal to the contents of two associated text box controls, which will be entered by the user as the project runs.

- D. Now the 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.
- 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, the Selection form, will be displayed to continue to the next step. Otherwise an error message will be displayed.

The last part of the code is used to clean and release the objects used for this data query and is identical to the coding in the last project, with one exception: all prefixes should be changed to “ora.”

#### **4.19.3.4 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 the ReadLogIn button to open its event procedure. Modify the code in this event procedure as shown in Figure 4.138.

Most code in the top section is identical to the code 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. As we did in the coding for method 1, the parameter must be preceded by a colon for the WHERE clause in the second query string. This is a 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 to that defined in the SQL query statement. The same 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 of the coding is identical to the coding we did in the last section. The only issue you need to pay attention to is that all prefixes of the objects used in this

	ReadLogIn	▼	Click	▼
A	<pre>Private Sub ReadLogIn_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ReadLogIn.Click     Dim cmdString1 As String = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "     Dim cmdString2 As String = "WHERE user_name=:UserName AND pass_word=:PassWord"     Dim cmdString As String = cmdString1 &amp; cmdString2</pre>			
B	<pre>Dim oraUserName As New OracleParameter Dim oraPassWord As New OracleParameter Dim oraCommand As New OracleCommand Dim oraDataReader As OracleDataReader Dim selForm As New SelectionForm</pre>			
C	<pre>oraUserName.ParameterName = "UserName" oraUserName.OracleType = OracleType.Char oraUserName.Value = txtUserName.Text oraPassWord.ParameterName = "PassWord" oraPassWord.OracleType = OracleType.Char oraPassWord.Value = txtPassWord.Text oraCommand.Connection = oraConnection oraCommand.CommandType = CommandType.Text oraCommand.CommandText = cmdString oraCommand.Parameters.Add(oraUserName) oraCommand.Parameters.Add(oraPassWord) oraDataReader = oraCommand.ExecuteReader</pre>	'Very important in some applications		
D	<pre>If oraDataReader.HasRows = True Then     selForm.Show()     Me.Hide() Else     MessageBox.Show("No matched username/password found!") End If oraCommand.Dispose() oraCommand = Nothing oraDataReader.Close() oraDataReader = Nothing End Sub</pre>	'Very important in some applications		

**Figure 4.138.** The coding for the ReadLogIn button event procedure.

part should be replaced by “ora”, such as in oraCommand, oraDataReader, and so on, since we are using an Oracle database for this section.

The coding for the Cancel command button event procedure is also identical to the coding we did in the last section, with no modification.

If the login is successful, the next form will be the Selection form, which allows users to select different information to view.

#### 4.19.4 The Coding for the Selection Form

Most coding in this form is identical to the coding of the Selection form in the last project. The only difference is the coding for the Exit command button. In the last project, a 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 the prefix “ora”. 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 in the LogIn class, the connection object should be preceded by the class name LogIn. The only modification you need to do is to change the prefix “sql” to “ora” for the connection instance, as shown in Figure 4.139.

cmdExit	▼	Click	▼
---------	---	-------	---

```

Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdExit.Click
    If LogInForm.oraConnection.State = ConnectionState.Open Then
        LogInForm.oraConnection.Close()
        LogInForm.oraConnection.Dispose()
        LogInForm.oraConnection = Nothing
    End If
    Application.Exit()
End Sub

```

Figure 4.139. The coding for the Exit button event procedure.

#### 4.19.5 Query Data Using Runtime Objects for the Faculty Form

First, let's take a look at the coding for the FacultyForm\_Load() event procedure. The differences between this coding and the coding in the last project are as follows:

- The namespace of the Data Provider is different. System.Data.SqlClient is utilized for the last project since an SQL Server database is used. Since we use the Oracle database in this section, change the namespace to System.Data.OracleClient, which is shown in Figure 4.140.
- The FacultyLabel object array used in the last project has seven elements, which are mapped to seven columns in the Faculty table. In this project, a modification is made when we perform the data query. Instead of querying all seven columns, we only query five columns of data, from the third to the last column. Therefore the size of the object array FacultyLabel is reduced to 5. Since the first element's index of the object array is 0,

(FacultyForm)	▼	Load	▼
---------------	---	------	---

```

A Imports System.Data
Imports System.Data.OracleClient
Public Class FacultyForm
B Private FacultyLabel(4) As Label
C Private Sub FacultyForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    If LogInForm.oraConnection.State <> ConnectionState.Open Then
        MessageBox.Show("Database has not been opened!")
        Exit Sub
    End If
    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("DataReader Method")
    ComboMethod.SelectedIndex = 0
End Sub

```

Figure 4.140. The coding for the FacultyForm\_Load() event procedure.

cmdSelect      Click

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString1 As String = "SELECT office, phone, college, title, email FROM Faculty "
    Dim cmdString2 As String = "WHERE name=:FacultyName"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim paramFacultyName As New OracleParameter
    Dim FacultyTableAdapter As New OracleDataAdapter
    Dim oraCommand As New OracleCommand
    Dim oraDataReader As OracleDataReader
    Dim oraDataTable As New DataTable
    paramFacultyName.ParameterName = "FacultyName "
    paramFacultyName.OracleType = OracleType.Char
    paramFacultyName.Value = ComboName.Text
    oraCommand.Connection = LogInForm.oraConnection
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add(paramFacultyName)
    Call ShowFaculty(ComboName.Text)

    If ComboMethod.Text = "TableAdapter Method" Then
        FacultyTableAdapter.SelectCommand = oraCommand
        FacultyTableAdapter.Fill(oraDataTable)
        If oraDataTable.Rows.Count > 0 Then
            Call FillFacultyTable(oraDataTable)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        oraDataTable.Dispose()
        oraDataTable = Nothing
        FacultyTableAdapter.Dispose()
        FacultyTableAdapter = Nothing
    Else
        oraDataReader = oraCommand.ExecuteReader
        If oraDataReader.HasRows = True Then
            Call FillFacultyReader(oraDataReader)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        oraDataReader.Close()
        oraDataReader = Nothing
    End If
    oraCommand.Dispose()
    oraCommand = Nothing
End Sub

```

'Very important for some applications

**Figure 4.141.** The coding for the Select button event procedure.

the upper bound index should be 4 for an object array containing five elements.

- C. The prefix of the connection object is changed to “ora” since an Oracle Data Provider is utilized in the project.

Next is the coding for the Select button event procedure. Open the Faculty form window by clicking the View Designer button from the Solution Explorer window, then double-click the Select button to open its event procedure. Make the following modifications, shown in Figure 4.141, to the original code:

- A. The query string is modified from querying seven to five columns since the first two columns are id and name, and we do not need these two

```

A Private Sub FillFacultyTable(ByVal FacultyTable As DataTable)
    Dim pos1 As Integer
    Dim column As New DataColumn
    Dim row As DataRow
    For pos2 As Integer = 0 To 4                                'Initialize the object array
        FacultyLabel(pos2) = New Label()
        Next pos2
        Call MapFacultyTable(FacultyLabel)
        For Each row In FacultyTable.Rows
            For Each column In FacultyTable.Columns
                FacultyLabel(pos1).Text = row(column)
                pos1 = pos1 + 1
            Next
        Next
    End Sub

```

**Figure 4.142.** The coding for the subroutine FillFacultyTable().

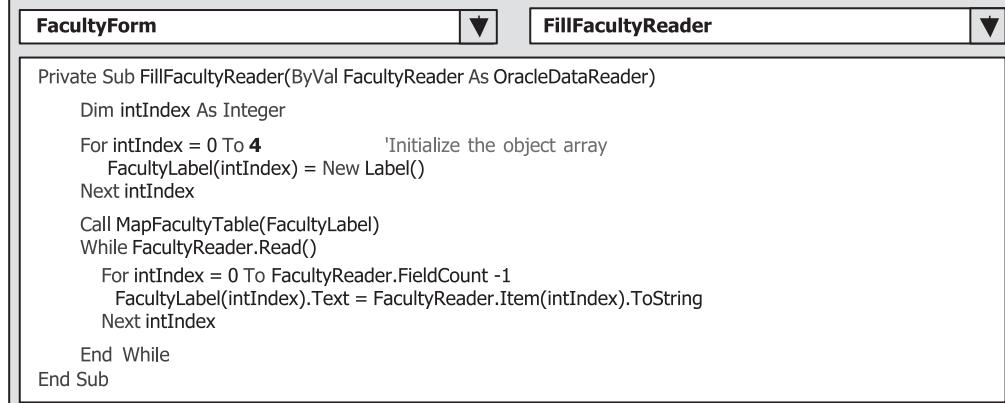
columns. Five labels on the Faculty form window are TitleLabel, OfficeLabel, PhoneLabel, CollegeLabel, and EmailLabel, which are mapped to the five columns in the Faculty table. The dynamic parameter must be prefixed by a colon for the WHERE clause such as “WHERE name=:FacultyName” since this is a requirement of the data query in an Oracle database.

- B. An OracleParameter object is created, and it is used later to hold the dynamic parameter’s name and value.
- 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 value.
- E. The OracleCommand object is initialized by assigning it with three values.

The following code is similar to the code we developed in the same event procedure for the last project. The only modification you need to make for the following code is change the prefix “sql” preceding each object to “ora”, such as sqlCommand to oraCommand, sqlDataTable to oraDataTable, and sqlDataReader to oraDataReader. Yes, it’s that easy!

For three subroutines, FillFacultyTable(), FillFacultyReader(), and MapFacultyTable(), only some little modifications are made. For the first two subroutines, the first modification is to change the upper bound index from 6 to 4 since we reduced the number of queried columns from seven to five. Then change the nominal argument’s type from SqlDataReader to OracleDataReader for the FillFacultyReader() subroutine. Step **A** in Figure 4.142, and Step **A** and **B** in Figure 4.143 show these modifications.

The modification made for the subroutine MapFacultyTable() is to shift the order index of the Label object array. In the last project, we queried seven columns from the Faculty table starting with column 0, which is the faculty\_id, and column 1,



The screenshot shows the Visual Studio code editor with the title bar "FacultyForm" and the tab "FillFacultyReader". The code is as follows:

```

A Private Sub FillFacultyReader(ByVal FacultyReader As OracleDataReader)
    Dim intIndex As Integer
    For intIndex = 0 To 4           'Initialize the object array
        FacultyLabel(intIndex) = New Label()
    Next intIndex
    Call MapFacultyTable(FacultyLabel)
    While FacultyReader.Read()
        For intIndex = 0 To FacultyReader.FieldCount -1
            FacultyLabel(intIndex).Text = FacultyReader.Item(intIndex).ToString()
        Next intIndex
    End While
B End Sub

```

Figure 4.143. The coding for the subroutine FillFacultyReader().

which is the name in the Faculty table. But we do not need these two columns in our project and we want to display the faculty information starting from column 2, which is the office in the Faculty table. So we assign the OfficeLabel to the second element in the Label object array. In this project, we want to query only five columns starting from column 2, which is the office in the Faculty table, so we assign the OfficeLabel to the 0th element in the Label object array. Refer to Figure 4.144 for this modification.

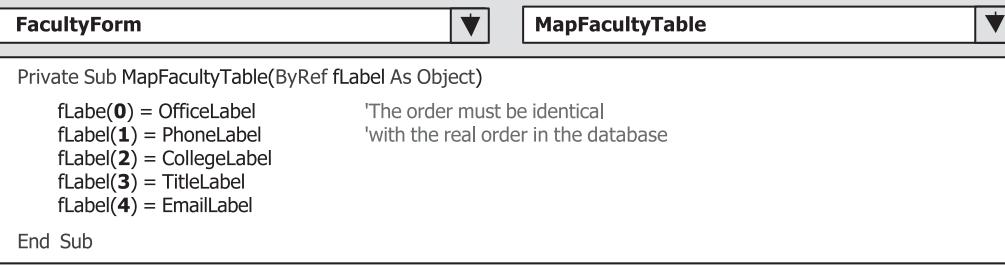
The coding for the subroutine ShowFaculty() and the Back button event procedure is identical to the coding we did for the project AccessSelectRTOBJECT, with no modification. Refer to Figures 4.79 and 4.80 in Section 4.17.3 for the detailed coding and explanations associated with these two pieces of code.

#### 4.19.6 Query Data Using Runtime Objects for the Course Form

Open the Course Form window by clicking the Course Form.vb item and the View Designer button in the Solution Explorer window. The Course Form window is shown in Figure 4.145.

The Query Method selection box allows users to use either the TableAdapter method or the DataReader method to query data from the Course table.

First, let's do the coding for the CourseForm\_Load() event procedure.



The screenshot shows the Visual Studio code editor with the title bar "FacultyForm" and the tab "MapFacultyTable". The code is as follows:

```

Private Sub MapFacultyTable(ByRef fLabel As Object)
    fLabel(0) = OfficeLabel           'The order must be identical
    fLabel(1) = PhoneLabel           'with the real order in the database
    fLabel(2) = CollegeLabel
    fLabel(3) = TitleLabel
    fLabel(4) = EmailLabel
End Sub

```

Figure 4.144. The coding for the subroutine MapFacultyTable().

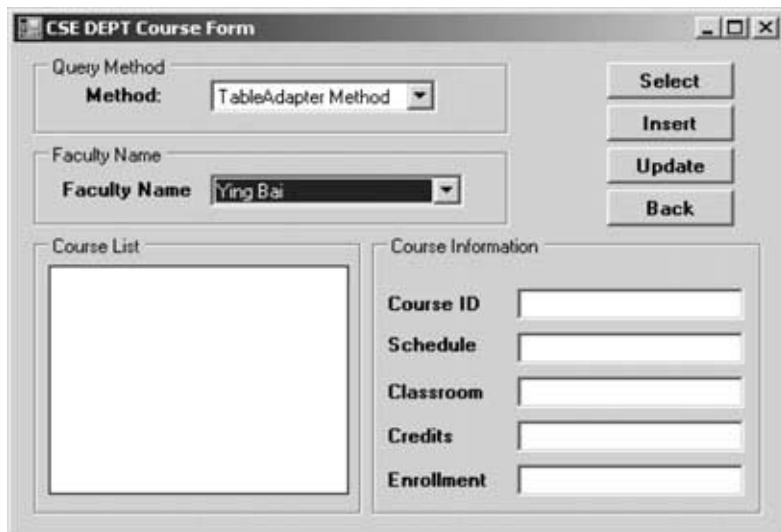


Figure 4.145. The Course Form window.

Basically, the coding of this event procedure is similar to what we did for the same event procedure in the last project. The only modifications (refer to Figure 4.146) are as follows:

- The Data Provider namespace is modified. The System.Data.OracleClient namespace is used to replace System.Data.SqlClient since we use an Oracle data provider in this section.

```
(CourseForm) ▾ Load ▾
A Imports System.Data
Imports System.Data.OracleClient
B Public Class CourseForm
C Private CourseTextBox(4) As TextBox           'We only need 5 columns in Course table
C Private CourseID() As String                 'Declare the CourseID dynamic array
D Private Sub CourseForm_Load(sender As System.Object, ByVal e As System.EventArgs) Handles Me.Load
E     If LogInForm.oraConnection.State <> ConnectionState.Open Then
         MessageBox.Show("Database has not been opened!")
         Exit Sub
     End If
E     ComboName.Items.Add("Ying Bai")
     ComboName.Items.Add("Satish Bhalla")
     ComboName.Items.Add("Black Anderson")
     ComboName.Items.Add("Steve Johnson")
     ComboName.Items.Add("Jenney King")
     ComboName.Items.Add("Alice Brown")
     ComboName.Items.Add("Debby Angles")
     ComboName.Items.Add("Jeff Henry")
     ComboName.SelectedIndex = 0
     ComboMethod.Items.Add("TableAdapter Method")
     ComboMethod.Items.Add("DataReader Method")
     ComboMethod.SelectedIndex = 0
End Sub
```

Figure 4.146. The coding for the CourseForm\_Load() event procedure.

- B. The size of the text box object array is 5 since we only need to query five columns from the Course table and display them in the five associated text box controls on the Course form window.
- C. Recall that in the first project, SelectWizard, the contents displayed in the Course list box are all the names of the courses taught by the selected faculty member when the Select button is clicked in the Course form. Also, the detailed information related to the selected course name in the Course list box will be displayed in five text boxes when the user clicks one course name. Note that the names of the courses in the Course table are not unique, which means that one course may have two sections with the same name. For example, the courses CSC-132A and CSC-132B have the same course name, Introduction to Programming. An error may occur if one tries to use the course name as a criterion to query all detailed information related to that course since the course name is not unique in the Course table. In order to solve this problem, we must use the course\_id, which provides a unique value in the Course table, as a criterion to query the detailed information such as course name, credit, classroom, schedule, and enrollment for that course. For each course name displayed in the Course list box, it has a unique course\_id associated. So we need to use a form-level dynamic string array CourseID() to save each course\_id that is related to each course name when it is selected. This CourseID array will be used in the next event procedure Course-List\_SelectedIndexChanged that will be triggered when the user clicks the course name from the CourseList box. Since the number of courses taught by each faculty member is different and cannot be known at this moment, we declare this string array as a dynamic one. Blank parentheses with no dimension follow the array name CourseID.
- D. Before we can perform any data queries, we need to check whether a valid connection is still open. Since we created the connection instance in the LogIn form with the Public accessing mode, the class name LogIn must precede the connection object here.
- E. This code is used to initialize the Faculty Name and Method Name combo boxes, and select the first item as the default item for both combo boxes.

The next coding job is for the Select button event procedure. After the user selects 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 taught by the selected faculty member.

One point 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 query the Faculty table to find the faculty\_id that is related to the faculty name selected by the user in the Faculty Name combo box on the Course form. Then we need to perform the second query, which is used to find all course\_id and related courses taught by the selected faculty member from the Course table. The queried course names are displayed in the Course list box, and the detailed course

information for each course will be displayed in five text boxes when the user clicks the associated course name from the Course list 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 4.18.6, and the second way is to combine these two queries into one stored procedure and call that stored procedure to perform the queries. We have already discussed how to use stored procedures in the SQL Server database environment in Section 4.18.8 for the data query operations with the Student table. In this section, we want to use a stored procedure in the Oracle database environment to perform these queries. Stored procedures in the SQL Server database and Oracle database environments are different, so in the following section we first need to discuss these two different kinds of stored procedures.

### 4.19.7 Stored Procedures in the Oracle Database Environment

Different database vendors provide different tools to support the development and implementation of stored procedures. For SQL Server 2005, Microsoft provides SQL Server Management Studio and SQL Server Management Studio Express. For the Oracle database, Oracle provides Oracle Database 10g and Oracle Database 10g Express Edition. Different methods can be used to create stored procedures; for example, six methods are shown in Section 4.18.8.1 to create stored procedures for an 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 Oracle Database 10g Express Edition to create stored procedures.

In Section 4.18.8.1, we discussed how to use the Server Explorer provided by Visual Studio 2005 to create stored procedures. Similarly, Visual Studio 2005 also provides the tools to help users to manage stored procedures in the Oracle database environment Server Explorer, although Oracle has provided the Oracle development tools for .NET.

Another important point to understand is that the stored procedures are categorized based on the query type or SQL statement type used by the stored procedure in the Oracle database. In SQL Server 2005, there is no difference between stored procedures using the SELECT, INSERT, UPDATE, or DELETE statements, and all these statements can be used by stored procedures. But in Oracle database, if a stored procedure needs to return data that the stored procedure needs to execute the SELECT statement, that stored procedure must be embedded into a package. The package in Oracle is a class and can contain variables, functions, and procedures. Therefore, stored procedures in Oracle must be divided into two parts, stored procedures and packages. Stored procedures that don't need to return any data (by executing the INSERT, UPDATE, and DELETE statements) can be considered pure stored procedures, and stored procedures that need to return data (by executing the SELECT statement) must be embedded into the package.

#### 4.19.7.1 The Syntax of Creating a Stored Procedure in Oracle

The syntax of creating a stored procedure in Oracle is:

```
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;
```

The keyword REPLACE is used for the modified stored procedures. Recall that in SQL Server 2005, the keyword ALTER is used for any stored procedure that has been modified since it was created. In Oracle, the keyword CREATE OR REPLACE is used to represent any procedure that is either newly created or modified.

Following the procedure's name, all input or output parameters are declared inside the braces. After the keyword AS, the stored procedure's body is displayed. The body begins with the keyword BEGIN and ends with the keyword END. Note that a semicolon must follow each SQL statement and the keyword END.

An example for creating a stored procedure in Oracle is:

```
CREATE OR REPLACE PROCEDURE InsertProcedure
{
    studentId      VARCHAR2,
    name           CHAR,
    credit          NUMBER
}
AS
BEGIN
INSERT INTO Student(student_id, s_name, s_credit)
VALUES(studentId, name, credit, SYSDATE);
END;
```

The length of the data type for each parameter is not necessary since this allows those parameters to have a varying-length value.

#### 4.19.7.2 The Syntax for Creating a Package in Oracle

To create a stored procedure that returns data, one needs to embed the stored procedure into a package. The syntax of creating a package is as follows:

```
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;
```

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, precisely the protocol of the stored procedure, is declared with the input and the output parameters (cursor 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 assign the returned result of the following SELECT statement to the cursor. As above, each statement must end with a semicolon, including the keyword END.

An example of creating a FacultyPackage in Oracle is as follows:

```

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;

```

The stored procedure is named SelectFacultyID with two parameters: the input parameter FacultyName and the output parameter FacultyID. The keywords IN and OUT follow the associated parameter to indicate the input/output direction of the parameter. The length of the stored procedure name is limited to 30 letters in Oracle. Unlike stored procedure names created in SQL Server 2005, there is no prefix applied for each procedure's name.

By now, 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 mentioned above, we want to combine two queries into a stored procedure or package to get all courses taught by the selected faculty: the first query is used to get the faculty\_id from the Faculty table based on the selected faculty name, and the second query is to get all courses taught by the selected faculty member 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 Oracle Database 10g XE. In this application, we prefer to use the Object Browser page to do this job since it provides a graphical user interface.

Unlike the SQL Server database, Visual Studio.NET 2005 does not provide a graphical user interface to help users to directly create, edit, and manipulate Oracle database components such as tables, views, and stored procedures inside the Visual Studio.NET environment. Oracle Database 10g XE does provide Oracle Developer



Figure 4.147. The opened Create Package window.

Tools (ODT) for .NET to allow users to create and manipulate database components such as tables, views, indexes, stored procedures, and packages directly inside the Visual Studio.NET environment by using an Oracle Explorer that is similar to the Server Explorer for the SQL Server database. To use this tool, one needs to install the Oracle Developer Tools for Visual Studio.NET.

In this section, we will use the Object Browser page provided by Oracle Database 10g XE to create our packages without installing and using the ODT.

#### 4.19.8 Create the Faculty\_Course Package for the Course Form

Open the Oracle Database 10g XE home page by going to Start>All Programs|Oracle Database 10g Express Edition|Go To Database Home Page items. Click the Object Browser and select the Create|Package item to open the Create Package window, which is shown in Figure 4.147.

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 clicking the Next button to open the Name page, which is shown in Figure 4.148.

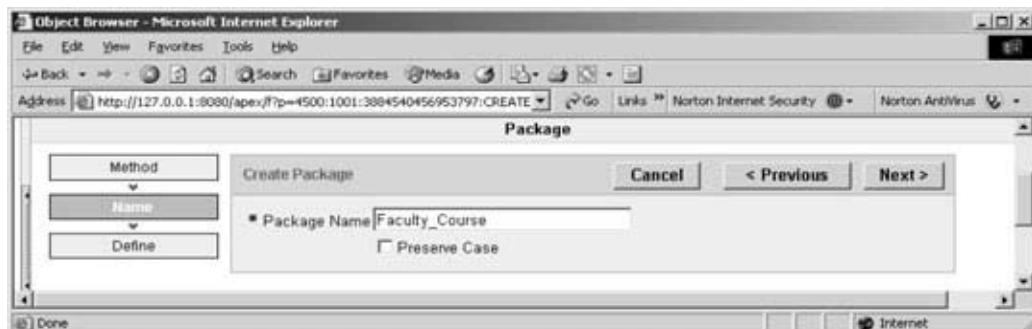


Figure 4.148. The Name page of the Package window.



Figure 4.149. The opened Specification page.

Enter the package name Faculty\_Course into the name box and click the Next button to go to the Specification page, which is shown in Figure 4.149.

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 functions for our application, remove the default function prototype, and change the default procedure name from test to our procedure name – SelectFacultyCourse. Your finished coding for the specification page should match the one that is shown in Figure 4.150.

The coding language we used in this section is called Procedural Language Extension for SQL or PL/SQL, which is a popular language that is widely used in Oracle database programming.

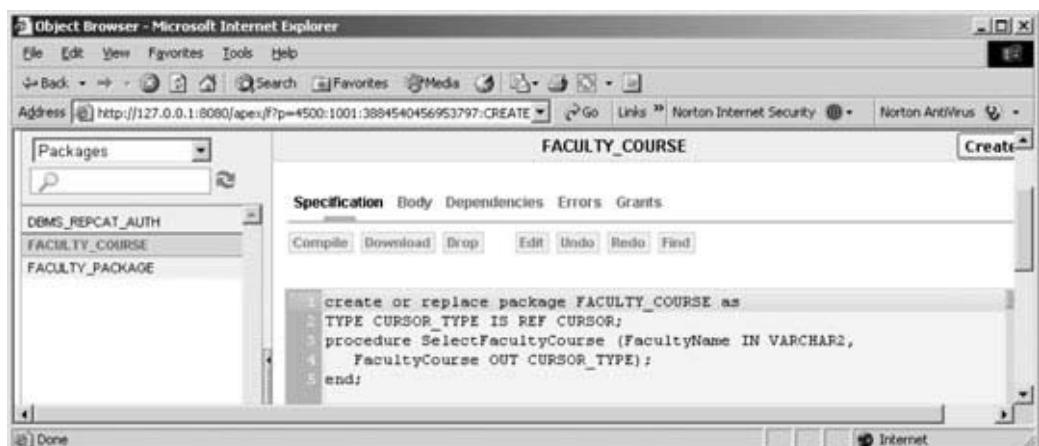


Figure 4.150. The coding for the Specification page.



Figure 4.151. The opened Body page of the package.

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 the keyword OUT. Each PL/SQL statement must be followed by a semicolon, and this rule also applies to the END statement.

Click the Finish button to complete this step. You can click the Compile button to compile this specification block if you like. Next, we need to create the body block of this package. Click the Body tab to open the Body page, which is shown in Figure 4.151.

Click the Edit button to begin to create the body part. Enter the PL/SQL code shown in Figure 4.152 into this body.

The procedure prototype is redeclared in line 2. But the 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, which is located at line 6.

Starting from BEGIN, our real SQL statements are included in lines 6 and 7. The first query is to get the faculty\_id from the Faculty table based on the input parameter FacultyName, which is the first argument of this procedure. The SELECT...INTO statement is utilized to temporarily store the returned faculty\_id into the intermediate variable facultyId.

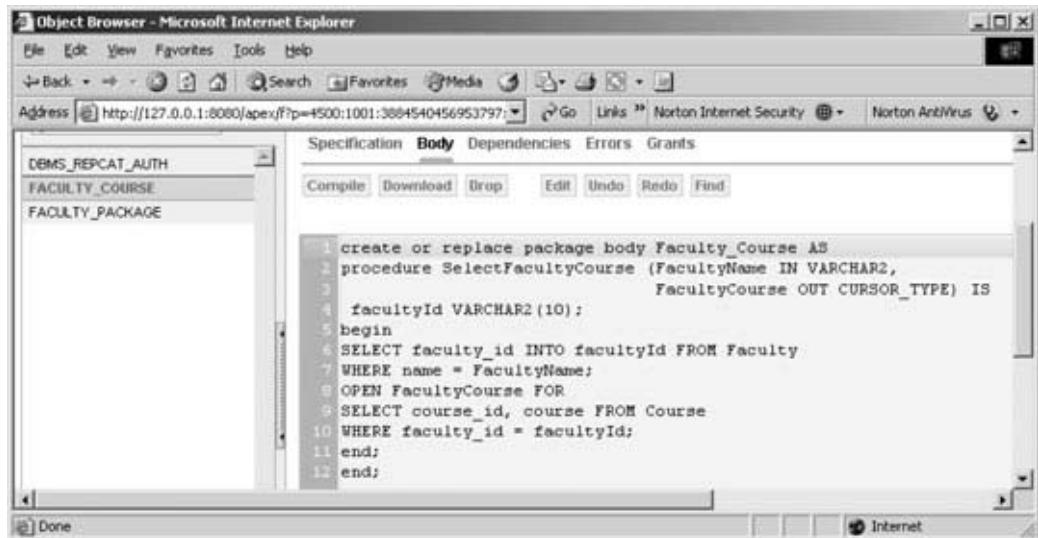


Figure 4.152. The coding for the Body part of the package.

The OPEN FacultyCourse FOR command is used to assign the returned data columns from the following query to the cursor variable FacultyCourse. Recall that we used a SET command to perform this assignment functionality in the SQL Server stored procedure in Section 4.18.9. In lines 9 and 10, the second query is declared, and it is to get all course\_id and courses taught by the selected faculty member from the Course table based on the intermediate variable's value, faculty\_id, which is obtained from the first query above. The queried results are assigned to the cursor variable FacultyCourse.

Now let's compile our package by clicking the Compile button. A successful compiling notice such as

---

**PL/SQL code successfully compiled (22:20:06)**

---

will be displayed if this package is bug-free, as shown in Figure 4.153.

The development of our Oracle package is complete, and now let's go to Visual Studio.NET to call this package to perform the course query for our Course form.

#### 4.19.9 Query Data Using the Oracle Package for the Course Form

Open the Course form window, double-click the Select button to open its event procedure, and enter the code shown in Figure 4.154 into this event procedure.

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

- A. The package query string is declared first, and this string contains both the package name (Faculty.Course) and the stored procedure's name (Select FacultyCourse). All query strings for the Oracle database package must

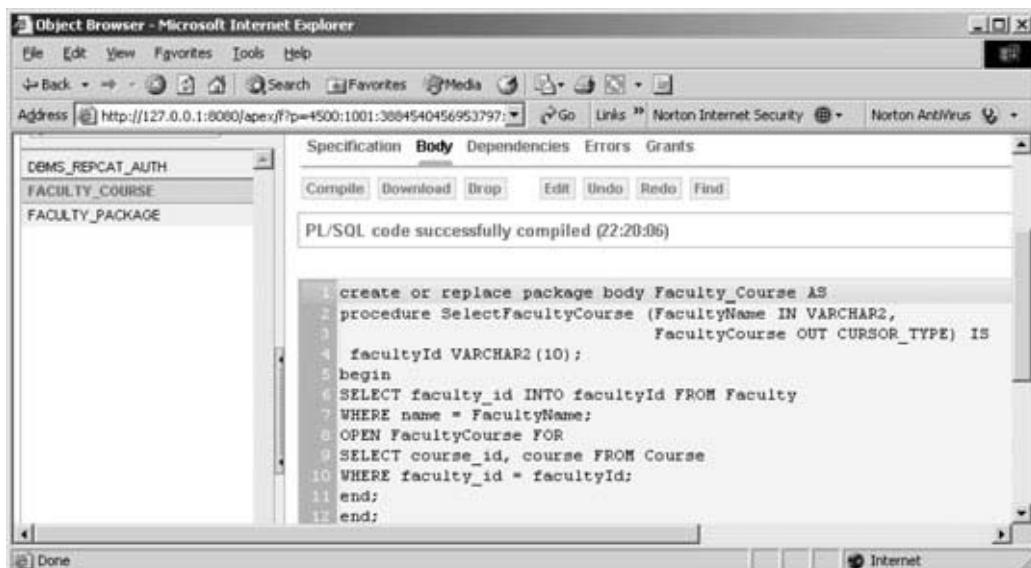


Figure 4.153. The compiled coding for the body part of the package.

follow this style. The package name and the procedure name defined in this string must be identical to the names we used when we created this package in the Object Browser in Oracle Database 10g XE. Otherwise the calling of this package would fail.

- B. All data components used to perform this query are declared and created here. First, two Oracle parameter objects are created: paramFacultyName and paramFacultyCourse. These two parameter objects will be passed into the calling package, and they work as input and output parameters, respectively. Some other components, such as the TableAdapter, Command, Data-Table, and Data Reader, are also created here.
- C. The first parameter object is initialized first. Both the parameter name, FacultyName, and the data type, VARCHAR, must be identical to the name and the data type we used when we created this procedure in Oracle Database 10g 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 the Output to it. Otherwise, the procedure calling may encounter an error, which 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 (step A).

cmdSelect

Click

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString As String = "Faculty_Course.SelectFacultyCourse"
    Dim paramFacultyName As New OracleParameter
    Dim paramFacultyCourse As New OracleParameter
    Dim CourseTableAdapter As New OracleDataAdapter
    Dim oraCommand As New OracleCommand
    Dim oraDataReader As OracleDataReader
    Dim oraDataTable As New DataTable
    paramFacultyName.ParameterName = "FacultyName"
    paramFacultyName.OracleType = OracleType.VarChar
    paramFacultyName.Value = ComboName.Text
    paramFacultyCourse.ParameterName = "FacultyCourse"
    paramFacultyCourse.OracleType = OracleType.Cursor
    paramFacultyCourse.Direction = ParameterDirection.Output
    oraCommand.Connection = LogInForm.oraConnection
    oraCommand.CommandType = CommandType.StoredProcedure
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add(paramFacultyName)
    oraCommand.Parameters.Add(paramFacultyCourse)
    If ComboMethod.Text = "TableAdapter Method" Then
        CourseTableAdapter.SelectCommand = oraCommand
        CourseTableAdapter.Fill(oraDataTable)
    If oraDataTable.Rows.Count > 0 Then
        Call FillCourseTable(oraDataTable)
    Else
        MessageBox.Show("No matched course found!")
    End If
    oraDataTable.Dispose()
    oraDataTable = Nothing
    CourseTableAdapter.Dispose()
    CourseTableAdapter = Nothing
    Else
        oraDataReader = oraCommand.ExecuteReader
        If oraDataReader.HasRows = True Then
            Call FillCourseReader(oraDataReader)
        Else
            MessageBox.Show("No matched course found!")
        End If
        oraDataReader.Close()
        oraDataReader = Nothing
    End If
    oraCommand.Dispose()
    oraCommand = Nothing
    CourseList.SelectedIndex = 0
End Sub

```

Figure 4.154. The coding for the cmdSelect event procedure.

- F. Two initialized parameter objects are added into the Command object, or to be exact, 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 at this moment, the Oracle package we developed is called and two queries are executed. The returned columns should be stored in the Course data table if this fill is successful.

- H. If the Count property of the Course table is greater than 0, which means that at least one row is filled into the table, the subroutine FillCourseTable() is called to fill the queried courses into the Course list box in the Course form window. Otherwise, an error message is displayed to indicate that this fill has failed.
- I. Some cleaning jobs are performed to release some data objects used for this query.
- J. If the user selected the DataReader method, the ExecuteReader() method is executed to invoke the DataReader to retrieve required columns and store the returned results into the DataReader. If the property HasRows is True, which means that the DataReader did read back some rows, the subroutine FillCourseReader() is called to fill the Course list 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. This statement is very important and is used to select the first course in the Course list box as the default course when the Course form is opened. More important, this command can work as a trigger event to trigger the Course list box's SelectedIndexChanged event procedure to display the detailed information related to that default course. The coding for this event procedure is our next job.

The subroutine FillCourseTable() and the Back button event procedure have nothing to do with any object used in this project, so no coding modification is needed. The subroutine FillCourseReader() needs only one small modification, which is to change the nominal argument's type to OracleDataReader since now we are using an Oracle data provider. The coding for subroutines FillCourseTable() and FillCourseReader() is similar to the coding we did in Figure 4.85 in Section 4.17.4. For your convenience, we list this piece of coding with the explanations here again, as shown in Figure 4.155.

Let's see how this piece of code works.

- A. A local DataRow object is created. In order to access the DataTable to pick up any data, one must scan the table by using either DataRow or DataColumn objects, and in most cases, one needs both of them. The DataRow and the DataColumn classes are included in the DataRowCollection and the DataColumnCollection classes, respectively. Although these two objects work like integers, one must use these two objects to access the DataTable to set, retrieve and manipulate the data to and from the DataTable. In this application, we only need to pick up the data located in the first column (course\_id) and the second column (course) of the Course table, so we do not need to use the DataColumn object.
- B. A local integer variable, pos, is declared here and is used as a loop counter later to store the course\_id value into the CourseID array.
- C. Recall that the CourseID is a form-level dynamic string array and its dimension cannot be determined when it is declared since different faculty members teach different numbers of courses. But this number can be determined after this Course table query. The total number of rows returned

	<b>CourseForm</b>	<b>FillCourseTable</b>
A		
B		
C		
D		
E		
F		
G		
H		
I		
J		

```

PrivateSub FillCourseTable(ByVal CourseTable As DataTable)
    Dim row As DataRow
    Dim pos As Integer
    ReDim CourseID(CourseTable.Rows.Count)      'redefine the CourseID array with the definite demission
    CourseList.Items.Clear()
    For Each row In CourseTable.Rows
        CourseList.Items.Add(row(1))           ' the 2nd column is course
        CourseID(pos) = row(0)                ' the 1st column is course_id
        pos = pos + 1
    Next
End Sub

Private Sub FillCourseReader(ByVal CourseReader As OracleDataReader)
    Dim strCourse As String
    Dim pos As Integer
    ReDim CourseID(9)                      'redefine the CourseID array with the definite demission
    CourseList.Items.Clear()
    While CourseReader.Read()
        strCourse = CourseReader.GetString(1)          ' the 2nd column is course
        CourseList.Items.Add(strCourse)
        CourseID(pos) = Convert.ToString(CourseReader.GetString(0))  ' the 1st column is course_id
        pos = pos + 1
    End While
End Sub

```

**Figure 4.155.** The coding for the subroutines FillCourseTable and FillCourseReader.

from this query is the number of courses taught by the selected faculty member. So now the dynamic string array CourseID can be redefined by using the number of courses (returned rows), which is equal to the property CourseTable.Rows.Count.

- D. Before we can fill the Course List box, a cleaning job is needed. This cleaning is very important; multiple repeated courses would be displayed in this list box if you forgot to clean up it first.
- E. A For Each loop is used to scan all rows of the filled Course table. Recall that we filled six columns' data from the Course table in the database to this Course table in the DataTable object starting with the first column, course\_id, and the second column, course (refer to the query string strCourseA defined in the cmdSelect button event procedure, Figure 4.84). Now we need to pick up both the first column's data, course\_id (column index = 0), and the second column's data, course (column index = 1), for each returned row or record of the Course table. Then the Add() method of the ListBox control is used to add each retrieved second column's data (row(1) = course) into the Course List box and save each retrieved first column's data (row(0) = course\_id) into the CourseID array with an appropriate index (pos).
- F. The loop counter pos, which also works as an index for the CourseID array, is updated by 1.
- G. For the FillCourseReader() subroutine, a local string variable strCourse is created. This variable can be considered an intermediate variable that is used to temporarily hold the queried data from the Course table. A local integer

- variable, pos, is created and works as a loop counter later for the While loop and the index for the CourseID array.
- H. The dynamic string array CourseID is redefined here by a definite dimension number, 9. Since the maximum number of courses taught by any faculty member in this department is less than 10, a dimension of 10 is defined to allow the string array to store up to 10 course\_id. This number can be modified based on your real project.
  - I. Similarly, we need to clean up the Course List box before it can be filled.
  - J. A While loop is utilized to retrieve each second column's data (GetString(1)) whose column index is 1 and data value is the course. The queried data is first assigned to the intermediate variable strCourse and then added into the Course List box by using the Add() method. Also, each retrieved first column's data (GetString(0)), whose column index is 0 and data value is the course\_id, is stored into the string array CourseID with an appropriate index (pos). The index pos is updated by 1 for the next loop.

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 credit, classroom, course schedule, and enrollment for the course selected by the user when the user double-clicks a course title from the Course List box. Five text box controls in the Course form are used to store and display the detailed course information.

Now let's begin our coding for this event procedure. Open the Course form window and double-click the Course List box (any place inside that list box) to open this event procedure. Enter the code shown in Figure 4.156 into this event procedure. Let's have a look at this piece of code and see how it works.

- A. The query string is first declared, and we need to retrieve five 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 here, in order to keep the code neat, we still retrieve this column in this query. A nominal parameter courseid that works as a dynamic parameter is assigned to the course\_id column as our query criterion. 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 to some objects since we are using the Oracle data components to perform this query.
- C. The Command object is initialized here by assigning it with associated properties 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 associated with the course selected by the user from the Course List box. We have already obtained this value and stored it in the CourseID array in the subroutines FillCourseTable() or FillCourseReader() in the Select button event procedure above. A little trick

	<b>CourseList</b>	<b>SelectedIndexChanged</b>
	<pre> Private Sub CourseList_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) _ Handles CourseList.SelectedIndexChanged     Dim cmdString1 As String = "SELECT course_id, credit, classroom, schedule, enrollment FROM Course "     Dim cmdString2 As String = "WHERE course_id =:courseid"     Dim cmdString As String = cmdString1 &amp; cmdString2     Dim CourseTableAdapter As New OracleDataAdapter     Dim oraCommand As New OracleCommand     Dim oraDataReader As OracleDataReader     Dim oraDataTable As New DataTable     oraCommand.Connection = LogInForm.oraConnection     oraCommand.CommandType = CommandType.Text     oraCommand.CommandText = cmdString     oraCommand.Parameters.Add("courseid", OracleType.Char).Value = CourseID(CourseList.SelectedIndex)     If ComboMethod.Text = "TableAdapter Method" Then         CourseTableAdapter.SelectCommand = oraCommand         CourseTableAdapter.Fill(oraDataTable)         If oraDataTable.Rows.Count &gt; 0 Then             Call FillCourseTextBox(oraDataTable)         Else             MessageBox.Show("No matched course information found!")         End If     Else         oraDataTable.Dispose()         oraDataTable = Nothing         CourseTableAdapter.Dispose()         CourseTableAdapter = Nothing     Else         oraDataReader = oraCommand.ExecuteReader         If oraDataReader.HasRows = True Then             Call FillCourseReaderTextBox(oraDataReader)         Else             MessageBox.Show("No matched course information found!")         End If     End If     oraDataReader.Close()     oraDataReader = Nothing End If oraCommand.Dispose() oraCommand = Nothing End Sub </pre>	

Figure 4.156. The coding for the SelectedIndexChanged event procedure.

is that the index values are kept unchanged when we add all courses into the Course List 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 table, the subroutine FillCourseTextBox() is called to fill five text box controls with the five retrieved columns. Otherwise, an error message is displayed to indicate that this fill has failed.
- G. A cleaning job is performed here to release some objects used for this fill operation.
- H. If the DataReader method is selected by the user, the ExecuteReader() method is executed to invoke the DataReader to retrieve five columns.

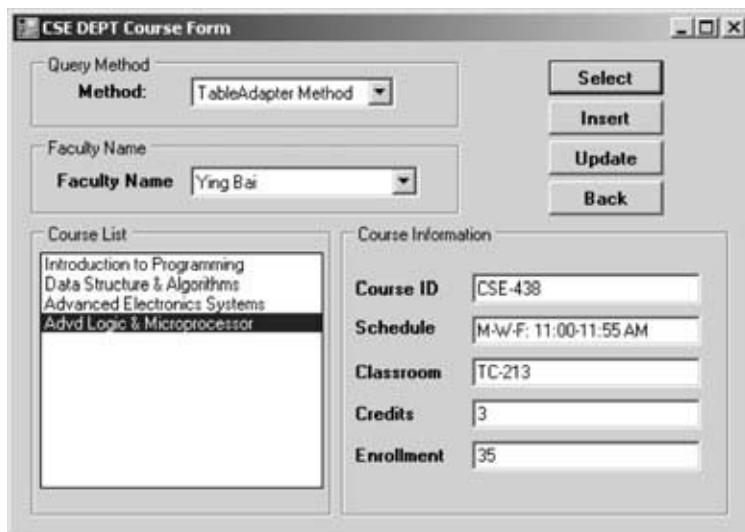


Figure 4.157. The running status of the Course form.

- 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 five text box 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 nominal argument's type 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() (Figures 4.87 and 4.88), refer to Section 4.17.4.

Now let's start this project to test the Course form we developed. Click the Debug| Start Debugging button to run the project. Enter a 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 the Select button to list all courses taught by this faculty member in the Course List box. Then click each course item in the Course List box, and the detailed course information related to the selected course will be displayed in five text box controls on this form, which is shown in Figure 4.157.

At this point we have finished all coding for this form. As for the coding for the Student form, we leave this job as 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 run this project. In this application, it

is the Debug folder of the project. In our case, this folder is located at **C:\Chapter 4\OracleSelectRTOBJECT\bin\Debug**.

## 4.20 CHAPTER SUMMARY

The main topic of this chapter was to develop professional data-driven applications in the Visual Basic.NET 2005 environment by using two methods. The data query was the main point of this chapter.

The first method is to utilize wizards and tools provided by Visual Basic.NET 2005 and ADO.NET to build a simple but powerful project, and the second is to use the runtime object method to build portable projects.

Comparably, the first method is simple, and it is easily understood and learned by students who are beginners to Visual Basic.NET and databases. This method utilizes powerful tools and wizards provided by Visual Basic.NET 2005 and ADO.NET to simplify the coding process, and most of the code is autogenerated by the .NET Framework and Visual Basic.NET 2005 as the user uses those tools and wizards to perform data operations such as adding a new data source, performing 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 for users to get a clear picture of what is really happening behind those tools and wizards. Most of the code is generated by the system automatically in the specific locations, so it is not easy to translate and execute the code in other platforms.

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 coding, it is very easy to translate and execute this kind of project in other platforms. This method provides a clear view for the users and enables users to have a global and detailed picture of how to control the direction of the project with the coding according to the users' desires. The shortcoming of this method is that the amount of coding makes the project complicated and hard for beginners to accept.

Three kinds of databases were discussed in this chapter: Microsoft Access, SQL Server, and Oracle. Each database was explained in detail with a real sample project. Each project used two different data query methods: the TableAdapter method and the runtime object method. A line-by-line illustration was provided for each sample project. In this chapter, readers obtained a solid knowledge of and practical experience in how to develop a professional data query application.

Having finished Part I in this chapter, you should be able to

- Use the tools and wizards provided by Visual Basic.NET 2005 and ADO.NET to develop simple but powerful data-driven applications to perform data queries in Microsoft Access, SQL Server 2005, and Oracle databases.
- Use the OleDbConnection, SqlConnection, or OracleConnection class to connect to Microsoft Access, SQL Server 2005 Express, and Oracle 10g XE databases.
- Perform data binding to a DataGridView using two methods.

- Use the OleDbCommand, SqlCommand, and OracleCommand class to execute a 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.

Having finished Part II in this chapter, you should be able to

- Use runtime objects to develop professional data-driven applications to perform data queries in Microsoft Access, SQL Server 2005, and Oracle databases.
- Use the OleDbConnection, SqlConnection, and OracleConnection class to dynamically connect to Microsoft Access, SQL Server 2005 Express, and Oracle 10g XE databases.
- Use the OleDbCommand, SqlCommand, and OracleCommand class to dynamically execute a 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 the Server Explorer to create, debug, and test stored procedures in Visual Studio.NET environment.
- Use an SQL stored procedure to perform a data query from Visual Basic.NET.
- Use SQL nested stored procedures to perform a data query from Visual Basic.NET.
- Use Object Browser in Oracle Database 10g XE to create, debug, and test stored procedures and packages.
- Use Oracle stored procedures and packages to perform a data query from Visual Basic.NET.

In Chapter 5, we will discuss the data insertion technique with three kinds of databases. Two methods are introduced: in Part I, the tools and wizards provided by Visual Basic.NET 2005 are used to develop a data-inserting query, and in Part II, runtime objects are used to perform the data insertion for three databases.

## 4.21 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 Command object.
- \_\_\_\_\_ 7. The DataSet can be considered 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 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 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 an Oracle database.
- \_\_\_\_\_ 13. You can directly create, edit, manipulate, and test stored procedures for the SQL Server database inside the Visual Studio.NET environment.
- \_\_\_\_\_ 14. To call an SQL Server stored procedure, one must set the CommandType property of the Command object to Procedure.
- \_\_\_\_\_ 15. To set up a dynamic parameter in a SELECT statement in the SQL Server database, the @ symbol must be prefixed to the nominal variable.
- \_\_\_\_\_ 16. The name of the dynamic parameter in a SELECT statement in the SQL Server database may be different from the name of the nominal parameter that is assigned to the Parameters collection of the Command object.
- \_\_\_\_\_ 17. To assign a dynamic parameter to the 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 10g XE.
- \_\_\_\_\_ 19. Two popular ways to query data from any database are using the Fill() method that belongs to the TableAdapter class or calling the ExecuteReader method that belongs to the Command class.

- \_\_\_\_\_20. A DataTable can be considered a collection of DataRowCollection and DataColumnCollection, and the latter contains DataRow and DataColumn objects.

## II. Multiple Choices

1. To connect a database dynamically, one needs to use the \_\_\_\_\_.
  - a. Data Source
  - b. TableAdapter
  - c. Runtime object
  - d. Tools and Wizards
2. Four popular Data Providers are \_\_\_\_\_.
  - a. ODBC, DB2, JDBC, and SQL
  - b. SQL, ODBC, DB2, and Oracle
  - c. ODBC, OLE DB, SQL, and Oracle
  - d. Oracle, OLE DB, 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 an SQL Server database.
  - a. =
  - b. LIKE
  - c. :=
  - d. @=
6. The \_\_\_\_\_ Data Provider can be used to execute the data query for the \_\_\_\_\_ Data Providers.
  - a. SQL Server, OleDb and Oracle
  - b. OleDb, SQL Server and Oracle
  - c. Oracle, SQL Server and OleDb
  - d. SQL Server, Odbc and Oracle

7. To perform a Fill() method to fill a data table, it executes \_\_\_\_\_ object with suitable parameters.
  - a. DataAdapter
  - b. Connection
  - c. DataReader
  - d. Command
8. To fill a list box or 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 modifies an SQL Server stored procedure.
  - a. CREATE
  - b. CREATE OR REPLACE
  - c. REPLACE
  - d. ALTER
12. To perform a runtime data query to an Oracle database, one needs to use \_\_\_\_\_.
  - a. OleDb Data Provider
  - b. Oracle Data Provider
  - c. Both (a) and (b)
  - d. None of the above
13. To query data from any database using the runtime object method, two popular methods are \_\_\_\_\_ and \_\_\_\_\_.
  - a. DataSet, TableAdapter
  - b. TableAdapter, Fill

- c. DataReader, ExecuteReader
  - d. TableAdapter, DataReader
14. To use a stored procedure to retrieve data columns from an Oracle database, one needs to create an \_\_\_\_\_.
  - a. Oracle package
  - b. Oracle stored procedure
  - c. Oracle Trigger
  - d. Oracle Index
15. An Oracle Package has two parts: \_\_\_\_\_ and \_\_\_\_\_.
  - a. Specification, body
  - b. Definition, specifications
  - c. Body, specification
  - d. Specification, execution

### III. Exercises

1. Use the tools and wizards provided by Visual Basic.NET and ADO.NET to complete the data query for the Student form in the AccessSelectWizard project (the project file is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 4\AccessSelect Wizard**).
2. Use runtime objects to complete the data query for the Student form by using the DataReader query method in the AccessSelectRTOBJECT project (the project file is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 4\AccessSelect-RTObject**).
3. Develop a method by adding some code into the cmdLogIn\_Click() event procedure of the project OracleSelectRTOBJECT to allow users to try the login process only three times. A warning message should be displayed and the project should be exited after three failed login attempts.
4. Use Procedural Language Extension for SQL (PL/SQL) to create a package in the Object Browser page of Oracle Database 10g XE. The package contains two stored procedures; one is used to query the student\_id from the Student table based on the input student name, and the second is to query all course\_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 SQLSelectRTOBJECT or the OracleSelectRTOBJECT project to perform the similar data query jobs for the Faculty form.
6. Use the TableAdapter method (Fill()) to perform the data query to the Student and StudentCourse tables for the Student form in the SQLSelectRTOBJECT project. For your reference, a sample project can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) the folder **DBProjects\Chapter 4\**

**SQLSelectRTOBJECT.** 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 the Student and StudentCourse tables using Oracle Database 10g XE. Either the TableAdapter or the DataReader method can be used for this query. The desired way is to use an Oracle package to build this query.

---

# 5

---

## Data Insertion with Visual Basic.NET

We spent a lot of time discussing and explaining two different data query methods in the last chapter. 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 is not inserted into the data tables in the real database. The latter is to insert data into the data tables in the real database.

As you know, ADO.NET provides a disconnected working mode for 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 the DataSet. Most of the time, you are just working on the data between your applications and your data table mappings in your DataSet. The main reason for 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 implementation between the users' applications and the data sources.

In this chapter, we will see how to insert data into the database. Inserting data into the database using Visual Basic.NET design tools and wizards is discussed in the first part, and inserting data into the database using the runtime object method is discussed in the second part.

After finishing this chapter, you will:

- Understand the working principle and structure of inserting data into a database using Visual Basic.NET design tools and wizards
- Understand the procedures to configure the TableAdapter object by using the TableAdapter Query Configuration Wizard and build a 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 of inserting data into the database using the runtime object method
- Design and build stored procedures to perform the data insertion

To successfully complete this chapter, you need to understand topics such as fundamentals of databases, introduced in Chapter 2, and ADO.NET, discussed in Chapter 3. Also, the sample database CSE\_DEPT that was developed in Chapter 2 will be used throughout this chapter.

In order to save time and avoid repetition, we will use the project Sample Wizards 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 to be precise, the event procedures related to those buttons, will be developed and built in this chapter. We will concentrate on the coding for the Insert button in this chapter.

## PART I DATA INSERTION WITH VISUAL BASIC.NET DESIGN TOOLS AND WIZARDS

In this part, we will discuss inserting data into the database using Visual Basic.NET design tools and wizards. We will develop two methods to perform this data insertion: First, we will use the TableAdapter DBDirect method, TableAdapter.Insert(), to directly insert data into the database. Second, we will show readers how to insert data into the database by first adding new records into the DataSet and then updating new records from the DataSet to the database using the TableAdapter.Update() method. Both methods utilize the TableAdapter's direct and indirect methods to complete the data insertion. The database we will use is the Microsoft Access database, CSE\_DEPT.mdb, which was developed in Chapter 2 and is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **database**. You can also try to use other databases such as Microsoft SQL Server 2005 or Oracle Database 10g XE. The only issue is that you need to select and connect to the correct database when you use the Data Source window to set up your data source for your Visual Basic.NET data-driven applications.

### 5.1 INSERT NEW DATA INTO A DATABASE

There are many different ways to insert new data into a database in Visual Studio.NET. Regularly, however, three methods are widely utilized:

1. Using TableAdapter's DBDirect methods, specifically the TableAdapter.Insert() method
2. Using TableAdapter's Update() method to insert new records that have already been added to 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 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 from 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, insertions, and deletions) to the database. A good habit is to try to use the TableAdapter.Insert() method when your application uses objects to store data (for example, you are using TextBox 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 properties such as UpdateCommand and DeleteCommand. The point is that when these properties are executed, data manipulations occur only in the data table in the DataSet, not in the database. In order to make those data modifications occur in the real database, TableAdapter's Update() method is needed to send to 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 the data manipulations, which means that both methods need to execute the Command object, or more precisely, the ExecuteNonQuery() method of the Command object, to finish these data operations on the database.

Because methods 1 and 2 are relatively simple, in this part we will concentrate on inserting data into the database using TableAdapter methods. First, we will discuss how to insert new records directly into the database using the TableAdapter.Insert() method, and then we will see 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 runtime objects.

### **5.1.1 Insert New Records into a Database Using the TableAdapter.Insert Method**

When you use this TableAdapter DBDirect method to perform data manipulations on a database, the main query must provide enough information for the DBDirect methods to be created correctly. The so-called main query is the default or original query method, such as Fill() or GetData(), when you first open any TableAdapter by using the TableAdapter Configuration Wizard. Enough information means that the data table must contain complete 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 5.1 lists three TableAdapter DBDirect methods.

It can be found from Table 5.1 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 is to update all changes made in the DataSet to the database based on the associated properties

**Table 5.1.** 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.

of the TableAdapter, such as InsertCommand, UpdateCommand, and DeleteCommand.

In this chapter, we only take care of inserting data, so only the first two methods are discussed in this chapter. The third method will be discussed in Chapter 6.

### 5.1.2 Insert New Records into a Database Using the TableAdapter.Update Method

To use this method to insert data into a database, one needs to perform the following steps:

1. Add new records to the desired DataTable by creating a new DataRow and adding it to the Rows collection.
2. After the new rows are added to the DataTable, call the TableAdapter.Update method. You can control the amount of data to be updated by passing an entire DataSet, a DataTable, an array of DataRows, or a single DataRow.

In order to provide a detailed discussion and explanation of how to use these two methods to insert new records into a database, a real example will be very helpful. Let's first create a new Visual Basic.NET project to handle these issues.

## 5.2 INSERT DATA INTO THE ACCESS DATABASE USING A SAMPLE PROJECT – INSERTWIZARD

Section 4.2 provided a very detailed introduction about the design tools and wizards in Visual Basic.NET, such as DataSet, BindingSource, TableAdapter, the Data Sources window, the Data Source Configuration Wizard, and DataSet Designer.

We need to use these to develop our sample data insertion project based on the SampleWizards project developed in the last chapter. First, let's copy that project and make some modifications to create it our new project. The advantage of creating our new project in this way is that we don't need to redo the data source connection and configuration since those jobs performed in the last chapter.

### 5.2.1 Create a New Project Based on the SampleWizards Project

Open Windows Explorer and create a new folder, **Chapter 5**. Then open your Internet browser and locate our project SampleWizards that was developed in the last chapter and is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 4**. Copy this project to our new folder **Chapter 5**. Change the name of the solution and the project from SampleWizards to InsertWizard. Double-click InsertWizard Project.vbproj to open this project.

On the opened project, perform the following modifications to get our desired project:

1. Go to the Project|InsertWizard Project Properties menu to open the project's property window. Change the Assembly name from SampleWizards Project to InsertWizard Project and the Root namespace from SampleWizards.Project to InsertWizard.Project.
2. Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to InsertWizard Project. Click OK to close this dialog box.

Go to File|Save All to save these modifications. Now we are ready to develop our graphical user interfaces (GUIs) based on the SampleWizards project we developed in the last chapter.

### 5.2.2 Application User Interfaces

As you know from the last chapter, six form windows work as the user interfaces for the SampleWizards project: LogIn, Selection, Faculty, Course, Student, and Grid. Of all these 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 in our database. First, let's concentrate on the Faculty form to perform the data insertion in our Faculty table in the database. To insert a new record into the Faculty table, a separate user interface or form is needed to allow us to enter the new faculty information.

Now let's create another new form, the Insert Faculty form.

### 5.2.3 Create the Insert Faculty Form Window

The functionality of this Insert Faculty form is that as the project runs, after the user has finished the login process and selected Faculty Information from the Selection form, the Faculty form window will be displayed. When the user clicks the Insert button, the Insert Faculty form window will be shown. This form will allow the user to use two different methods to insert data into the database, the

TableAdapter.Insert() method and the TableAdapter.Update() method. The form will also allow the user to enter all pieces of information into the appropriate text boxes for the newly inserted faculty member. By clicking the Insert button, a new record of a faculty member is inserted into the database. However, if the user wants to remove those pieces of information before finishing this insertion, the Cancel button can be clicked and all information entered will be erased. The Select button is used to validate this data insertion by retrieving the newly inserted data. The Back button is used to allow the user to return to the Selection form to perform other data operations.

Go to the File|Project|Add Windows Form menu to open the Add New Item dialog box. Keep the default template, Windows Form, selected, and enter Insert Faculty Form.vb into the Name box as the name for this new form. Then click the Add button to add this form into our project.

Add the items that are shown in Table 5.2 into this form.

**Table 5.2. Objects for the Insert Faculty Form**

Type	Name	Enabled	Text	TabIndex	DropDownStyle
CheckBox	chkPhoto	True	Faculty Photo	2	
GroupBox	GroupBox1	True	Faculty photo	3	
Label	Label1	True	Photo Name	3.0	
TextBox	txtPhotoName	False		3.1	
Label	Label2	True	Photo Location	3.2	
TextBox	txtPhotoLocation	False	Default Location	3.3	
Label	Label3	True	Method	4	
ComboBox	comboMethod	True		5	DropDownList
GroupBox	GroupBox2	True	Faculty ID and name	0	
Label	Label4	True	Faculty ID	0.0	
TextBox	TxtID	True		0.1	
Label	Label5	True	Faculty Name	0.2	
TextBox	txtName	True		0.3	
GroupBox	GroupBox3	True	Faculty information	1	
Label	Label6	True	Title	1.0	
TextBox	TxtTitle	True		1.1	
Label	Label7	True	Office	1.2	
TextBox	txtOffice	True		1.3	
Label	Label8	True	Phone	1.4	
TextBox	txtPhone	True		1.5	
Label	Label9	True	College	1.6	
TextBox	txtCollege	True		1.7	
Label	Label10	True	Email	1.8	
TextBox	txtEmail	True		1.9	
Button	cmdInsert	True	Insert	6	
Button	cmdSelect	True	Select	7	
Button	cmdCancel	True	Cancel	8	
Button	cmdBack	True	Back	9	
PictureBox	PhotoBox	True			
Form	InsertFacultyForm	True	CSE_DEPT Insert Faculty Form		

In addition to the form's properties shown in Table 5.2, the following properties of the form should be set up:

- AcceptButton: cmdInsert (set Insert button as default button)
- StartPosition: CenterScreen (set the form in the center)

The check box chkPhoto is used to allow users to choose the faculty photo for insertion, including the name of the faculty photo and the location of that photo, if they like. Selecting this check box indicates that a faculty photo will be included and displayed if this data insertion operation is successful. Also, the user needs to provide the photo's name (txtPhotoName) and the location (txtPhotoLocation) in which the photo is stored if this check box is selected, and that information will be used later to load and display the photo when the newly inserted faculty information is validated by clicking the Select button. One point to note is that the faculty photo or image file should be stored in the location indicated by the txtPhotoLocation box before this project is run.

The finished Insert Faculty form window is shown in Figure 5.1.

From both Table 5.2 and Figure 5.1, it can be seen that the text box Photo Name and the text box Photo Location are disabled (Enabled properties are False in Table 5.2 and the boxes are gray in Figure 5.1). The reason for this is that we want to use the Faculty Photo check box to control these two controls to make the project more professional. In other words, these two controls are disabled until the Faculty Photo check box is checked, and in that situation, it allows the user to enter the photo's name and location into these two controls. If the check box is unchecked, which means that the user does not want to include a faculty photo in this data insertion operation, no faculty photo will be involved in this data insertion action.

The Method combo box allows users to select a method to insert data into the database – either the TableAdapter.Insert() method or the TableAdapter.Update() method.

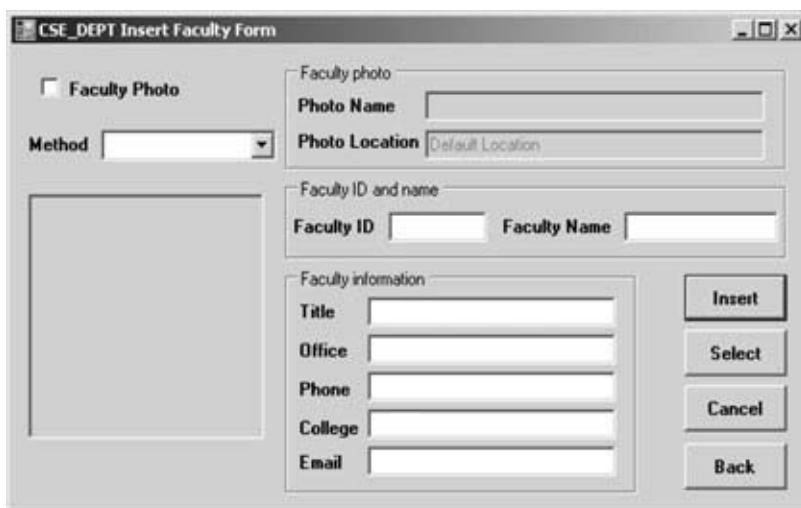


Figure 5.1. The finished Insert Faculty form window.

The order of the TabIndex values we created in Table 5.2 selects the Faculty Name text box as the default one and set a focus to it to allows users to directly enter the faculty name without needing to click that box first.

### 5.2.4 Validate Data Before the Data Insertion

It is important to validate data before it can be inserted into the database since we want to make sure that data inserted into the database is correct. The most popular validation mode is to make sure that each data value is not NULL and it contains a specific value. Of course, one can insert 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 it 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 text box, is not an empty string unless the user intends to leave it empty. In that case, NULL must be entered. To make this validation simple, we develop a control collection and add all those text boxes into this collection. In this way, we don't need to check each text box, but instead we can use the For Each... Next loop to scan the whole collection to find empty text boxes.

#### 5.2.4.1 Visual Basic Collection and .NET Framework Collection Classes

There are two kinds of collection classes available for Visual Basic.NET applications: one is the 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 declaration:

---

```
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 uses only the class name with the default namespace. Both declarations work well for 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 develop the coding for the data validation using this collection component for our application.

#### 5.2.4.2 Validate Data Using the Generic Collection

First, we need to create the generic collection object for our Insert faculty form. Since this collection will be used by the different procedures in this form, a form-level or a model-level object should be created. Open the code window of the Insert Faculty form by clicking the View Code button in the Solution Explorer window, and enter the code that is shown in Figure 5.2 into the form's declaration section.

The 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. This variable will be used to temporarily store the faculty name entered by the user in the txtName text box, 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 item content. The code in which the default namespace is utilized also works fine. Here we comment it out to illustrate that we prefer to use the full class name to create this collection object.

In order to use the collection object to check all text boxes, one needs to add all text boxes into the collection object after the collection object, FacultyCollection, is created by using the Add() method. One point we need to emphasize is the order in which to perform this validation check. As the project starts, all text boxes are blank. The user needs to enter all faculty information into the appropriate

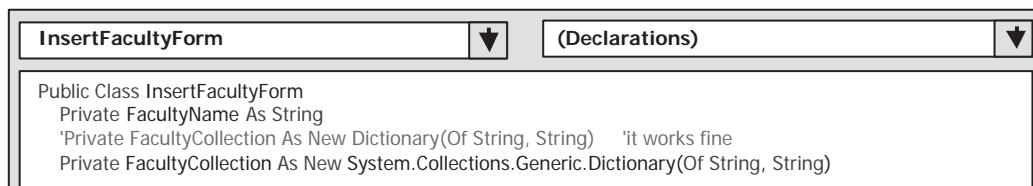
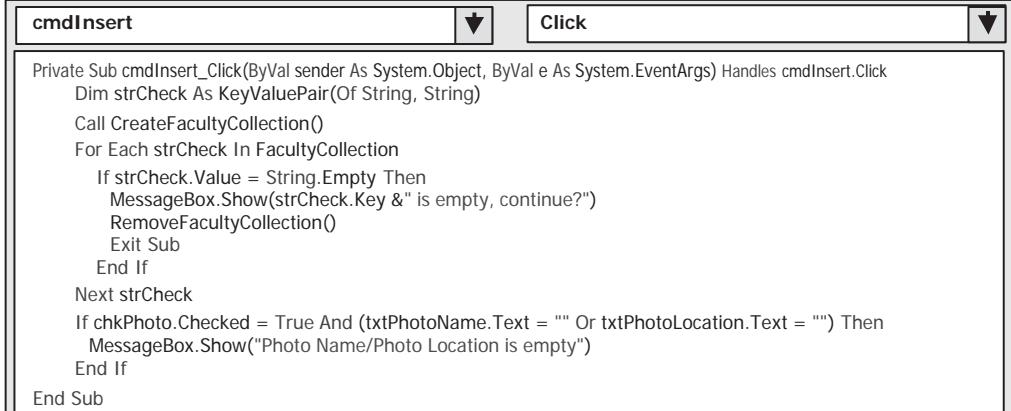


Figure 5.2. The form-level collection object.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdInsert" and the tab says "Click". The code is as follows:

```

Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim strCheck As KeyValuePair(Of String, String)
    Call CreateFacultyCollection()
    For Each strCheck In FacultyCollection
        If strCheck.Value = String.Empty Then
            MessageBox.Show(strCheck.Key & " is empty, continue?")
            RemoveFacultyCollection()
            Exit Sub
        End If
    Next strCheck
    If chkPhoto.Checked = True And (txtPhotoName.Text = "" Or txtPhotoLocation.Text = "") Then
        MessageBox.Show("Photo Name/Photo Location is empty")
    End If
End Sub

```

**Figure 5.3.** The coding for the Insert button event procedure.

text box. Then the user clicks the Insert button to perform this data insertion. The time to add all text boxes into the collection object is after the user has finished entering all information into the text boxes, not before. Also, each time you finish data validation by checking all text boxes, all the text boxes should be removed from that collection since the collection only allows those text boxes to be added once.

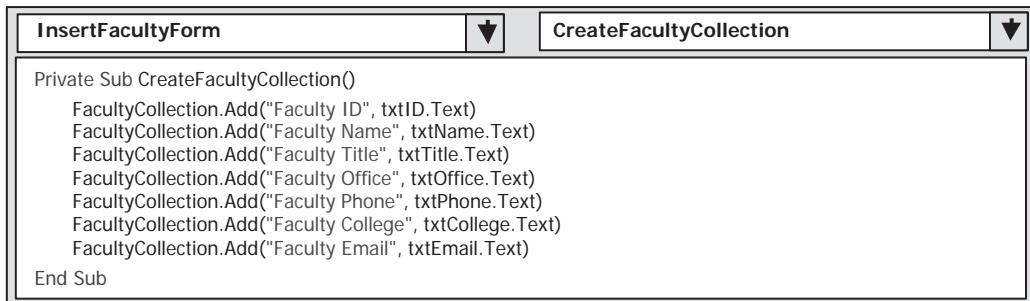
Another point to be noted is that in order to simplify this data validation, in this application we need all text boxes to be filled with specific information, or NULL needs to be entered if no information will be entered. In other words, we don't allow any text boxes to be empty. The data insertion will not be performed until all text boxes are nonempty in this application.

Based on these descriptions, we need to create two user-defined subroutines to perform the adding and removing of text boxes from the collection object.

Open the GUI window of the Insert Faculty form by clicking the View Designer button in the Solution Explorer window, and then double-click the Insert button to open its event procedure. Enter the code that is shown in Figure 5.3 into this event procedure.

Let's take a look at this coding 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 step **B**, both a key and the content of the associated text box are added to the collection FacultyCollection when the subroutine CreateFacultyCollection() is called. We need both the key and the value of a text box to validate the data for each text box; to check whether a text box is empty, the value of that text box is used, and to identify the emptied text box, the key of the text box is used.
- B. Next, we need to call the subroutine CreateFacultyCollection() to add all text boxes into the collection FacultyCollection. Refer to Figure 5.4 for the coding of this subroutine.
- C. The For Each...Next loop is utilized to scan all text boxes to check and identify any text box whose content is empty in the FacultyCollection. If



The screenshot shows a Microsoft Visual Studio code editor window. The title bar has two tabs: "InsertFacultyForm" on the left and "CreateFacultyCollection" on the right. Both tabs have a downward arrow icon to their right. The main code area contains the following VB.NET code:

```

Private Sub CreateFacultyCollection()
    FacultyCollection.Add("Faculty ID", txtID.Text)
    FacultyCollection.Add("Faculty Name", txtName.Text)
    FacultyCollection.Add("Faculty Title", txtTitle.Text)
    FacultyCollection.Add("Faculty Office", txtOffice.Text)
    FacultyCollection.Add("Faculty Phone", txtPhone.Text)
    FacultyCollection.Add("Faculty College", txtCollege.Text)
    FacultyCollection.Add("Faculty Email", txtEmail.Text)
End Sub

```

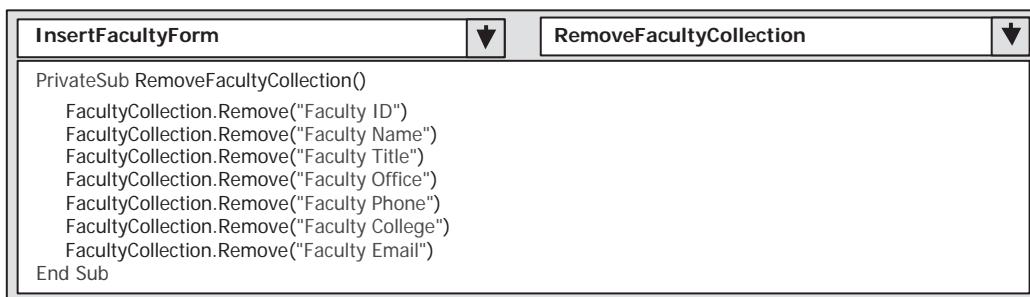
Figure 5.4. The coding for the subroutine CreateFacultyCollection.

any text box is empty, that text box will be identified by its key, and a message box is used to ask the user to fill some information in it. Then the subroutine RemoveFacultyCollection() is called to remove all text boxes that have been added to the collection in the subroutine CreateFacultyCollection() since the collection only allows those text boxes to be added once. The program will exit if this situation happens. The coding for the subroutine RemoveFacultyCollection() is shown in Figure 5.5.

- D. If the Checked property of the chkPhoto is True, which means that the user selected the Faculty Photo check box and wants to include a faculty photo for this data insertion, we need to check whether the text box containing the photo name or the text box containing the photo location is empty. A message box will be shown to remind the user to enter the photo name and the photo location if either of them is empty.

Now let's take care of the coding for the subroutine CreateFacultyCollection(), which is shown in Figure 5.4.

The coding is very simple and straightforward. Each text box is added to 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 text box, which is considered the Value parameter. In this way, each text box can be identified by its key. Of course each text box 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.



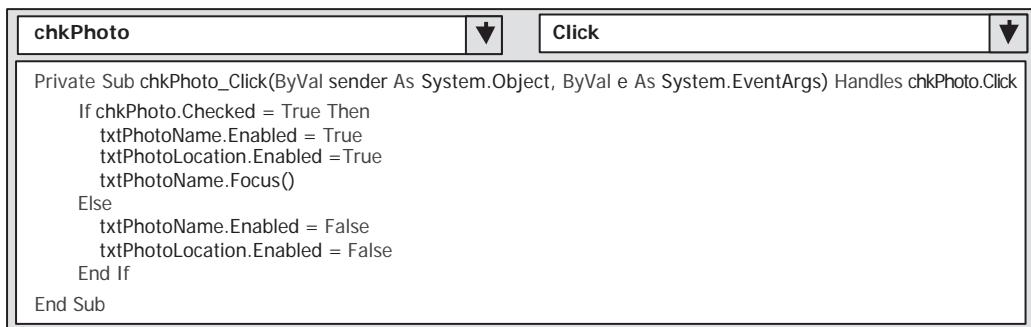
The screenshot shows a Microsoft Visual Studio code editor window. The title bar has two tabs: "InsertFacultyForm" on the left and "RemoveFacultyCollection" on the right. Both tabs have a downward arrow icon to their right. The main code area contains the following VB.NET code:

```

PrivateSub RemoveFacultyCollection()
    FacultyCollection.Remove("Faculty ID")
    FacultyCollection.Remove("Faculty Name")
    FacultyCollection.Remove("Faculty Title")
    FacultyCollection.Remove("Faculty Office")
    FacultyCollection.Remove("Faculty Phone")
    FacultyCollection.Remove("Faculty College")
    FacultyCollection.Remove("Faculty Email")
End Sub

```

Figure 5.5. The coding for the subroutine RemoveFacultyCollection.



The screenshot shows the Microsoft Visual Studio code editor. The title bar says "chkPhoto" and the dropdown menu says "Click". The code in the editor is:

```
Private Sub chkPhoto_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles chkPhoto.Click
    If chkPhoto.Checked = True Then
        txtPhotoName.Enabled = True
        txtPhotoLocation.Enabled = True
        txtPhotoName.Focus()
    Else
        txtPhotoName.Enabled = False
        txtPhotoLocation.Enabled = False
    End If
End Sub
```

**Figure 5.6.** The coding for the chkPhoto\_Click event procedure.

To remove all text boxes from the collection, the subroutine RemoveFacultyCollection() should be called, and the coding for this subroutine is shown in Figure 5.5. The Key parameter of each text box is used as the identifier for each text box, and the Remove() method is called to remove all text boxes from the collection object.

The next coding job is for the check box chkPhoto. Recall that when the project begins to run, the Photo Name and Photo Location text boxes are disabled. If the user selects this check box, it means that the user wants to include a faculty photo in the data insertion. Both the Photo Name and the Photo Location boxes should be enabled and focused to allow the user to enter the associated information about the faculty photo. Open the code window by clicking the View Code button in the Solution Explorer window. Select the item chkPhoto from the Class Name combo box, and then select the item Click from the Method Name combo box to open the chkPhoto\_Click() event procedure. Enter the code shown in Figure 5.6 into this event procedure.

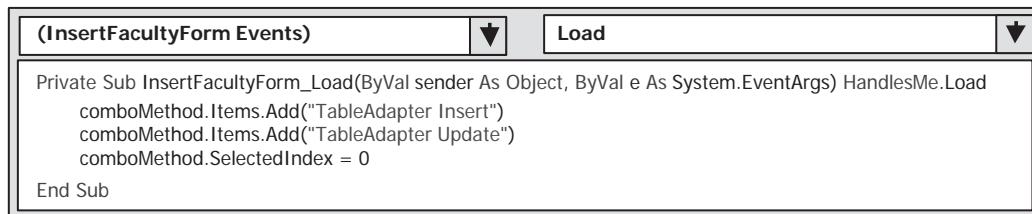
First, we need to check whether the chkPhoto check box has been selected, because in this case, there are two possibilities: chkPhoto is selected or cleared. If it is selected, both the txtPhotoName and txtPhotoLocation text boxes should be enabled and a focus set for the Photo Name text box to make it convenient for the user to directly enter the photo name into that text box without clicking it first. Otherwise, both text boxes should be disabled since chkPhoto is not selected.

At this point, we have completed the coding for the data validation. Next, we need to handle some initialization and termination coding jobs for the data insertion.

### 5.2.5 Initialization and Termination Coding for the Data Insertion

In this section we need to handle the coding for the following event procedures:

- Coding for the Form\_Load event procedure to initialize the comboMethod combo box to display two data insertion methods: TableAdapter.Insert() and TableAdapter.Update()
- Coding for the Cancel button event procedure to erase all content from the text boxes that contained the faculty information
- Coding for the Back button event procedure to return the program to the Selection form window to allow users to select other data operations



The screenshot shows the Visual Studio code editor with the title bar '(InsertFacultyForm Events)'. Below it, there are two dropdown menus: one for 'Class Name' and one for 'Method Name', both currently set to 'Load'. The main code area contains the following VB.NET code:

```
Private Sub InsertFacultyForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    comboMethod.Items.Add("TableAdapter Insert")
    comboMethod.Items.Add("TableAdapter Update")
    comboMethod.SelectedIndex = 0
End Sub
```

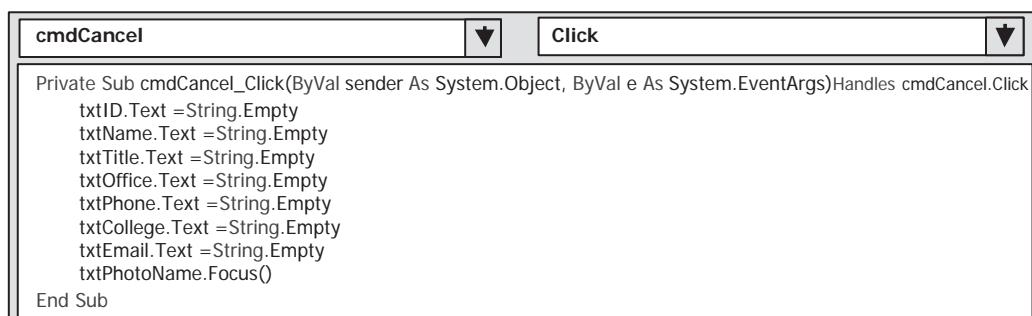
**Figure 5.7.** The coding for the Form\_Load event procedure.

The first coding is for the `comboMethod` combo box. As the project runs and the Insert Faculty form window is shown, two different methods should be displayed in this box to allow users to select one to perform the data insertion, either the `TableAdapter.Insert()` method or the `TableAdapter.Update()` method. This display should be made as the form window is first loaded and displayed, so we need to put the coding for this combo box into the `Form_Load` event procedure. Open the code window for the Insert Faculty form by clicking the View Code button in the Solution Explorer window, then select the `insertfacultyform Events` item from the Class Name combo box, and select the Load item from the Method Name combo box to open this event procedure. Enter the code shown in Figure 5.7 into this event procedure.

The coding is straightforward and easy to understand. Two methods are added to the combo box by using the `Add()` method, and the first method is selected as the default one by setting the `SelectedIndex` property to 0.

The coding for other command buttons, such as Cancel and Back, is simple. The function of the Back button is to return to the Selection form window, so its coding is only one line: `Me.Close()`. The function of the Cancel button is to clean up all the text boxes' contents except Photo Name and Photo Location; we want to keep their contents to be used later. The Default Location in the Photo Location text box means that the photo is located in the folder in which the Visual Basic.NET executable file is located. The coding for the Cancel button is shown in Figure 5.8.

To clean up the contents of a text box, either an empty string or the class variable `String.Empty` can be used, and both need to be assigned to the `Text` property of the text box to clean it up. A focus is set to the Photo Name text box to make it convenient for the user.



The screenshot shows the Visual Studio code editor with the title bar 'cmdCancel'. Below it, there are two dropdown menus: one for 'Class Name' and one for 'Method Name', both currently set to 'Click'. The main code area contains the following VB.NET code:

```
Private Sub cmdCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdCancel.Click
    txtID.Text = String.Empty
    txtName.Text = String.Empty
    txtTitle.Text = String.Empty
    txtOffice.Text = String.Empty
    txtPhone.Text = String.Empty
    txtCollege.Text = String.Empty
    txtEmail.Text = String.Empty
    txtPhotoName.Focus()
End Sub
```

**Figure 5.8.** The coding for the Cancel button event procedure.

Now we need to take care of the coding for the data insertion. Because we are using design tools to perform this job, first we need to configure the TableAdapter and build the insert query using the TableAdapter Query Configuration Wizard.

## 5.2.6 Build the Insert Query

As we mentioned, two methods will be discussed in this part: the TableAdapter DBDirect method TableAdapter.Insert() and the TableAdapter.Update() method. First, let's concentrate on the first method.

### 5.2.6.1 Configure the TableAdapter and Build the Data Insertion Query

In order to use the TableAdapter.Insert() DBDirect method to access the database, we first need to configure the TableAdapter and build the Insert query.

Open the Data Sources window by going to the Data>Show Data Sources menu item. In the opened window, click the Edit the DataSet with Designer button that is second from the left on the toolbar in the Data Sources window to open this Designer. Then right-click the bottom item from the Faculty table, and select the Add Query item from the popup menu to open the TableAdapter Query Configuration Wizard. Keep the default selection “Use SQL statements” unchanged, and click Next to go to the next window. Select and check the INSERT item from this window since we need to perform the query to insert new records, and then click Next again to continue. Click the Query Builder button since we want to build our insert query. The opened Query Builder window is shown in Figure 5.9.

The default Insert query statement is matched to our requirement since we want to add a new faculty record that contains the new information about that inserted faculty member, including the faculty\_id, name, office, phone, college, title, and email. Click the OK button to go to the next window. Click Next to confirm this query and continue to the next step. Modify the query function name from the

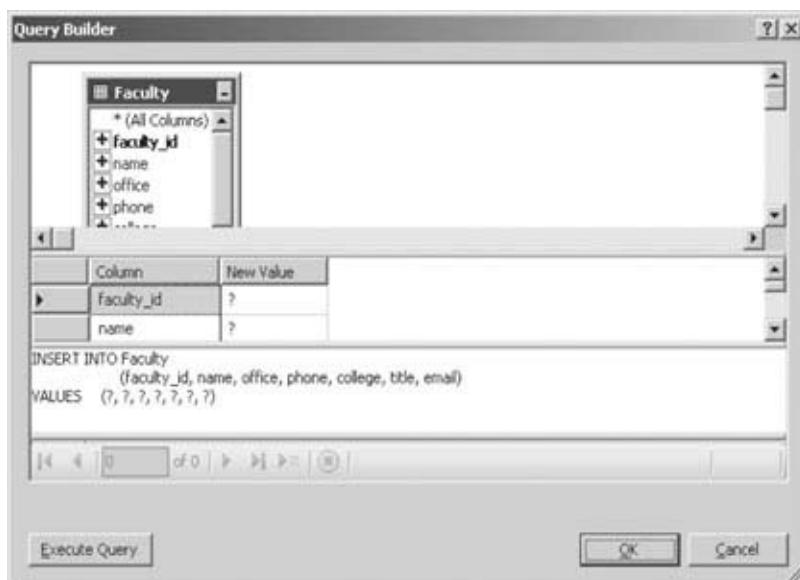


Figure 5.9. The opened Query Builder window.

default one to InsertFaculty, and click Next to go to the last window. Click the Finish button to complete this query building and close the wizard. Immediately you will find that a new query function has been added to the Faculty TableAdapter as the last item.

Now that we have finished the configuration of the TableAdapter and the building of the insert query, it is time to develop the code to run the TableAdapter to complete this data insertion query. First, we need to develop the code for the first method – using the TableAdapter DBDirect method, TableAdapter.Insert().

### 5.2.7 Develop Code to Insert Data Using the TableAdapter.Insert Method

Open the GUI of the Insert Faculty form by clicking the View Designer button from the Solution Explorer window, double-click the Insert button to open its event procedure, and add the code shown in Figure 5.10 into this event procedure.

Recall that we created some code for this event procedure in Section 5.2.4.2 to perform the data validation, so the old code is indicated with a gray background.

Let's look at this piece of coding to see how it works.

- First, we need to create a TableAdapter object for the FacultyTableAdapter class since we need this object to insert data directly into the database. As

```

cmdInsert_Click
Click

Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim strCheck As KeyValuePair(Of String, String)
    Dim intInsert As Integer
    Call CreateFacultyCollection()
    For Each strCheck In FacultyCollection
        If strCheck.Value = String.Empty Then
            MessageBox.Show(strCheck.Key & " is empty, continue?")
            RemoveFacultyCollection()
            Exit Sub
        End If
    Next strCheck
    If chkPhoto.Checked = True And (txtPhotoName.Text = "" Or txtPhotoLocation.Text = "") Then
        MessageBox.Show("Photo Name/Photo Location is empty")
    End If
    FacultyName = txtName.Text      'reserve the faculty name for validation
    If comboMethod.Text = "TableAdapter Insert" Then
        intInsert = FacultyTableApt.InsertFaculty(txtID.Text, txtName.Text, txtOffice.Text, _
                                                    txtPhone.Text, txtCollege.Text, txtTitle.Text, txtEmail.Text)
    Else
        'coding to use the TableAdapter.Update() method later....
    End If
    If intInsert = 0 Then
        MessageBox.Show("The data insertion is failed")
        cmdInsert.Enabled = True      'insert is failed, enable the Insert button
        Exit Sub
    End If
    cmdCancel.PerformClick()      'clean up all faculty information
    txtName.Text = FacultyName   'recover the faculty name
    cmdInsert.Enabled = False    'disable the Insert button, the same data can only be inserted by 1 time
End Sub

```

Figure 5.10. The modified coding for the Insert button event procedure.

we know, all TableAdapters in this application are located at the namespace CSE\_DEPTDataSetTableAdapters, so this namespace must be pre-fixed before the desired TableAdapter class.

- B. A local integer variable intInsert is declared here, and it is used to hold the returned value from the execution of the TableAdapter.Insert() method. The value of this returned integer, which indicates how many records have been successfully inserted or updated in the database, can be used to determine whether this data insertion is successful or not. A returned value of 0 means that no record is added or updated in 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 in the text box txtName since we need this name to validate the 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. Seven pieces of new information about the newly inserted faculty member entered by the user into seven text boxes, will be inserted into the Faculty table in the database.
- E. If the user selected the TableAdapter.Update() method to perform this data insertion, the data insertion will be performed by calling that method. The coding for this method will be discussed in the next section.
- F. If the data insertion is successful, the returned integer will reflect the number of records that have been inserted into the database correctly. As mentioned in **B**, a returned value of 0 indicates that the insertion has 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 data insertion, the same data can only be inserted into the database once. To avoid multiple insertions, in this application (in fact, in most popular applications), we will disable the Insert button after one record is inserted successfully (refer to step **I** below). If the insertion has failed, we need to recover or re-enable the Insert button to allow the user to try another insertion later.



Most databases, including Access, SQL Server, and Oracle, do not allow multiple data insertions of the same data item into the databases. Each data item or record can only be added or inserted into the database once. In other words, no duplicating record can be added or can exist in the database. Each record in the database must be unique. A popular way to avoid this situation is to disable the Insert button after one insertion is done.

- G. If the data insertion is successful, next we need to confirm this insertion or validate it by retrieving the newly inserted record from the database, and the returned data should be displayed in seven text boxes to allow us to check it. In order to do that, we first need to clean up the contents of the seven text

boxes. An easy way to do that is to call the Cancel button event procedure since we have already developed the code to do this cleaning job in that event procedure. There are different ways to call this event procedure, but here we use the easiest way, which is to execute the PerformClick() method of the Cancel button object. This method is equivalent to clicking the Cancel button to trigger and run its event procedure to finish this cleaning job.

- H. As we mentioned before, when we perform the data validation to validate this data insertion, a 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 recover this faculty name after the contents of all text boxes are cleaned up.
- I. As mentioned in F, the database does not allow 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 prevent it from being clicked again.

From the above explanations, we know that a good way to avoid multiple insertions of the same data item into the database is by disabling the Insert button after the insertion is successfully completed. A question arises: When and how can this button be enabled again to allow us to insert some other different records if we want to do that later? The solution to this question is to develop another event procedure to handle this issue. Think about it: when we want to insert a new data item into the database, first we must enter each new piece of information into each associated text box, such as txtID, txtName, txtOffice, txtPhone, txtTitle, txtCollege, and txtEmail. In other words, as long as the content of a text box is changed, which means that a new record will be inserted, we should enable the Insert button at that moment to allow users to perform this new insertion. Visual Basic.NET does provide an event called TextChanged and an associated event procedure for the text box control. So we need to use this event procedure to enable the Insert button as long as a TextChanged event has occurred. Another question arises: With the occurrence of which text box's TextChanged event should we trigger the associated event procedure to enable the Insert button to allow users to insert a new record? Any text box's TextChanged event? To answer this question, we need to review the data issue in the database. As you know, our sample database CSE\_DEPT (i.e., in our Faculty data table) identifies a record based on its primary key. In other words, only those records with different primary keys can be considered different records. So the to our question is that answer only a new record should be allowed when the content of the text box that stores the primary key, in our case the faculty\_id, is changed. This means that a new record will be inserted, and as this happens, that text box's TextChanged event procedure should be triggered to enable the Insert button.

To open the TextChanged event procedure for the text box txtID, open the GUI of the Insert Faculty form window by clicking the View Designer button in the Solution Explorer window, and then double-click the txtID text box to open its TextChanged event procedure. Change the event procedure's name from txtID\_TextChanged to FacultyInfoChanged, and enter the code shown in Figure 5.11 into this event procedure.

The underscore (“\_”) is used to break a single line of coding into multiple lines in Visual Basic.NET if that line is too long. Note that you must leave a space before the

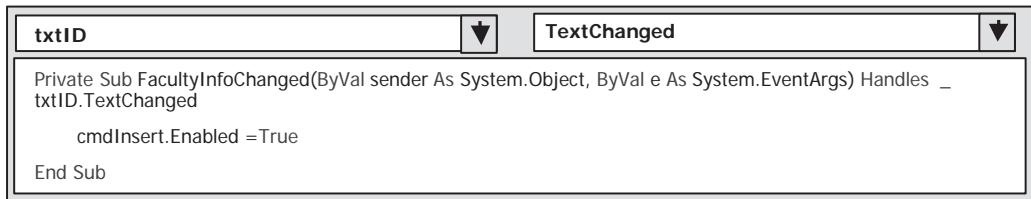


Figure 5.11. The coding for the TextChanged event procedure.

underscore and press the Enter key on your keyboard after the underscore to start the next line. The coding for this event procedure is simple: enable the Insert button by setting the Enabled property of that button to True as the txtID TextChanged event occurs.

Now that we have finished the coding for the first data insertion method, let's do the coding for the second method.

### 5.2.8 Develop Code to Insert Data Using the TableAdapter.Update Method

When a data-driven application uses a DataSet to store data, as we did for this application by using CSE\_DEPTDataSet, one can use the TableAdapter.Update() method to insert a new record into the database.

To insert a new record into the database using this method, there are two steps:

- Add new records to the desired data table in the DataSet, for example, in this application, the Faculty table in the DataSet CSE\_DEPTDataSet.
- Call the TableAdapter.Update() method to update the newly 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 a different argument in the Update() method: an entire DataSet, a DataTable, an array of DataRow, or a single DataRow.

Now let's develop our code based on the above two steps to insert data using this method.

Reopen the GUI of the Insert Faculty form window, and double-click the Insert button to open its event procedure. We have already developed most of the code for this procedure in the last section, and now we need to add the code to perform the second data insertion method. Browse to the Else block (step E in Figure 5.10), and enter the code shown in Figure 5.12 into this block.

In order to distinguish between the new code and the old code that has been added before, all the old code is indicated with a gray background.

Let's take a close look at this piece of newly inserted code to see how it works.

- A. First, we need to declare a new object of the DataRow class. Each DataRow object can be mapped to a real row in a data table. Since we are using the DataSet to manage all data tables in this project, the DataSet must be pre-fixed to the DataRow object. Also, we need to create a row in the Faculty data table – the FacultyRow is selected as the DataRow class.

```

cmdInsert_Click
    Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
        Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
        Dim newFacultyRow As CSE_DEPTDataSet.FacultyRow
        Dim strCheck As KeyValuePair(Of String, String)
        Dim intInsert As Integer
        Call CreateFacultyCollection()
        For Each strCheck In FacultyCollection
            If strCheck.Value = String.Empty Then
                MessageBox.Show(strCheck.Key & " is empty, continue?")
                RemoveFacultyCollection()
                Exit Sub
            End If
        Next strCheck
        If chkPhoto.Checked = True And (txtPhotoName.Text = "" Or txtPhotoLocation.Text = "") Then
            MessageBox.Show("Photo Name/Photo Location is empty")
        End If
        FacultyName = txtName.Text      'reserve the faculty name for validation
        If comboMethod.Text = "TableAdapter Insert" Then
            intInsert = FacultyTableApt.InsertFaculty(txtID.Text, txtName.Text, txtOffice.Text,
                txtPhone.Text, txtCollege.Text, txtTitle.Text, txtEmail.Text)
        Else
            newFacultyRow = Me.CSE_DEPTDataSet.Faculty.NewFacultyRow()
            newFacultyRow = InsertFacultyRow(newFacultyRow)
            CSE_DEPTDataSet.Faculty.Rows.Add(newFacultyRow)
            intInsert = FacultyTableApt.Update(CSE_DEPTDataSet.Faculty)
        End If
        If intInsert = 0 Then
            MessageBox.Show("The data insertion is failed")
            cmdInsert.Enabled = True      'insert is failed, enable the Insert button
            Exit Sub
        End If
        cmdCancel.PerformClick()      'clean up all faculty information
        txtName.Text = FacultyName   'recover the faculty name
        cmdInsert.Enabled = False    'disable the Insert button, the same data can only be inserted by 1 time
    End Sub

```

**Figure 5.12.** The coding 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 newly inserted faculty information, which is stored in seven text boxes, into this newly created `DataRow` object. The functionality of the coding for this user-defined function is explained below. The function returns a completed `DataRow` in which all information about the new record has been added.
- D. The completed `DataRow` is added to 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 to the real database. As we mentioned before, you can control the amount of data to be added to the database by passing different arguments. Here we only want to add one new record to the `Faculty` table, so a data table is passed as the argument. This `Update()` method is supposed to return an

```

A Private Function InsertFacultyRow(ByRef facultyRow As CSE_DEPTDataSet.FacultyRow) As _
CSE_DEPTDataSet.FacultyRow
B     facultyRow.faculty_id = txtID.Text
    facultyRow.name = txtName.Text
    facultyRow.office = txtOffice.Text
    facultyRow.phone = txtPhone.Text
    facultyRow.college = txtCollege.Text
    facultyRow.title = txtTitle.Text
    facultyRow.email = txtEmail.Text
C     Return facultyRow
End Function

```

**Figure 5.13.** The coding for the user-defined function InsertFacultyRow.

integer value to indicate whether this update is successful or not. The value of this returned integer is equal to the number of rows that have been successfully added to the database. A returned value of 0 means that this update has failed since no new row has been added to the database.

Now let's develop the coding for the user-defined function InsertFacultyRow(). Open the code window and enter the code shown in Figure 5.13 into this function. Let's see how this piece of code works.

- A. In Visual Basic.NET, unlike C or 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 the passing-by-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.
- B. Seven pieces of new information stored in the associated text boxes are added to this new DataRow object, that is, added to 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 passing-by-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 for our data insertion by using two methods. Now let's test our coding 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, then the Selection form, and then the Faculty form. Then you can click the Insert button in the Faculty form to open the Insert Faculty form window to test the data insertion. The second way, which is more flexible, is to start directly from the Insert Faculty form. To run the project

```

cmdInsert Click
Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim insertFaculty As New InsertFacultyForm
    insertFaculty.Show()
End Sub

```

Figure 5.14. The coding for the Insert button event procedure in the Faculty form.

in the first way, you need to add some code into the Insert button event procedure in the Faculty form to direct the program to open the Insert Faculty form window and close the Faculty form window. To do that, go to the Faculty form window, and open the Insert button event procedure by double-clicking the Insert button. Enter the code shown in Figure 5.14 into this event procedure.

First, an Insert Faculty form object is created, and then the Show() method of the Insert Faculty form object is called to open that form window.

To run the project in the second way, one must confirm that the Startup form in the Project Properties window is the Insert Faculty form window. To do that, go to the Project|InsertWizard Project Properties menu item to open the Project Properties window. Keep the default tab Application selected, and make sure that the content of the Startup form box is the InsertFacultyForm. 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 either way. We prefer to run it the first way. Make sure that the Startup form is the LogIn form, and then click the Start Debugging button to run the project. Enter the correct username and password into the LogIn form, and select Faculty Information from the Selection form window. Click the Insert button from the opened Faculty form window to open the Insert Faculty form window, which is shown in Figure 5.15.

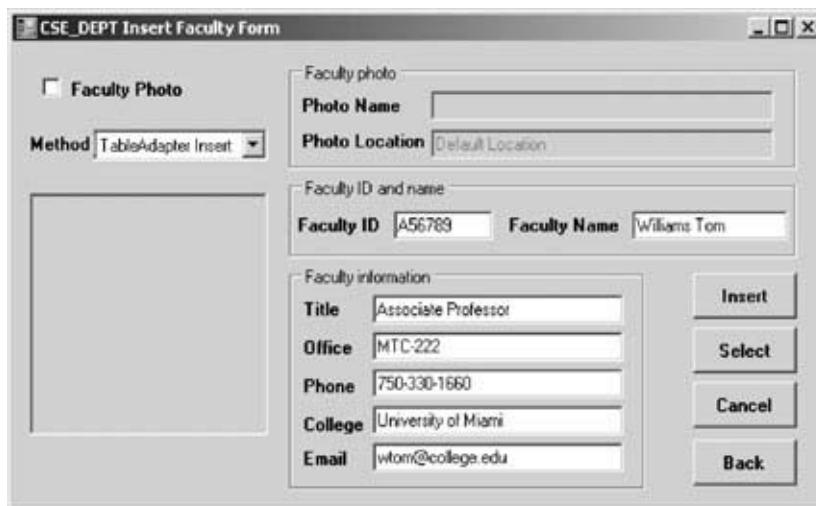


Figure 5.15. The running status of the Insert Faculty form window.

First, we want to test the first method, TableAdapter.Insert(), to add a new faculty record into the database, so keep the default value in the combo box unchanged. Enter the new information for this faculty member into the associated text box, such as txtID, txtName, txtOffice, txtPhone, txtCollege, txtTitle, and txtEmail, respectively. After all the new information has been entered into all associated text boxes, click the Insert button to execute this data insertion using the first method. If this insertion is successful, no warning message will be displayed; the contents of all text boxes, except the faculty name text box, will be cleaned up; and the Insert button will be disabled to avoid the same data being added to the database more than once.

Click the Back button to terminate the project. To confirm this data insertion, now we can open the database to check whether this new record has been added to the Faculty table. To do that, go to the Debug folder of the current project – in our case, it is **C:\Book5\Chapter 5\InsertWizard Solution\InsertWizard Project\bin\Debug** – to open the embedded database CSE\_DEPT.mdb by double-clicking it. Then open the Faculty table by double-clicking it, too. The opened Faculty table is shown in Figure 5.16.

It can be found from this table that the second row with faculty\_id as A56789 is the new record we just added to this Faculty table. Recall that there were a total of eight faculty members in this Faculty table when we designed this database in Chapter 2. Count how many records there are in this table right now. There are nine, so our data insertion is successful!

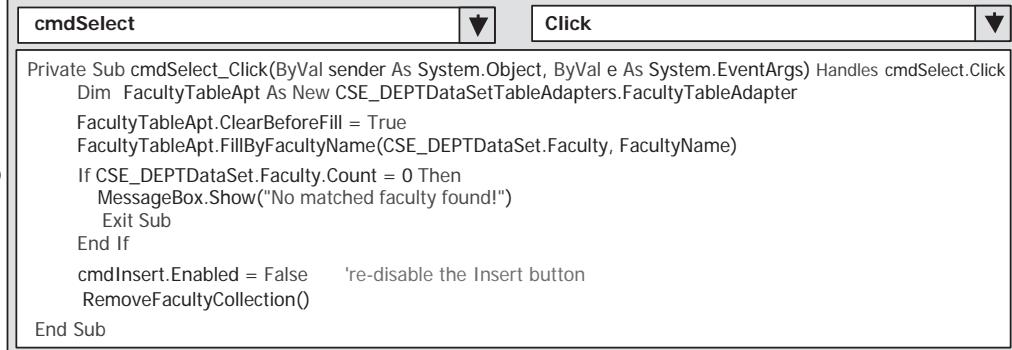
Next, we want to develop a more professional method to test our data insertion operation. This method is to retrieve the inserted records from the database and display them in the form window to confirm our data insertion. We need to use the Select button in the Insert Faculty form window and its event procedure to complete this method.

### 5.2.9 Validate Data After the Data Insertion

To use the Select button and its event procedure to confirm our data insertion, first you need to open the Insert Faculty form window and then double-click the Select button to open its event procedure. The functionality of this event procedure is to

	faculty_id	name	office	phone	college	title	email
•	A52990	Black Anders	MTC-218	750-378-9967	Virginia Tech	Professor	banderson@college.edu
•	A56789	Williams Tom	MTC-222	750-330-1660	University of Miami	Associate Professor	wform@college.edu
•	A77587	Debby Angle	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@college.edu
•	B86750	Alice Brown	MTC-257	750-330-6660	University of Florida	Assistant Professor	abrown@college.edu
•	B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Assistant Professor	yba@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 Profes	sjohnson@college.edu
•	K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@college.edu

Figure 5.16. The opened Faculty table.



```

cmdSelect_Click
    Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
        Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
        FacultyTableApt.ClearBeforeFill = True
        FacultyTableApt.FillByFacultyName(CSE_DEPTDataSet.Faculty, FacultyName)
        If CSE_DEPTDataSet.Faculty.Count = 0 Then
            MessageBox.Show("No matched faculty found!")
            Exit Sub
        End If
        cmdInsert.Enabled = False      're-disable the Insert button
        RemoveFacultyCollection()
    End Sub

```

**Figure 5.17.** The coding for the Select button event procedure.

retrieve the newly inserted records from the database and display them in the associated text boxes in this form, such as txtID, txtName, txtTitle, txtOffice, txtPhone, txtCollege, and txtEmail. To do that, we need to use the query function FillByFacultyName() we built in Section 4.11 in Chapter 4. Refer to that section and Figure 4.49 for detailed information on building this query function. Besides building that query function, you also need to perform data binding to make the connections between the text box controls in the Insert Faculty form window and the data columns in the DataSet. Detailed data binding will be discussed in Section 5.2.9.2 below.

Now let's begin to do our coding for this event procedure.

### 5.2.9.1 Develop Code to Retrieve the Newly Inserted Records

Enter the code shown in Figure 5.17 into the opened Select button event procedure.

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

- Since the query function, which we built in Section 4.11 and which contains a SELECT statement, is based on FacultyTableAdapter, first we need to create an object of that TableAdapter in order to use that query function.
- As we did before, we need to clean up all old contents from the Faculty data table in the DataSet before we can call the query function to fill that table. This will make our data query neat.
- Then we can call this query function to fill the Faculty table in the DataSet with the data retrieved from the Faculty table in the database. Remember that this function contains two arguments. The first one is the Faculty data table in the DataSet, and the second is a dynamic parameter that is used for this query and is the faculty name located in the WHERE clause. Since we have already stored the faculty name in our form-level variable FacultyName in the Insert button's event procedure, we can use it directly as the second argument for this query function.
- By checking the Count property of the Faculty table, we can determine whether this data insertion is successful or not. This property contains the number of records that have been retrieved from the database correctly. A zero means that no record has been found and retrieved from the database and this data insertion has failed. In this case, a message box will be

displayed with a warning message to indicate this situation, and the program will be exited.

- E. If this query function is successful, it means that our data insertion is fine and the newly inserted data has been retrieved from the database and displayed in the associated text boxes. In Section 5.2.7, however, recall that in order to avoid multiple insertions of the same data, the Insert button is disabled (refer to step **I** in Figure 5.10) and all text boxes storing the faculty information, except the faculty name text box, are cleaned up (refer to steps **G** and **H** in Figure 5.10) as soon as the user clicks the Insert button to perform the data insertion operation. Also refer to coding shown in Figure 5.11. The Insert button will be enabled as long as the content of the faculty ID text box is changed, which means that the user wants to perform another new data insertion operation. There is a trick here; the successful query function will bring the newly inserted record back and display it in all faculty-related text boxes that are blank now. This displaying is equivalent to triggering the TextChanged event for the faculty ID text box, and therefore the Insert button will be enabled. That is a wrong enabling operation since no new record will be inserted into the database, and this TextChanged event is only triggered by the displayed data that is newly inserted and just retrieved from the database. To solve this wrong enabling operation, we need to redisable the Insert button.
- F. After this data validation is successfully completed, all text boxes storing the faculty information should be removed from the collection object since no duplicated text boxes can be added to the same collection more than once.

Before we can run the project to test this data validation, we need to first perform data binding to connect each faculty information-related text box to each data column in the DataSet. As this data mapping or data binding is set up, the retrieved data can be displayed in each associated text box when this data validation is executed.

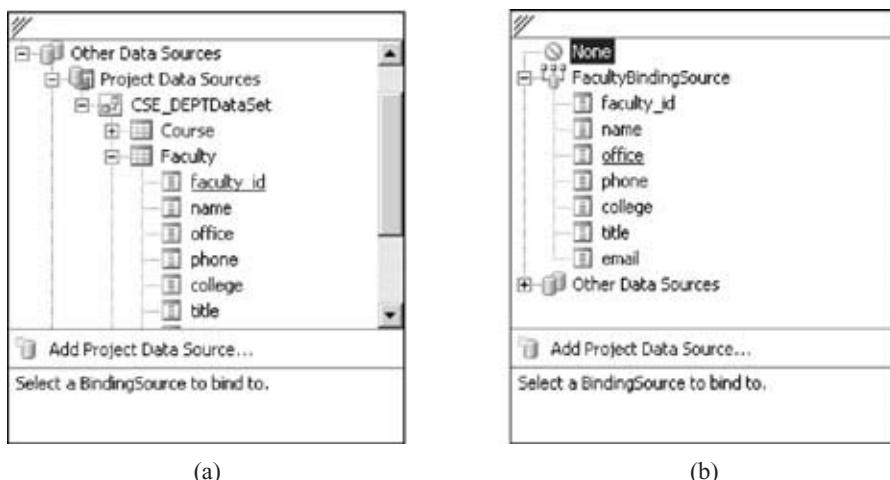
### **5.2.9.2 Data Binding for All Faculty-Information-Related TextBoxes**

As we did in Section 4.11 in Chapter 4, we need to set up a one-to-one relationship between each text box that contains one piece of faculty information and each data column in the Faculty data table in the DataSet. We need to set up this relationship in the GUI of the Insert Faculty form window.

Open that form window by clicking the View Designer button in the Solution Explorer window, and set up a one-to-one relationship for the following text boxes:

- Faculty ID TextBox – txtID
- Faculty Title TextBox – txtTitle
- Faculty Office TextBox – txtOffice
- Faculty Phone TextBox – txtPhone
- Faculty College TextBox – txtCollege
- Faculty Email TextBox – txtEmail

You need to note that we did not set up any binding relationship between the Faculty Name text box and the name column in the Faculty table in the DataSet.



**Figure 5.18.** Select the BindingSource dialog box.

The reason for that is we do not need this relationship and we only need to use this Faculty Name as the dynamic parameter to execute the query function to retrieve the newly inserted record from the database. The Faculty Name is an input parameter and should be entered by the user as the project runs.

First, click the Faculty ID text box and go to the property window, then select the item DataBindings, and expand to the Text subproperty. Click the drop-down arrow next to the Text to open the Select a BindingSource dialog box, and then expand to Other Data Sources|Project Data Sources|CSE\_DEPTDataSet|Faculty. Click the plus icon before the Faculty table to expand to and select the faculty\_id, as shown in Figure 5.18a.

In this way, we have finished setting up the mapping relationship between the Faculty ID text box and the faculty\_id column in the faculty table in the DataSet.

In a similar way, we can finish all other bindings for the rest of the text boxes. For example, to set up the binding relationship between the Office text box and the office column in the Faculty table, click the Office text box and go to the DataBindings property, expand to the Text subproperty, and click the drop-down arrow to open the Select a BindingSource dialog box. This time you will find that there is a binding source named FacultyBindingSource already in there. This is because we created this BindingSource when we set up the binding relationship for our first text box, Faculty ID. Use this BindingSource; expand it and select the office item from this table by clicking it. Your finished binding dialog box should match the one that is shown in Figure 5.18b.

After this data binding has been set up, now we can run the project to test our data validation functionality.

Click the Start Debugging button to run the project. Enter the matched user-name and password, such as jhenry and test, to log on, and select Faculty Information from the Selection form window to open the Faculty form window. Click the Insert button to open the Insert Faculty form window to perform the data insertion.

Now let's test how to insert data using the TableAdapter.Update() method. To do that, select the TableAdapter Update from the comboMethod combo box, and enter the following information into the associated text boxes as the new faculty information:

- |                         |                       |
|-------------------------|-----------------------|
| ■ A66789                | Faculty ID text box   |
| ■ Williams Johnson      | Faculty Name text box |
| ■ Associate Professor   | Title text box        |
| ■ MTC-229               | Office text box       |
| ■ 750-330-5588          | Phone text box        |
| ■ University of Toronto | College text box      |
| ■ wjohnson@college.edu  | Email text box        |

Click the Insert button to insert this new record into the database. This new record will first be inserted into the Faculty table in the DataSet, and then it will be inserted into the Faculty table in the database when the TableAdapter.Update() method is executed. All faculty information-related text boxes, except the Faculty Name text box, will become blank after this data insertion, and the Insert button will be disabled.

Now let's test our data validation functionality by clicking the Select button to retrieve the newly inserted data from the database and redisplay it in the associated text boxes. All information in the newly inserted faculty record is displayed in the associated text boxes, as shown in Figure 5.19.

Note that here we are using the Faculty Name as the criterion to retrieve the newly inserted data, so we need to keep the Faculty Name all the time as the project runs. If you want to add another new record, you need to first change the Faculty ID since the database identifies each record based on the faculty\_id that is the primary key in the Faculty table, and then you can enter all other information into the associated text boxes. By clicking the Insert button (right now it should be enabled), another new record will be inserted into the database. Again, you can test that newly

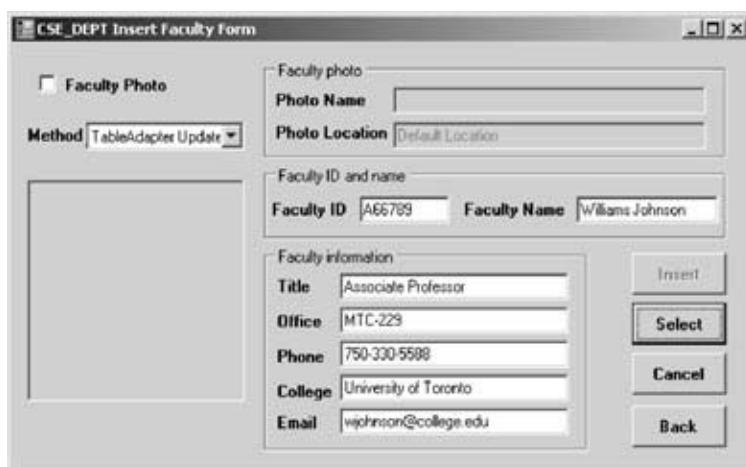


Figure 5.19. The testing status of the data validation.

inserted data by clicking the Select button. Click the Back button to terminate this project after you finish this data validation testing.

So far so good. But traditionally speaking, “a picture is worth a thousand words.” We need to add the newly inserted faculty member’s photo into the picture box to make this data insertion more professional. Let’s do that now.

### 5.2.9.3 Develop Code to Display the Newly Inserted Faculty Photo

Four controls used to help to display the newly inserted faculty member’s photo on this Insert Faculty form window are the Faculty Photo check box, the Photo Name text box, the Photo Location text box, and the PhotoBox picture box. The Faculty Photo check box is used to control the Photo Name and Photo Location text boxes. When the Faculty Photo check box is selected by the user, it means that the user wants to display the faculty photo for this data insertion. The default location to store the faculty photo file is the folder in which our Visual Basic.NET executable file is located. When the default location is used, you do not need to provide the detailed path and the folder in which the photo file is located. If the user wants to use another folder to store the faculty photo file, the full location of that photo file, which includes the path and the name of the folder in which the photo file is located, must be given and entered in the Photo Location text box as the project runs. More importantly, the photo file must be stored in that location before the project runs in order to allow the project to locate it.

The coding should be done in the Select button event procedure during the data validation since the photo name and the photo location can be obtained from two text boxes, the Photo Name and the Photo Location, respectively, during the data insertion process if the Faculty Photo check box is selected. When the Select button is clicked by the user to do the data validation, the newly inserted data should be retrieved from the database and the faculty photo file should be loaded from the Photo Location and displayed in the PhotoBox picture box.

Now let’s develop the coding for this faculty photo to display.

Open the Select button event procedure by first clicking the View Designer button in the Solution Explorer window to open the Insert Faculty form, and then double-clicking the Select button. Add the code shown in Figure 5.20 into this event procedure. Since we have already developed some code for this event procedure, all the code we developed before is indicated with a gray background.

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

- A. A local string variable strPhotoFile is created and used to hold the full location of the faculty photo file, and this full location is obtained later by concatenating the full path and the name of the faculty photo file from two text boxes, txtPhotoName and txtPhotoLocation, respectively.
- B. If the Faculty Photo check box is selected, which means that the user wants to include a faculty photo for this data insertion, the value StretchImage is assigned to theSizeMode property of the PictureBox to make the faculty photo image match the size of the PictureBox.
- C. Next, we need to identify the location in which the faculty photo file is stored. If the content of the Photo Location text box is Default Location, which means that the photo file is located in the folder in which our Visual

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim strPhotoFile As String
    If chkPhoto.Checked = True Then
        PictureBox.SizeMode = PictureBoxSizeMode.StretchImage
    If txtPhotoLocation.Text = "Default Location" Then
        PictureBox.Image = System.Drawing.Image.FromFile(txtPhotoName.Text)
    Else
        strPhotoFile = txtPhotoLocation.Text & "\" & txtPhotoName.Text
        PictureBox.Image = System.Drawing.Image.FromFile(strPhotoFile)
    End If
    FacultyTableApt.ClearBeforeFill = True
    FacultyTableApt.FillByFacultyName(CSE_DEPTDataSet.Faculty, FacultyName)
    If CSE_DEPTDataSet.Faculty.Count = 0 Then
        MessageBox.Show("No matched faculty found!")
        Exit Sub
    End If
    cmdInsert.Enabled = False      're-disable the Insert button
    RemoveFacultyCollection()
End Sub

```

**Figure 5.20.** The modified coding for the Select button event procedure.

Basic.NET executable file is located, we can directly pick up this photo file, based on the name of the photo file, from the txtPhotoName text box and display it by executing the FromFile() method.

- D. If the content of the Photo Location text box is not the Default Location, which means that the user wants to select a special folder to store this faculty photo file, the photo file name obtained from the Photo Name text box is concatenated with the photo path obtained from the Photo Location text box to form a full name of the faculty photo. The backslash (\) is used to separate the path and the name of the faculty photo file. Finally, the FromFile() method is called with the full name of the faculty photo as the argument to display this faculty photo in the PictureBox picture box.

At this point we have completed the coding for the data insertion and the data validation. Before we can run the project to test these tasks, we must first store the faculty photo file in the desired location, either in the default folder or in the user-selected folder, and this location must be entered into the Photo Location text box as the project runs. When this is done, click the Start Debugging button to run our project to test the data validation functionality.

After completing the login and selection process, in the opened Insert Faculty form window, select the Faculty Photo check box and enter the following information into the associated text boxes:

- |              |                       |
|--------------|-----------------------|
| ■ Mhamed.jpg | Photo Name text box   |
| ■ B56789     | Faculty ID text box   |
| ■ Ali Mhamed | Faculty Name text box |
| ■ Professor  | Title text box        |
| ■ MTC-353    | Office text box       |



Figure 5.21. The testing status of the data validation with the faculty photo.

- 750-378-3355 Phone text box
- University of Main College text box
- amhamed@college.edu Email text box

Then click the Insert button to insert this new record into the database. Now the Insert button is disabled. To test the data validation with the faculty photo displaying ability, click the Select button. We want to use the default location to store the faculty photo file, so we keep the content of the Photo Location text box unchanged. Immediately, the newly inserted data is retrieved from the database and displayed in the associated text boxes after the Select button is clicked. Also, the faculty photo is displayed in the photo box, as shown in Figure 5.21.

Click the Back button to terminate the project.

To test how to load the faculty photo from specified folder, such as from the folder **C:\Book5\Chapter 5**, just enter this path into the Photo Location text box as the project runs. But first you have to make sure that the faculty photo file, which is Mhamed.jpg in this application, has been already saved in that folder before the project runs. Now click the Start Debugging button to start the project, perform the login and selection processes to open the Insert faculty form window, select the Faculty Photo check box and enter the above folder into the Photo Location text box, complete all other text boxes with the suitable new faculty information, and then click the Insert button to insert this new record into the database. Next you can click the Select button to test the data validation and to load the faculty photo from the specified folder. Your running Insert Faculty form window should match the one shown in Figure 5.22.

Click the Back button to terminate the project. Our project is successful! The completed project is located in the folder **DBProjects\Chapter 5\InsertWizard Solution** [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

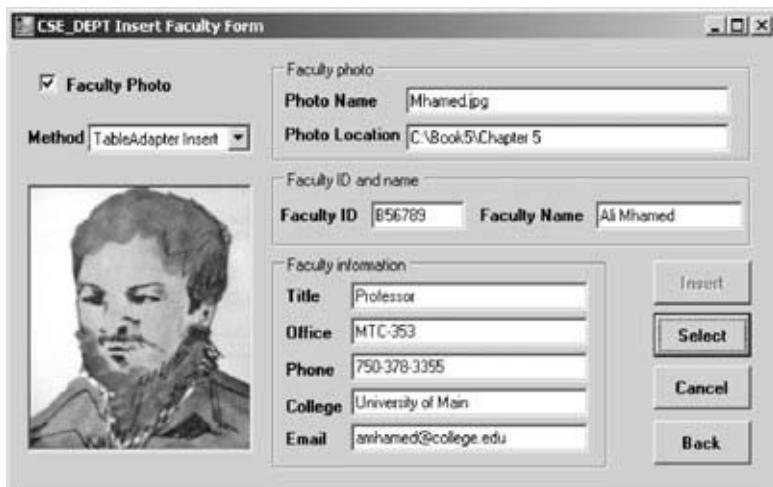


Figure 5.22. The testing status of the data validation with the faculty photo.

In the next section, we will discuss how to insert new data into an SQL Server database.

### 5.3 INSERT DATA INTO THE SQL SERVER DATABASE USING A SAMPLE PROJECT – SQLINSERTWIZARD

In this section, we discuss how to insert data into the SQL Server database. Basically there is no significant difference between inserting data into an Access or an SQL Server database. For both kinds of databases, one can use either method provided by the TableAdapter class we discussed in the last section, such as TableAdapter.Insert() or TableAdapter.Update(), to insert new records into the databases. A small difference between these two databases is that you cannot call stored procedures to insert new records into the Access database by using the TableAdapter (you can do that using the runtime object method by executing the ExecuteNonQuery () method), but you can use stored procedures to finish data insertions for the SQL Server database by using the TableAdapter. In this section, we concentrate on data insertion using the two TableAdapter methods, and in the next section we discuss how to insert data using the stored procedure method.

Because inserting data into an Access database is similar to inserting data into an SQL Server database, here we provide a quick review of inserting data into the latter. Only the differences between inserting data in the two databases will be emphasized.

As we did in the last section, we want to use the existing project to perform this data insertion job to save time and space, so open the project named SelectWizard Project that we developed in Chapter 4 (you can find this project in the folder **DBProjects\Chapter 4** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354)). An SQL Server database is used for that project, that is, an SQL Server 2005 Express database file, CSE\_DEPT.mdf, has been set and connected to that project already. So we don't need to perform any database connection or configuration jobs if we modify that project to create our new project to be used in this section.

### 5.3.1 Modify the Existing Project to Create a New Data Insertion Project

Open Windows Explorer and create a new folder, **Chapter 5**. Then open your net browser and locate our project SelectWizard Project that was developed in the last chapter and is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 4**. Copy this project to our new folder, **Chapter 5**. Change the name of the solution and the project from SelectWizard to SQLInsertWizard, and then double-click SQLInsertWizard Project.vbproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to the Project|SQLInsertWizard Project Properties menu to open the project's property window. Change the Assembly name from SelectWizard Project to SQLInsertWizard Project and the Root namespace from SelectWizard\_Project to SQLInsertWizard\_Project.
- Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to SQLInsertWizard Project. Click OK to close this dialog box.

Go to File|Save All to save these modifications. Now we are ready to develop our GUIs based on the SelectWizard Project we developed in Chapter 4.

### 5.3.2 Create a New Form Window to Insert Data for the Course Form

Recall that in the project SelectWizard, the Course form window contains a button named Insert. We want to use that button to insert a new course record into the Course table in the database. To do that, we need to create a new form window named Insert Course into this new project, and this new form should be triggered by clicking the Insert button on the Course form window as the project runs.

The functionality of this Insert Course form is as follows: As the project runs, after the user has finished the login process and selected Course Information from the Selection form, the Course form window will be displayed. When the user clicks the Insert button, the Insert Course form window will be displayed. This form allows users to use two different methods to insert data into the database: either the TableAdapter.Insert() method or the TableAdapter.Update() method. The form also allows users to enter all pieces of new course information into the appropriate text boxes for the selected faculty member based on the faculty name. By clicking the Insert button, a new course record about a faculty member is inserted into the database. However, if the user wants to reenter those pieces of information before finishing this insertion, the Cancel button can be used and all information entered will be erased. The Select button is used to validate this data insertion by retrieving the newly inserted data. The Back button is used to allow users to return to the Selection form to perform other data operations.

Go to the File|Project|Add Windows Form menu item to open the Add New Item dialog box. Keep the default template, Windows Form, selected, and enter Insert Course Form.vb into the Name box as the name for this new form. Then click the Add button to add this form into our project.

**Table 5.3.** Objects for the Insert Course Form

Type	Name	Text	TabIndex	DropDownStyle
GroupBox	GroupBox1	Method and Faculty Information	0	
Label	Label1	Insert Method	0.0	
ComboBox	ComboMethod		0.1	DropDownList
Label	Label2	Faculty Name	0.2	
ComboBox	ComboName		0.3	DropDownList
GroupBox	GroupBox2	New Course Information	1	
Label	Label3	Faculty ID	1.0	
TextBox	txtFacultyID		1.1	
Label	Label4	Course ID	1.2	
TextBox	txtCourseID		1.3	
Label	Label5	Course Title	1.4	
TextBox	txtCourse		1.5	
Label	Label6	Schedule	1.6	
TextBox	TxtSchedule		1.7	
Label	Label7	Classroom	1.8	
TextBox	txtClassroom		1.9	
Label	Label8	Credits	1.10	
TextBox	txtCredits		1.11	
Label	Label9	Enrollment	1.12	
TextBox	txtEnrollment		1.13	
Button	cmdInsert	Insert	2	
Button	cmdSelect	Select	3	
Button	cmdCancel	Cancel	4	
Button	cmdBack	Back	5	
Form	InsertCourseForm	CSE_DEPT Insert Course Form		

Add the items shown in Table 5.3 into this form.

In addition to the form's properties shown in Table 5.3, the following properties of the form should be set up:

- **AcceptButton:** cmdInsert (set Insert button as default button)
- **StartPosition:** CenterScreen (set the form in the center)

Your finished Insert Course form should match the one shown in Figure 5.23.

The order of the TabIndex values we created in Table 5.3 is to make the Faculty Name combo box the default one and set a focus to it to allow users to select the faculty name without clicking that box first.

Next, let's first configure the CourseTableAdapter and build the insertion data query using the TableAdapter Query Configuration Wizard.

### 5.3.3 Project Initialization and Data Validation Before Data Insertion

Just as we did for the last sample project, data validation must be performed before the course information can be inserted into the database. To save the time and the space, we use a string array to store all course information. In all, we have

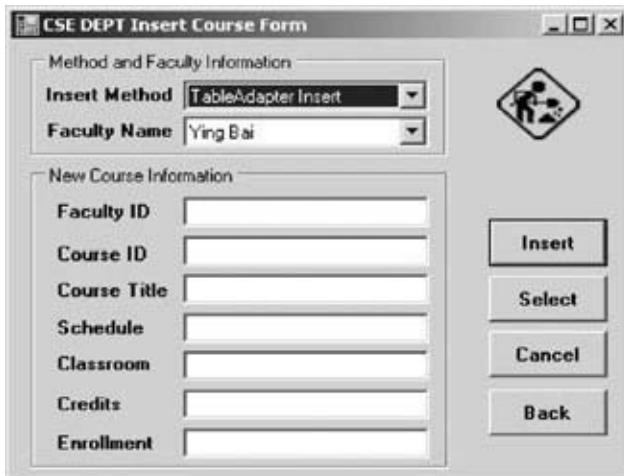


Figure 5.23. The completed Insert Course form window.

six pieces of course information: faculty\_id, course\_id, course schedule, course classroom, course credits, and course enrollment. These should be entered by the user into six text boxes as the project runs. Also, the Faculty Name combo box should be initialized by adding all faculty members to allow users to make a selection from this box as the project runs.

To do this, open the code window of the Insert Course form, select the item InsertCourseForm Events from the Class Name combo box, and select the item Load from the Method Name combo box to open the Form\_Load event procedure. Enter the code shown in Figure 5.24 into this event procedure.

Let's take a look at this piece of coding to see how it works.

- First, we create a string array with the upper bound of 5, which means that this array contains six string items with the index starting from 0, not 1. This

```
(InsertCourseForm Events) ▾ Load ▾
Public Class InsertCourseForm
    Private CourseInfo(5) As String
    Private Sub InsertCourseForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        A
        ComboName.Items.Add("Ying Bai")
        B
        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")
        C
        ComboName.Items.Add("Jeff Henry")
        D
        ComboName.SelectedIndex = 0
        ComboMethod.Items.Add("TableAdapter Insert")
        ComboMethod.Items.Add("TableAdapter Update")
        ComboMethod.SelectedIndex = 0
    End Sub
End Class
```

Figure 5.24. The coding for the Form\_Load event procedure.

- array is used to store six pieces of course information entered by the user in the six associated text boxes.
- All faculty members are added to the ComboName combo box by executing the Add() method.
  - The first item in the box is selected as the default one by setting the SelectedIndex property to 0 (the index of the combo box also starts from 0).
  - Two data insertion methods are added to the ComboMethod combo box to allow users to select either the TableAdapter.Insert() method or the TableAdapter.Update() method to insert the new record. The first method is set up as the default one as the project runs.

Now let's develop the code for the data validation before performing the data insertion. This data validation is to make sure that all text boxes that contain the course information are nonempty. This means that you need to fill all text boxes with a piece of course-related information, and the project does not allow an empty text box or a blank piece of information to be inserted into the database. If you want to intentionally keep some columns in the Course table empty, you need to place a NULL into the associated text box.

Open the Insert button event procedure by double-clicking the Insert button from the Insert Course form window, and enter the code shown in Figure 5.25 into this event procedure.

The functionality of this piece of coding is very simple. First, a user-defined subroutine, InitCourseInfo(), is called to assign each text box's content to the associated string item in the string array, and then another user-defined subroutine, CheckCourseInfo(), is executed to make sure that all text boxes are filled with a piece of course-related information and no empty text box exists. The detailed coding for these two subroutines is shown in Figure 5.26.

The functionality of these two subroutines is as follows:

- The so-called initialization job is to assign each text box's content to the associated string item in the string array. In this way, it is easier for us to check those text boxes to make sure that no text box is empty.
- A For loop is utilized to scan all elements in the string array, which is equivalent to scanning all text boxes, to check whether any of them is empty. A message box with a reminder message will be displayed if this situation occurs.

The other coding job for the project initialization is the coding for the Cancel and the Back command buttons. When the Back button is clicked, the current form should be closed and the project needs to be returned to the Selection form to allow

cmdInsert	▼	Click	▼
<pre>Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click     InitCourseInfo()     CheckCourseInfo() End Sub</pre>			

**Figure 5.25.** The coding for the Insert button event procedure.

```

A Private Sub InitCourseInfo()
    CourseInfo(0) = txtFacultyID.Text
    CourseInfo(1) = txtCourseID.Text
    CourseInfo(2) = txtSchedule.Text
    CourseInfo(3) = txtClassRoom.Text
    CourseInfo(4) = txtCredits.Text
    CourseInfo(5) = txtEnroll.Text
End Sub

B Private Sub CheckCourseInfo()
    Dim pos As Integer
    For pos = 0 To 5
        If CourseInfo(pos) = String.Empty Then
            MessageBox.Show("Fill all Course Information box, enter a NULL for blank column")
            Exit Sub
        End If
    Next
End Sub

```

**Figure 5.26.** The coding for two user-defined subroutines.

users to perform other data actions. The coding for this button is easy. Open its event procedure and enter Me.Close(). Yes, that's it!

The functionality of the Cancel button is to clean up most text boxes' contents to allow users to reenter all course-related information into those text boxes. Open the Insert Course form window and double-click the Cancel button to open its event procedure, and enter the code shown in Figure 5.27 into this event procedure.

The coding for this event procedure is straightforward and easy to understand. All text boxes' contents, except the Course ID, are cleaned up to allow users to reenter new course information. The first reason we keep the Course ID text box's contents unchanged is that this piece of information will be used later for the data validation purpose since we need to retrieve the newly inserted course record from the database based on the Course ID. Another reason is that we want to prevent the TextChanged event from occurring to reenable the Insert button (refer to Section 5.3.7 and Figure 5.36 below).

Now we can start to configure the TableAdapters to build our data insertion query.

```

cmdCancel Click
Private Sub cmdCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdCancel.Click
    txtFacultyID.Text = String.Empty
    txtCourse.Text = String.Empty
    txtSchedule.Text = String.Empty
    txtClassRoom.Text = String.Empty
    txtCredits.Text = String.Empty
    txtEnroll.Text = String.Empty
End Sub

```

**Figure 5.27.** The coding for the Cancel button' event procedure.

### 5.3.4 Configure the TableAdapter and Build the Data Insertion Query

Recall that when we built our sample database CSE\_DEPT in Chapter 2, there was no faculty name column in the Course table, and the only relationship existing between the Faculty and the Course tables was the faculty\_id, which is the primary key in the Faculty table but a foreign key in the Course table. As the project runs and the Insert Course form window is displayed, 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 belong to two TableAdapters, the FacultyTableAdapter and the CourseTableAdapter. Now let's create these two query functions under two TableAdapters.

Recall that in Section 4.14 in the last chapter, we built a query function under the FacultyTableAdapter, and the function is named FindFacultyIDByName(). So in this section, we don't need to redo this job and can use that query function directly from this project. What we need to do is to create the query function to insert a new course record based on the selected faculty.id obtained from the first query.

Open the Data Sources window by clicking the Data>Show Data Sources menu item from the menu bar. Then right-click anywhere inside this window and select the Edit DataSet with Designer item to open the DataSet Designer Wizard. Right-click on the last line of the Course table and choose the Add|Query item to open the TableAdapter Query Configuration Wizard. Keep the default item Use SQL statements selected and click the Next button to go to the next window. Select the INSERT item by checking this radio button, and click Next again to continue. Click the Query Builder button in the next window to open the Query Builder dialog box, which is shown in Figure 5.28.

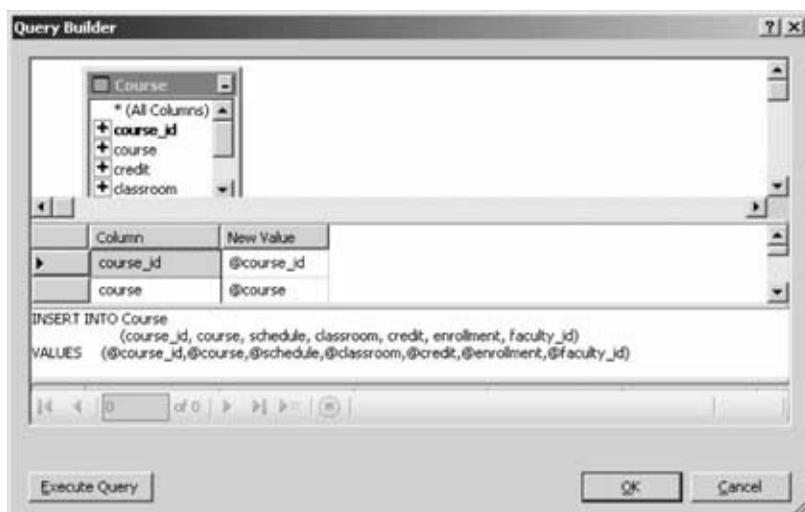


Figure 5.28. The Query Builder dialog box.

The default INSERT statement is matched to our data insertion command, so we keep it as our query command and click the OK button to go to the next window.

In the next window, the Query Builder needs us to confirm this query function. Two queries are displayed in this window. The first one is an INSERT statement and is what we need for our data insertion, but the second one is the SELECT statement. We won't need this second statement until we perform our data validation after our data insertion is completed. So highlight the second statement and delete it from this window. Click the Next button to continue.

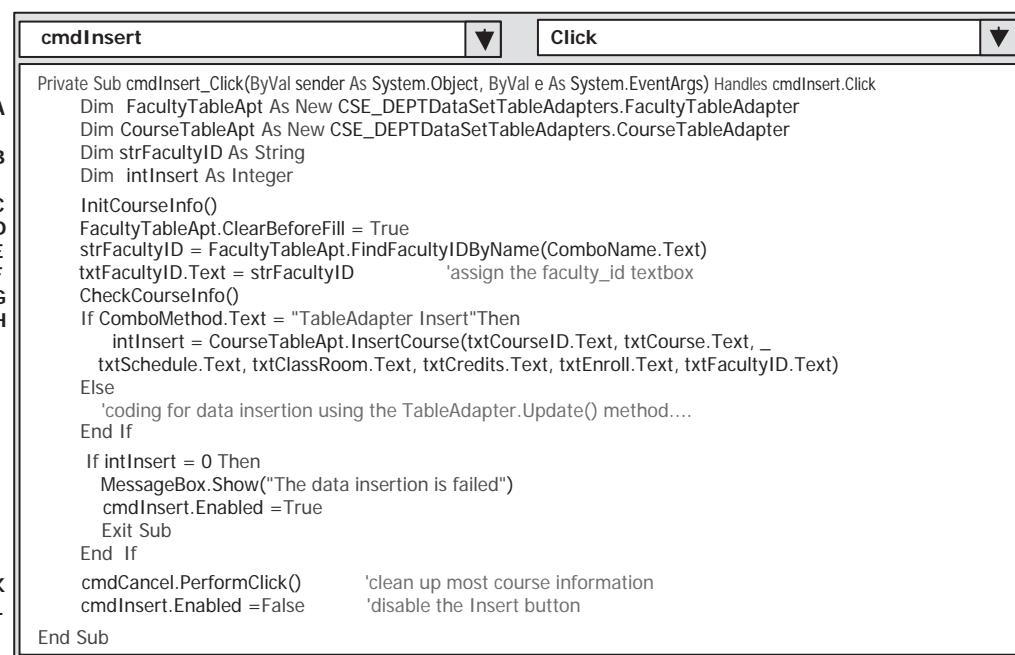
The next window needs us to enter the name for our query function. Enter InsertCourse into this name box and click the Next button to go to the next window, and click the Finish button to complete this Query Builder process and exit this dialog box.

Now that we have finished the configuration and the query building of the Course TableAdapter, we need to develop the code to execute this data insertion function.

### 5.3.5 Develop the Code to Insert Data Using the TableAdapter.Insert Method

Open the Insert Course form window and double-click the Insert button to open its event procedure, and then enter the code shown in Figure 5.29 into this event procedure.

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

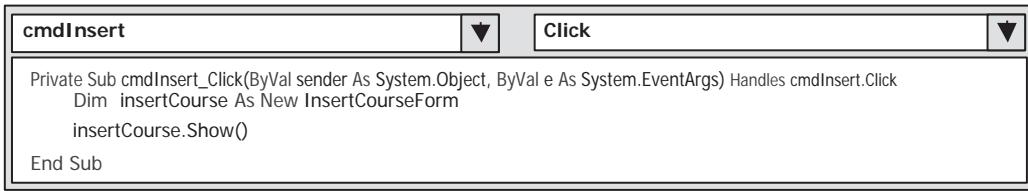


The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdInsert" and the dropdown menu says "Click". The code is labeled with letters A through L on the left side:

```
Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim CourseTableApt As New CSE_DEPTDataSetTableAdapters.CourseTableAdapter
    Dim strFacultyID As String
    Dim intInsert As Integer
    InitCourseInfo()
    FacultyTableApt.ClearBeforeFill = True
    strFacultyID = FacultyTableApt.FindFacultyIDByName(ComboName.Text)
    txtFacultyID.Text = strFacultyID           'assign the faculty_id textbox
    CheckCourseInfo()
    If ComboMethod.Text = "TableAdapter Insert" Then
        intInsert = CourseTableApt.InsertCourse(txtCourseID.Text, txtCourse.Text, _
        txtSchedule.Text, txtClassRoom.Text, txtCredits.Text, txtEnroll.Text, txtFacultyID.Text)
    Else
        'coding for data insertion using the TableAdapter.Update() method....
    End If
    If intInsert = 0 Then
        MessageBox.Show("The data insertion is failed")
        cmdInsert.Enabled = True
        Exit Sub
    End If
    cmdCancel.PerformClick()          'clean up most course information
    cmdInsert.Enabled = False        'disable the Insert button
End Sub
```

Figure 5.29. The coding for the Insert button event procedure.

- A. Two TableAdapter objects are created at the beginning of this procedure since we need to use them to perform the data query (get the faculty\_id from the Faculty table based on the faculty name) and the data insertion (insert a new course record in the Course table based on the faculty\_id).
- B. The first query function, FindFacultyIDByName(), which we built in Section 4.14 in Chapter 4, is used to return a single data value, faculty\_id, from the Faculty table based on the faculty name selected by the user from the ComboName combo box as the project runs. A local string variable, strFacultyID, is created and will be used to hold the returned faculty\_id obtained from the first query function. The local integer variable intInsert is used to hold the returning status of calling the second query function InsertCourse().
- C. The initialization subroutine InitCourseInfo() is executed to assign each string item in the form-level string array CourseInfo() to the associated text box to set up a one-to-one relationship between them.
- D. The ClearBeforeFill property of the TableAdapter is set to True to clean up the Faculty data table to make it ready to be filled with new records.
- E. The first query is executed to pick up the faculty\_id from the Faculty table based on the faculty name stored in the ComboName combo box.
- F. The returned faculty\_id value is assigned to the Faculty ID text box. This step is necessary, and you will find later that as the project runs, only the faculty name will be displayed and can be selected by the user to perform this data insertion. The user has no knowledge of the faculty\_id. So when performing the data insertion, a NULL must be filled into the faculty\_id text box in order to make this data insertion operate properly.
- G. Next, we need to check and make sure that no text box is empty.
- H. If the user selected the TableAdapter Insert() method to perform this data insertion, the second query function, InsertCourse(), which we just built in the last section, is executed to insert a new course record into the Course table in the database.
- I. If the user selected the TableAdapter.Update() method to perform this data insertion, the associated coding that will be developed in the next section will be executed to first add the new course record into the Course table in the DataSet and then update it in the Course table in the database.
- J. The second query function, InsertCourse(), will return an integer to indicate the execution status of calling this function. That is, the returned integer's value equals the number of records that have been successfully added or inserted into the database. A returned zero means that no record has been inserted into the database and this data insertion has failed. If that case occurs, a message box with a warning message is displayed and the project is exited. The Insert button is enabled again to allow users to perform another insertion.
- K. After a new record has been successfully inserted into the database, we need to validate this operation. In order to do that, first we need to clean up most pieces of course information from the associated text boxes. Executing the PerformClick() method of the Cancel button is equivalent to clicking the Cancel button to trigger its event procedure to perform this cleaning job.



```
cmdInsert Click
Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim insertCourse As New InsertCourseForm
    insertCourse.Show()
End Sub
```

Figure 5.30. The coding for the Insert button event procedure in the Course form.

- L. Finally, we need to disable the Insert button to avoid multiple insertion operations of the same record. This Insert button will be enabled again if the user types a new faculty\_id into the Faculty ID text box, which means that the user wants to insert a new course record, and this typing will create a TextChanged event for the Faculty ID text box and trigger its event procedure.

Now let's test this data insertion by running our project. We have two ways to run the project. One way is to trigger the Insert Course form window by clicking the Insert button in the Course form window. To start the project in this way, we must start from the LogIn form and then continue to Selection form and the Course form by selecting the Course Information item from the Selection form window. The other way is simple. We can start the project directly from the Insert Course form window. By setting up the Startup form box with the different form window from the SQLInsertWizard Project Properties window, we can decide which way to start the project.

To start the project in the first way, we need to do a little modification to the Course form to add coding to the Insert button event procedure. Open the Course form window and double-click the Insert button to open its event procedure, and enter the code shown in Figure 5.30 into this event procedure.

When the Insert button is clicked, a new InsertCourseForm object is created and the Show() method is called to display this form window.

Before we can run the project, go to the Project|SQLInsertWizard Project 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 LogIn Form. Now let's run the project to test our data insertion by clicking the Start Debugging button. The running status of the project is shown in Figure 5.31.

Enter a suitable username and password, such as jhenry and test, for the LogIn form, and select the Course Information item from the Selection form window to open the Course form window. Then click the Insert button to open the Insert Course Form window.

At this time, we can only test the first data insertion method, TableAdapter.insert(), since we have not developed the code for the second method. Select a faculty from the Faculty Name combo box, and enter the new course information, shown in Figure 5.31, into the associated text box. Since we have no knowledge about the Faculty ID, a NULL is entered at this moment. Click the Insert button; all text boxes that contain the new course information are cleaned except the Course ID text box. We want to keep the content of this Course ID text box unchanged to prevent a TextChanged event from occurring, and therefore to avoid having Insert

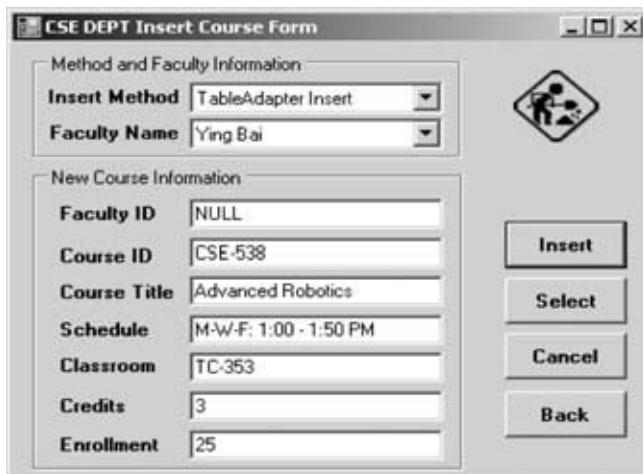


Figure 5.31. The running status of the project.

button reenabled to allow multiple insertions of the same data. The running status of this testing is shown in Figure 5.32, and our data insertion is successful.

Next let's develop the code to insert new course data using the second method, TableAdapter.Update(). But before we can do this coding, we must perform the data binding between the text boxes storing the new course information and the data columns in the Course table in the DataSet. The reason for this is that the Insert Course form window does not recognize this DataSet until the data binding is performed and a one-to-one relationship is set up between each text box that stores the new course information and the associated column in the Course table. Our DataSet, CSE\_DEPTDataSet, will be automatically added to the current form window and will be recognized after this data binding is completed.

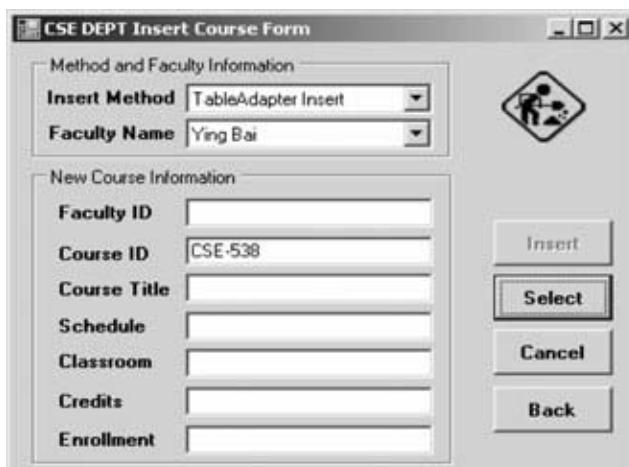


Figure 5.32. The execution status of the data insertion.

### 5.3.6 Perform Data Bindings for the Insert Course Form Window

The following controls need to be bound with the associated columns in the Course table:

- Faculty ID textbox
- Course ID text box
- Course Title text box
- Schedule text box
- Classroom text box
- Credits text box
- Enrollment text box

Open the Insert Course Form window and select the first control from the above list, say, Faculty ID text box, and then go to the property window, select the DataBindings item and browse to the Text subproperty, click on the drop-down arrow that is located at the right border to open the Select a Data Source dialog box, and then expand to the Course data table and select the faculty\_id item, as shown in Figure 5.33a. In this way, we finish the data binding for the first control.

In a similar way, you can perform all other bindings one by one. One point to be noted is that after you finish the first binding, a binding source named CourseBindingSource is created, and you need to use it to perform the data binding for all other controls listed above. An example of data binding for the course title is shown in Figure 5.33b.

As soon as you finish the first binding, three objects, CSE\_DEPTDataSet, CourseBindingSource, and CourseTableAdapter, are automatically added to the current form window at the bottom.

Now we can develop the second method to perform the data insertion.

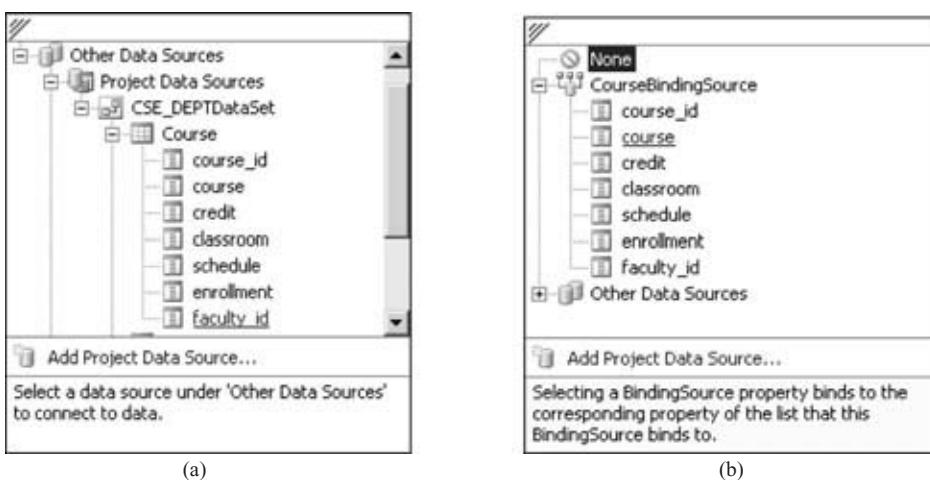


Figure 5.33. Data binding process for the Course form.

```

cmdInsert Click
A
Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim CourseTableApt As New CSE_DEPTDataSetTableAdapters.CourseTableAdapter
    Dim newCourseRow As CSE_DEPTDataSet.CourseRow
    Dim strFacultyID As String
    Dim intInsert As Integer
    InitCourseInfo()
    FacultyTableApt.ClearBeforeFill = True
    strFacultyID = FacultyTableApt.FindFacultyIDByName(ComboName.Text)
    txtFacultyID.Text = strFacultyID           'assign the faculty_id textbox
    CheckCourseInfo()
    If ComboMethod.Text = "TableAdapter Insert" Then
        intInsert = CourseTableApt.InsertCourse(txtCourseID.Text, txtCourse.Text, _
                                                txtSchedule.Text, txtClassRoom.Text, txtCredits.Text, txtEnroll.Text, txtFacultyID.Text)
    Else
        newCourseRow = Me.CSE_DEPTDataSet.Course.NewCourseRow
        newCourseRow = InsertCourseRow(newCourseRow)
        CSE_DEPTDataSet.Course.Rows.Add(newCourseRow)
        intInsert = CourseTableApt.Update(CSE_DEPTDataSet.Course)
    End If
    If intInsert = 0 Then
        MessageBox.Show("The data insertion is failed")
        cmdInsert.Enabled = True
        Exit Sub
    End If
    cmdCancel.PerformClick()      'clean up most course information
    cmdInsert.Enabled = False     'disable the Insert button
End Sub
B
C
D
E

```

Figure 5.34. The modified coding for the Insert button event procedure.

### 5.3.7 Develop the Code to Insert Data Using the TableAdapter.Update Method

Open the Insert button event procedure and add the code shown in Figure 5.34 into this event procedure. Since we have already developed some code for the first data insertion method, all of the existing code is indicated with a gray background.

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

- 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 to the DataRow object. Also, we need to create a row in the Course data table; the CourseRow is selected as the DataRow class.
- Next, we need to create a new object of the NewCourseRow class.
- A user-defined function, InsertCourseRow(), is called to add all information about the new course, which is stored in seven text boxes, into this newly created DataRow object. The functionality of the coding for this user-defined function is explained below. The function returns a completed DataRow in which all information about the new course has been added.
- The completed DataRow is added to the Course table in our DataSet object. One point to note is that adding a new record into the data table in the

```

Private Function InsertCourseRow(ByRef courseRow As CSE_DEPTDataSet.CourseRow) As CSE_DEPTDataSet.CourseRow
    courseRow.faculty_id = txtFacultyID.Text
    courseRow.course_id = txtCourseID.Text
    courseRow.course = txtCourse.Text
    courseRow.schedule = txtSchedule.Text
    courseRow.classroom = txtClassRoom.Text
    courseRow.credit = txtCredits.Text
    courseRow.enrollment = txtEnroll.Text
    Return courseRow
End Function

```

**Figure 5.35.** The coding for the user-defined function InsertCourseRow.

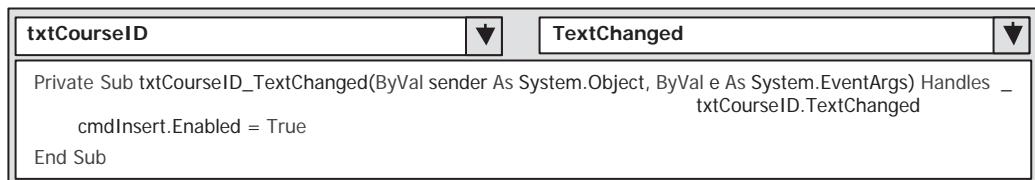
DataSet has nothing to do with adding a new record into the data table in the database. The data tables in the DataSet are only mappings of the real data tables in the database. To add this new record into the database, one needs to perform the next step.

- E. The TableAdapter's method Update() is executed to add this new record to the real database. As we mentioned before, you can control the amount of data to be added to the database by passing different arguments. Here we only want to add one new record into the Course table, so a data table is passed as the argument. This Update() method returns an integer value to indicate whether this update is successful or not. The value of this returned integer is equal to the number of rows that have been successfully added to the database. A returned value of zero means that this update has failed since no new row has been added to the database.

Now let's develop the coding for the user-defined function InsertCourseRow(). Open the code window and enter the code shown in Figure 5.35 into this function.

Let's see how this piece of code works.

- A. In Visual Basic.NET, unlike C or 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 InsertCourseRow() 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 the passing by 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.
- B. Seven pieces of new information stored in the associated text boxes are added to this new DataRow object, that is, added to a new row of the Course table in the DataSet.



```

txtCourseID TextChanged
Private Sub txtCourseID_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles txtCourseID.TextChanged
    cmdInsert.Enabled = True
End Sub

```

Figure 5.36. The coding for the TextChanged event procedure of the text box.

- C. Finally, this completed DataRow object is returned to the calling procedure. Another advantage of using this passing by 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.

We have one more step to finish this data insertion process. As mentioned above, after one new piece of data is inserted into the database, the Insert button will be disabled to avoid multiple insertions of the same data. In order to reenable this button, you have to change the content of the Course ID text box, which means that you want to insert a new course record. To make this happen, we need to use the TextChanged event of the Course ID text box to trigger its event procedure to enable the Insert button. To do that, open the Insert Course Form window and double-click the Course ID text box to open its TextChanged event procedure. Enter one line of code into this event procedure, as shown in Figure 5.36.

Now we have finished developing two methods to perform the data insertion job. Next, we need to design a method to validate this data insertion to confirm that it is successful.

### 5.3.8 Data Validation for Newly Inserted Records

As we did for the last sample project, InsertWizard Project, we can develop code for the Select button event procedure to perform this data insertion validation. But we have another, better way to do this job. First, we will still use the Select button in the Insert Course Form window to do this validation job, and then we discuss the better method later.

To use the Select button and its event procedure to confirm our data insertion, first you need to open the Insert Course form window, and then double-click the Select button to open its event procedure. The functionality of this event procedure is to retrieve the newly inserted records from the database and display them in the associated text boxes in this form, such as txtFacultyID, txtCourseID, txtCourse, txtSchedule, txtClassroom, txtCredits, and txtEnroll. To do that, we need to first build a query function to retrieve the newly inserted course record by filling the Course table with the CourseID as the dynamic parameter. The query function is named FillByCourseID(). Now let's build this query function using the TableAdapter Query Configuration Wizard.

Go to the Data|Show Data Sources menu item to open the Data Sources window, and then right-click anywhere inside this window and select the item Edit DataSet with Designer to open the DataSet Designer window. Right-click on the last item of the Course table and select the Add|Query item to open the

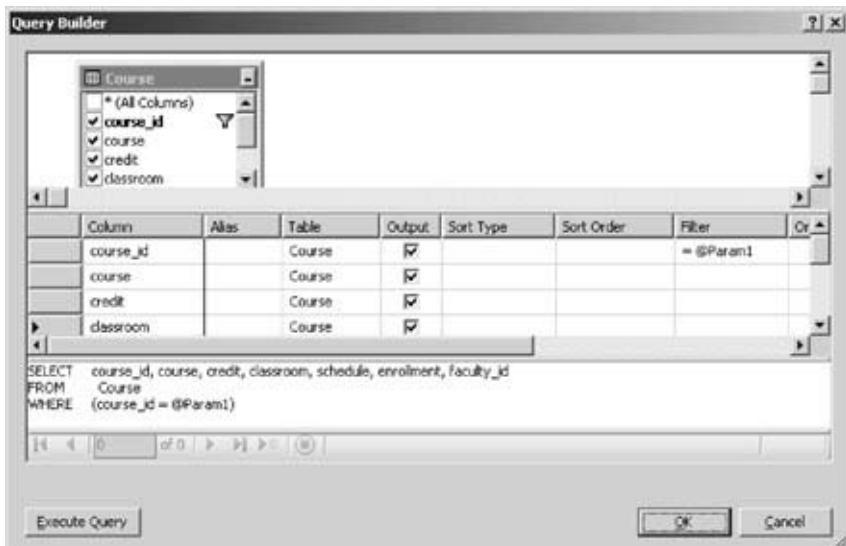


Figure 5.37. The Query Builder window.

TableAdapter Query Configuration Wizard. Keep the default selection Use SQL statements unchanged, and click the Next button to go to the next window. In the next window, keep the default selection SELECT, which returns rows, and then click the Next button to continue. Click the Query Builder button in the next window to open the Query Builder dialog box, which is shown in Figure 5.37.

In the middle pane, move the cursor along the first row (course\_id row) to the Filter column, click that column and type "?", then press the Enter key on your keyboard. Immediately a dynamic parameter named =@Param1 appears in the Filter column and a WHERE clause is attached at the end of the SELECT statement in the bottom pane, which is shown in Figure 5.37.

Click OK to go to the next window, and click Next to confirm this query. In the next window, enter FillByCourseID into the Method name box as our query function's name and click Next. Finally, click the Finish button to close this wizard.

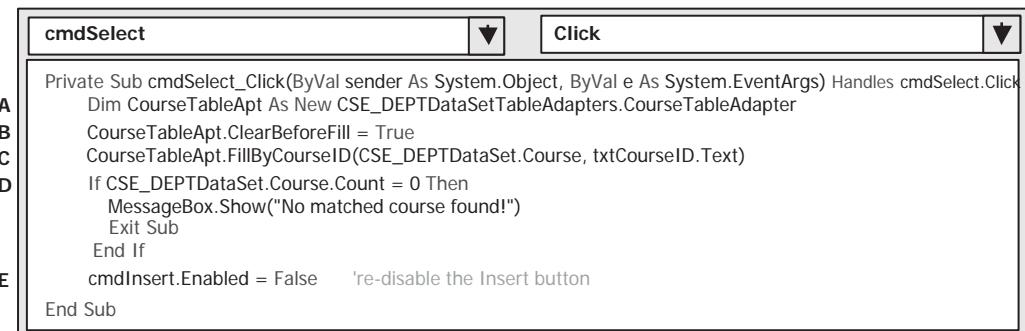
Now let's do our coding for the Select button event procedure to call the query function we just built to validate the new course record. We will use this as our first method to validate the newly inserted data.

### 5.3.8.1 Develop Code to Perform the Data Validation

With the Select button event procedure open, enter the code shown in Figure 5.38 into this event procedure.

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

- A. Since the query function FillByCourseID(), which we just built, is based on the CourseTableAdapter, first we need to create an object of the TableAdapter in order to use that query function.
- B. As we did before, we need to clean up all old contents from the Course data table in the DataSet before we can call the query function to fill that table. This will make our data query neat.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdSelect" and the dropdown menu says "Click". The code is as follows:

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim CourseTableApt As New CSE_DEPTDataSetTableAdapters.CourseTableAdapter
    CourseTableApt.ClearBeforeFill = True
    CourseTableApt.FillByCourseID(CSE_DEPTDataSet.Course, txtCourseID.Text)
    If CSE_DEPTDataSet.Course.Count = 0 Then
        MessageBox.Show("No matched course found!")
        Exit Sub
    End If
    cmdInsert.Enabled = False      're-disable the Insert button
End Sub

```

**Figure 5.38.** The coding for the Select button event procedure.

- C. Then we can call this query function to fill the Course table in the DataSet with the data retrieved from the Course table in the database. Remember that this function contains one argument, which is a dynamic parameter that is used for this query and is the course\_id located in the WHERE clause. Since we have kept the content of the Course ID text box unchanged, now we can use it as the argument for this query function.
- D. By checking the Count property of the Course table, we can determine whether this fill is successful or not. This property contains the number of records that have been retrieved from the database correctly. A zero means that no record has been found and retrieved from the database and this fill (that is, this data insertion) has failed. In that case, a message box will be displayed with a warning message to indicate this situation, and the program is exited.
- E. If this query function is successful, our data insertion is fine and the newly inserted data has been retrieved from the database and displayed in the associated text boxes. In Section 5.3.5, however, recall that in order to avoid the multiple insertions of the same data, the Insert button is disabled (refer to step **L** in Figure 5.29) and all text boxes storing the course information, except the Course ID text box, are cleaned up (refer to step **K** in Figure 5.29) as soon as the user clicks the Insert button to perform the data insertion operation. Also refer to the coding shown in Figure 5.36, where the Insert button will be enabled again when the content of the Course ID text box is changed, which means that the user wants to perform another new data insertion operation. There is a trick here; the successful query function will bring the newly inserted record back and display the information in all course-related text boxes that are blank now. This displaying is equivalent to triggering the TextChanged event for the Course ID text box, and therefore, the Insert button will be enabled. That is enabling operation wrong since no new record will be inserted into the database and this TextChanged event is only triggered by the display of the newly inserted records just retrieved back from the database. To correct this wrong enabling operation, we need to reenable the Insert button.

At this point, we complete the coding to validate the newly inserted course record. Now let's test this data validation by running our project. Click the Start

Debugging button to run the project, enter a suitable username and password, such as jhenry and test, in the LogIn form window, and select the Course Information item from the Selection form window to open the Course form. On the opened Course form, click the Insert button to open the Insert Course form window, which is shown in Figure 5.39. Enter the following course-related information in the associated text boxes:

- |                         |                       |
|-------------------------|-----------------------|
| ■ NULL                  | Faculty ID text box   |
| ■ CSE-665               | Course ID text box    |
| ■ Advanced Robotics     | Course Title text box |
| ■ T-H: 11:00 – 12:20 PM | Schedule text box     |
| ■ TC-226                | Classroom text box    |
| ■ 3                     | Credits text box      |
| ■ 25                    | Enrollment text box   |

Your finished information should match that shown in Figure 5.39.

Now click the Insert button to insert this new course-related information into the database. Immediately, all information, except the Course ID, is cleaned after this insertion. To validate the insertion, click the Select button to retrieve the newly inserted course record from the database and redisplay it in the associated text boxes. The result of this validation is shown in Figure 5.40.

It can be seen that the NULL in the Faculty ID text box is replaced by a valid faculty ID (B78880) after this validation. Click the Back button to terminate this validation.

Our first method to validate the newly inserted record is successful. Next, we show you a much easier way to perform this data validation, using the Select button's event procedure in the Course form window.

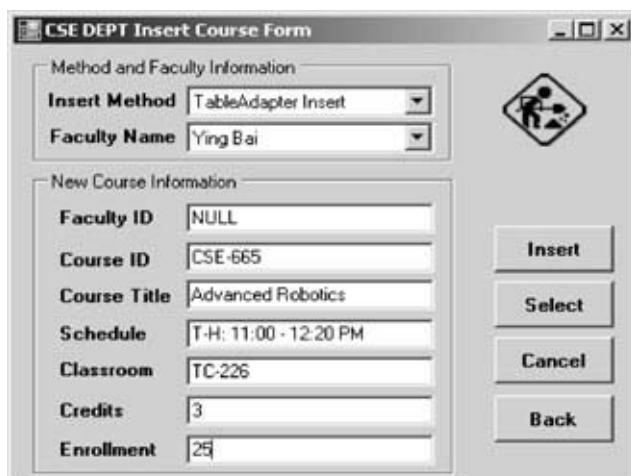


Figure 5.39. The running status of the Insert Course form window.

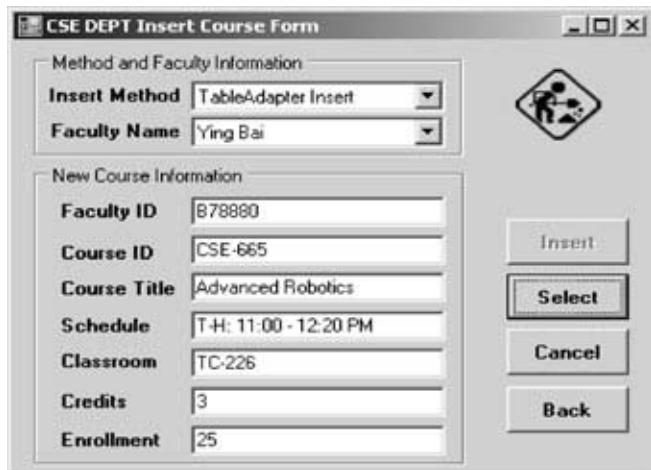


Figure 5.40. The data validation process.

### 5.3.8.2 Use the Select Button in the Course Form to Perform the Data Validation

Using this method to validate the newly inserted data is very simple and efficient. The basic idea of this method is to use the query function `FillByFacultyID()`, which we developed in Figure 4.58 in Section 4.14 of Chapter 4, to retrieve newly inserted records from the Course table and display them in a Course List box in the Course form window. Since that query function has been developed, we can use it. The Select button event procedure contains two query functions; `FindFacultyIDByName()` and `FillByFacultyID()`. We need both functions to validate our newly inserted record.

To use this method to validate the newly inserted data, run the project and go to the Course form window, and then click the Insert button to open the Insert Course form window. Enter all course-related information in the associated text boxes as we did in the last section, then click the Insert button to insert those pieces of information in the database. To validate this new record, click the Back button to close the Insert Course form window and return to the Course form window. Now you can click the Select button to retrieve that newly inserted record and display it in the Course List box. By clicking that new record, all pieces of course-related information are displayed in the associated text boxes, as shown in Figure 5.41.

Compared with our first method, this method is simple and efficient since all code has been developed and is ready to be used. Click the Back button on the Course form to close the Course form and the Exit button on the Selection form to terminate the project.

At this point, we complete the insertion new data into the SQL Server database using two methods. Next, we want to discuss how to insert a new record into the database using stored procedures. Recall that we did not provide this kind of discussion for the Microsoft Access database in Section 5.2 since the Microsoft Access database does not allow users to use stored procedures when using the Visual

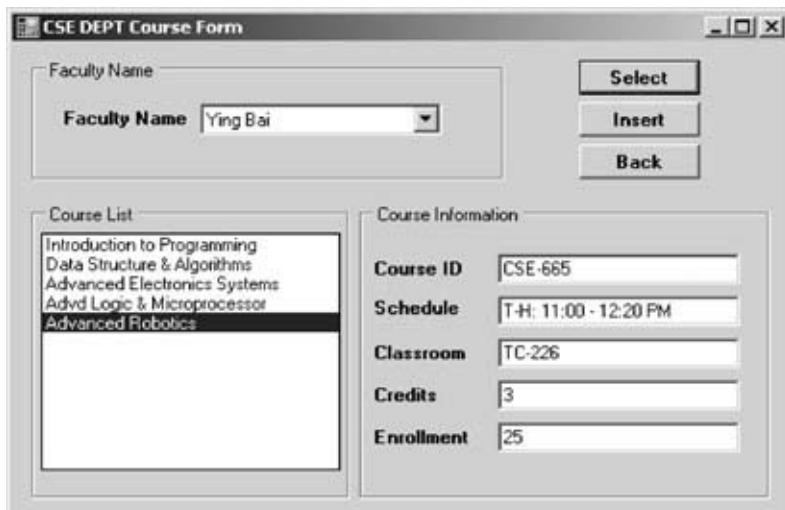


Figure 5.41. The data validation using the Course form window.

Basic.NET Design Tools and Wizards such as the TableAdapter Query, Configuration Wizard, and Query Builder. But the SQL Server database does allow users to use stored procedures to perform different data actions.

### 5.3.9 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 stored procedures. To make it simple, we still use the Course and Insert Course form windows to insert a new course record into the Course table in the database using a stored procedure. To do that, first we need to create a stored procedure named InsertCourseSP under the Course table using the TableAdapter Query Configuration Wizard, and then we need to make some modifications to the coding of the Insert button event procedure and the Form\_Load event procedure of the Insert Course form. The modifications include the following:

1. Add one more item, Stored Procedure, into the combo box control ComboBoxName to allow users to select this method to perform the data insertion.
2. Add one more ElseIf clause for the If block to allow users to select the second method, TableAdapter.Update(), and add one more Else clause for the third method, Stored Procedure, to allow users to select this method.
3. Add the associated coding to call the built stored procedure to perform the data insertion.

Let's first create a stored procedure under the Course table using the TableAdapter Query Configuration Wizard.

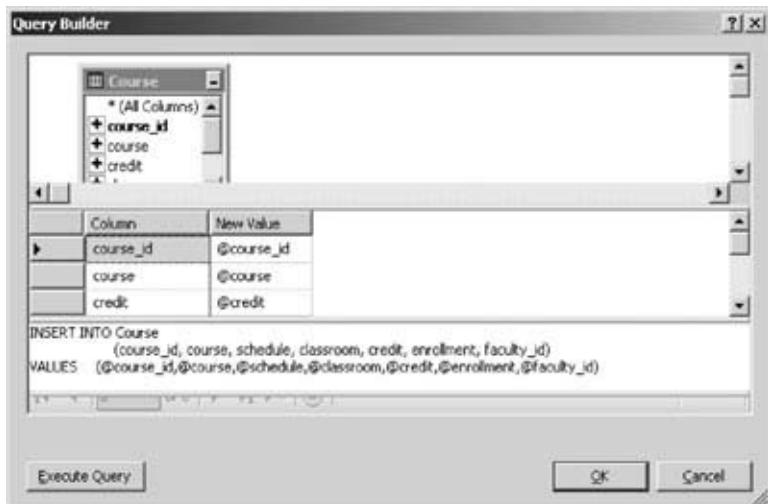


Figure 5.42. The opened Query Builder dialog box.

### 5.3.9.1 Create the Stored Procedure Using the TableAdapter Query Configuration Wizard

Open the Data Sources window by clicking the Data|Show Data Sources menu item, and then right-click on any location inside the Data Sources window and select the Edit the DataSet with Designer item from the popup menu to open this window. Right-click the last item from the Course table and select the Add|Query item from the popup menu to open the TableAdapter Query Configuration Wizard. Check the Create new stored procedure radio button since we want to create a new stored procedure to do the data insertion, and click the Next button to go to the next window. Check the INSERT radio button and click Next to continue. Click the Query Builder button in the opened window since we need to build a new query. The Query Builder dialog box opens and is shown in Figure 5.42.

Make sure that the order of the inserted columns in the INSERT INTO statement is identical to the order shown in Figure 5.42, since this order is very important and it must be identical to the order of the columns in the stored query procedure that will be called from the Insert button event procedure later. Click the OK button to go to the next dialog box to confirm our query function, which is shown in Figure 5.43.

Since we only need to insert a record into the database, highlight the second SELECT statement and delete it by pressing the Delete key on your keyboard. Click the Next button again, and enter InsertCourseSP as the name of this stored query procedure into the name box. Click the Next and the Finish buttons to end this process.

### 5.3.9.2 Modify the Code to Perform the Data Insertion Using the Stored Procedure

The first modification is to add one more item, Stored Procedure into the combo box ComboName 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

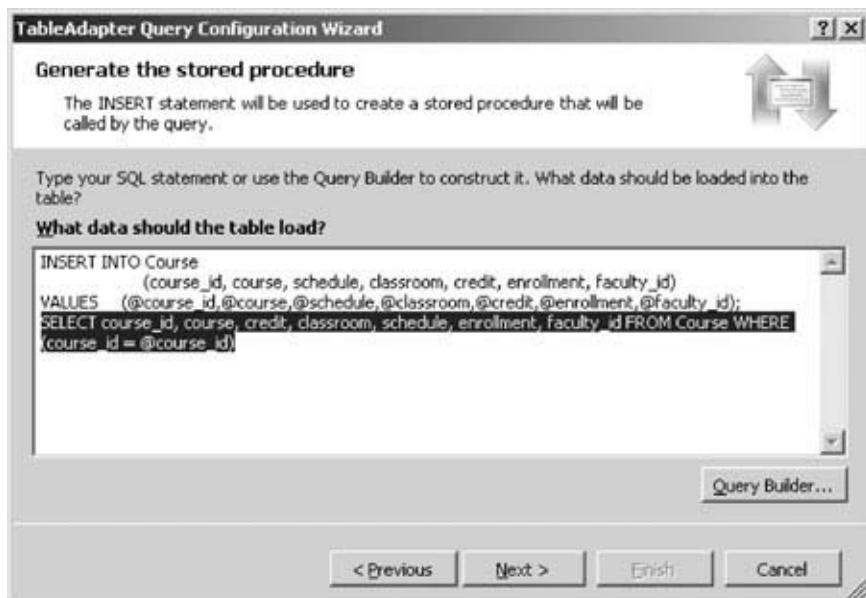


Figure 5.43. The confirmation dialog box.

the item (InsertCourseForm Events) from the Class Name combo box, and then selecting the item Load from the Method Name combo box in the code window of the Insert Course Form window. Add one more line of code, shown in Figure 5.44, into this event procedure. The code we developed before is indicated with a gray background.

Now let's perform the second and third modifications. Both modifications are performed inside the Insert button event procedure. Open the Insert button event procedure and add the code shown in Figure 5.45 into this event procedure. The newly added code is indicated by steps **A** and **B**. All other code we developed before is highlighted with a gray background.

```
(InsertCourseForm Events) Load
Private Sub InsertCourseForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.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 Insert")
    ComboMethod.Items.Add("TableAdapter Update")
    ComboMethod.Items.Add("Stored Procedures")
    ComboMethod.SelectedIndex = 0
End Sub
```

Figure 5.44. The modification to the Form\_Load event procedure.

cmdInsert      Click

```

Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim CourseTableApt As New CSE_DEPTDataSetTableAdapters.CourseTableAdapter
    Dim newCourseRow As CSE_DEPTDataSet.CourseRow
    Dim strFacultyID As String
    Dim intInsert As Integer
    InitCourseInfo()
    FacultyTableApt.ClearBeforeFill = True
    strFacultyID = FacultyTableApt.FindFacultyIDByName(ComboName.Text)
    txtFacultyID.Text = strFacultyID           'assign the faculty_id textbox
    CheckCourseInfo()
    If ComboMethod.Text = "TableAdapter Insert" Then
        intInsert = CourseTableApt.InsertCourse(txtCourseID.Text, txtCourse.Text, _
        txtSchedule.Text, txtClassRoom.Text, txtCredits.Text, txtEnroll.Text, txtFacultyID.Text)
    ElseIf ComboMethod.Text = "TableAdapter Update" Then
        newCourseRow = Me.CSE_DEPTDataSet.Course.NewCourseRow()
        newCourseRow = InsertCourseRow(newCourseRow)
        CSE_DEPTDataSet.Course.Rows.Add(newCourseRow)
        intInsert = CourseTableApt.Update(CSE_DEPTDataSet.Course)
    Else
        intInsert = CourseTableApt.InsertCourseSP(txtCourseID.Text, txtCourse.Text, _
        txtSchedule.Text, txtClassRoom.Text, txtCredits.Text, txtEnroll.Text, txtFacultyID.Text)
    End If
    If intInsert = 0 Then
        MessageBox.Show("The data insertion is failed")
        cmdInsert.Enabled = True
        Exit Sub
    End If
    cmdCancel.PerformClick()      'clean up most course information
    cmdInsert.Enabled = False    'disable the Insert button
End Sub

```

Figure 5.45. The code modifications for the Insert button event procedure.

Now you can run the project to test inserting data using the stored procedure. From Figure 5.45, it can be seen that there is no difference between calling a query function and calling a stored procedure to perform the data insertion. That is true for this data action because the stored procedure is a function or a collection of functions to perform some special functionality or functionalities. By using the TableAdapter Query Configuration Wizard, we cannot create a stored procedure that can be used to perform multiple data actions on multiple different data tables since each TableAdapter can only access the associated data table. But to insert data into the database using the runtime object method, which we will discuss in Part II, one stored procedure can access multiple different data tables and perform multiple different data manipulation operations.

At this point, we have finished developing our sample project to insert data into the SQL Server database. The completed project SQLInsertWizard is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**.

Next, we will discuss how to insert data into the Oracle database using the Visual Basic.NET design tools and wizards.

## 5.4 INSERT DATA INTO THE ORACLE DATABASE USING A SAMPLE PROJECT – ORACLEINSERTWIZARD

Because of the similarity between the SQL Server database and the Oracle database, all the coding we developed in the last section can be used to access the

Oracle database to perform the data insertion functionality. The only difference between these databases is the connection string when the Oracle database is connected to the Visual Basic.NET applications. In order to save space and time, we will not duplicate that code in this section. Refer to Section 4.2.2.2 in Chapter 4 and Appendix E for more detailed information on how to add and connect the Oracle database with your Visual Basic.NET applications using the Design Tools and Wizards as well as the associated coding. Refer to Appendix F to get a clear picture on how to use the sample Oracle 10g XE database CSE\_DEPT. As long as this connection is set up, all coding jobs are identical to those we did for the SQL Server database in the last section, and you can use that code to access the Oracle database to perform the different data actions. A complete data insertion project named OracleInsertWizard Project is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**.

## PART II DATA INSERTION WITH RUNTIME OBJECTS

To insert data into the database using the runtime object method is a flexible and professional way to perform data insertion in the Visual Basic.NET environment. Compared with the method we discussed in Part I, in which Visual Basic.NET Design Tools and Wizards are utilized to insert data into the database, the runtime object method provides a sophisticated way to do this job more efficiently and conveniently even though 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 and 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 are either before the project runs or after the project runs. In other words, you cannot modify that query function while the project runs.

To overcome those shortcomings, we will discuss how to insert data using the runtime object method in this part.

Basically, you need to use the TableAdapter to perform data actions in the database if you develop your coding using the Visual Basic.NET Design Tools and Wizards in the design time. But you should use the DataAdapter to perform those data manipulations if you develop your project using the runtime object method.

### 5.5 THE RUNTIME OBJECT METHOD

We have provided a very detailed introduction to and discussion about the runtime object method in Section 4.16 in Chapter 4. Refer to that section for more detailed information about this method. For your convenience, here we highlight some important points and the general methodology of this method as well as some key notes about using this method to perform data actions in the databases.

As you know, ADO.NET provides different classes to help users develop professional data-driven applications by using the different methods to perform specific

data actions such as 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 newly 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 uses the so-called DataSetDataAdapter 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 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, you can call this DataAdapter's Update() method. This method will perform an InsertCommand based on your modifications. The DeleteCommand will be executed if you delete rows from the table in the DataSet and call this Update() method. This method is relatively simple since you do not need to call specific methods such as 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 provide any further 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 and 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 SQLInsertRTOBJECT, AccInsertRTOBJECT, and OracleInsertRTOBJECT 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 SQLInsertRTOBJECT project first, and then illustrate the coding differences between these databases by using the real code for the other two sample projects.

Now let's first develop the sample project SQLInsertRTOBJECT to insert data into the SQL Server database using the runtime object method. Recall that in Sections

4.18.4–4.18.8 in Chapter 4, 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, we divide this discussion into two sections:

1. Insert data into the Faculty table from the Faculty form window using the runtime object method.
2. Insert data into the Course table from the Course form using the runtime stored procedure method.

In order to avoid duplicate coding, we will modify the existing project SQLSelectRTOBJECT that we developed in Chapter 4 to create the new project SQLInsertRTOBJECT used in this section.

## **5.6 INSERT DATA INTO THE SQL SERVER DATABASE USING THE RUNTIME OBJECT METHOD**

Open Windows Explorer and create a new folder, **Chapter 5**. Then open your Internet browser and locate the folder **VBProjects\Chapter 4** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). Copy the project SQLSelectRTOBJECT to the new folder C:\Chapter 5. Change the name of the project from SQLSelectRTOBJECT to SQLInsertRTOBJECT. Double-click on SQLInsertRTOBJECT.vbproj to open this project.

Perform the following modifications on the opened project to get our desired project:

- Go to Project|SQLInsertRTOBJECT Properties to open the project property window. Change the Assembly name from SQLSelectRTOBJECT to SQLInsertRTOBJECT and the Root namespace from SQLSelectRTOBJECT to SQLInsertRTOBJECT.
- Click the Assembly Information button to open the Assembly Information dialog box, and change the Title and the Product to SQLInsertRTOBJECT. Click OK to close this dialog box.

Go to File|Save All to save these modifications. Now we are ready to develop our graphical user interfaces based on our new project SQLInsertRTOBJECT.

### **5.6.1 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 need to add one more Windows form as the user interface for this new project.

#### **5.6.1.1 Add a Data Insertion Form Window – Insert Faculty Form**

The functionality of this Insert Faculty form is as follows: As the project runs, after the user has finished the login process and selected Faculty Information from the Select form, the Faculty form window will be displayed. When the user clicks the

Insert button, the Insert Faculty form window will show up. This form allows users to insert data into the Faculty data table in the database using the runtime object method. The form also allows users to enter all pieces of information into the appropriate text boxes for the newly inserted faculty member. By clicking the Insert button, a new record about a faculty member is inserted into the database. However, if the user wants to reenter those pieces of information before finishing this insertion, the Cancel button can be used and all information entered will be erased. The Back button is used to allow users to return to the Faculty form to perform the validation to confirm that the data insertion was successful.

Go to the File|Project|Add Windows Form menu item to open the Add New Item dialog box. Keep the default template, Windows Form, selected and enter Insert Faculty Form.vb into the Name box as the name for this new form. Then click the Add button to add this form into our project.

To save time and space, we can copy all controls of this Faculty form from the project SQLInsertWizard Project we developed in this chapter. Open that project, which is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**, and open the Insert Faculty Form window. Go to Edit|Select All to select all controls on that form window, go to the Edit|Copy menu item to copy those items. Now open our newly created project SQLInsertRTOBJECT and our new Insert Faculty Form window, enlarge it to an appropriate size, and go to the Edit|Paste menu item to paste those controls into this form. One important issue to be noted is that the project SQLInsertWizard Project is developed using the Visual Basic.NET design tools and wizards, so some objects related to those design tools and wizards such as the Data BindingSource will be added to this form as you paste those controls. Because we don't need those objects in this runtime object method, delete all of them from the new Insert Faculty Form window. To do that, right-click the FacultyBindingSource from the bottom of this form window and select the Delete item from the popup menu to remove it.

In addition to removing the components related to the design tools and wizards, you need to perform the following modifications to this form:

- Remove the ComboMethod combo box control from this form since we only use one method, the ExecuteNonQuery method of the Command class, to perform this runtime data insertion.
- Remove the Select button from this form since we will not perform the data validation until we click the Back button to return to the Faculty form window. In other words, the data validation is performed in the Faculty form.
- Make sure that the following properties of the form are set up:
  - AcceptButton: cmdInsert (set Insert button as default button)
  - StartPosition: CenterScreen (set the form in the center)

Your finished form window, Insert Faculty Form, is shown in Figure 5.46.

Detailed descriptions of the functionality of each control on this form can be found in Section 5.2.3 in this chapter. Simply speaking, the Faculty Photo check box is used to control both the Photo Name and the Photo Location text boxes to allow users to select the newly inserted faculty member's photo. The Faculty ID text box is a key text box since the Insert button will be enabled if this text box's

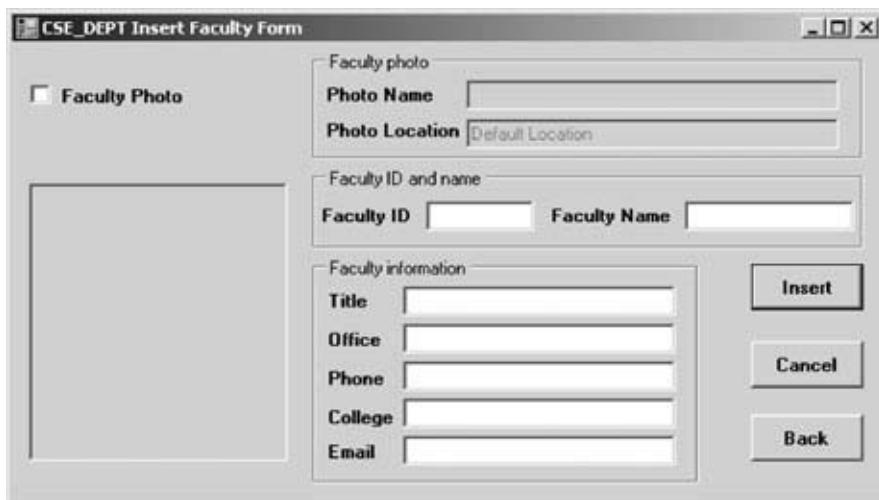


Figure 5.46. The modified Insert Faculty Form window.

content is changed, which means that a new faculty record will be inserted. The Cancel button is to allow users to clean up contents of all text boxes (except the Faculty ID) to reenter the faculty information. A new faculty record will be inserted into the database when the Insert button is clicked.

Our GUI design is done. Now let's develop the code for this form.

### 5.6.1.2 Develop the Code to Insert Data into the Faculty Table

The coding for this data insertion is divided into three steps: data validation before data insertion, data insertion using the runtime object method, and data validation after data insertion. The purpose of the first step is to confirm that all inserted data stored in each associated text box is complete and valid. In other words, all text boxes 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 for the first step now.

#### 5.6.1.2.1 Validate Data Before the Data Insertion and Startup Coding

First, let's take care of the startup coding. The startup coding includes the coding for the `Form_Load()` event procedure, global and form-level variable declarations, and coding for the Cancel and Back buttons. Open the code window of the Insert Faculty Form window and enter the code shown in Figure 5.47.

Let's look at this piece of coding to see how it works.

- The namespaces of the SQL Server database components are imported at the beginning of this form since we need to use components or classes related to the SQL Server database such as the Data Provider and Data Table.
- Two global variables are declared here. Both global variables, `FacultyName` and `InsertFacultyFlag`, will be used by the Faculty form to validate the data insertion later. As you know, to perform the data query in the Faculty form,

```

(InsertFacultyForm Events) ▾ Load ▾
A Imports System.Data
Imports System.Data.SqlClient
B Public Class InsertFacultyForm
C     Public FacultyName As String
    Public InsertFacultyFlag As Boolean
    Private FacultyBox(4) As TextBox
    Private FacultyInfo(7) As String
D     Private Sub InsertFacultyForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
E         If LogInForm.sqlConnection.State <> ConnectionState.Open Then
            LogInForm.sqlConnection.Open()
        End If
        InsertFacultyFlag = True
    End Sub
F     Private Sub cmdBack_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdBack.Click
G         Me.Hide()
    End Sub
    Private Sub cmdCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdCancel.Click
        txtName.Text = String.Empty
        txtTitle.Text = String.Empty
        txtOffice.Text = String.Empty
        txtPhone.Text = String.Empty
        txtCollege.Text = String.Empty
        txtEmail.Text = String.Empty
        txtPhotoName.Focus()
    End Sub

```

**Figure 5.47.** The startup coding.

the faculty name is used as the criterion, so we need to keep it as a global variable and access this faculty name defined in the Insert Faculty Form from the Faculty form later when a data validation is executed. The purpose of the InsertFacultyFlag is to indicate whether the Insert Faculty Form has been called to add a new faculty record or not. This variable is very important to decide whether a valid faculty photo has been found and displayed in the Faculty form when a data validation is performed. The accessing mode Public is used for these two global variables' declarations.

- C. Two form-level variables are created here, too. The first variable, FacultyBox, is a text box array and is used to map to five text boxes that store the faculty information in the Insert Faculty Form window. The second variable, FacultyInfo, is a string array that is mapped to all seven text boxes that hold the seven data columns of the Faculty data table, and it is used for the data validation before the data insertion.
- D. When the project runs and the Insert Faculty Form window is opened, the Form.Load event procedure is executed. First, we need to check whether a valid connection between our Visual Basic.NET project and the database exists. The data queries such as INSERT, UPDATE, and DELETE are different from the data query SELECT. To perform a data query including a SELECT statement, one can use the Fill() method to populate a data table. The point is that when the Fill() method is executed, first it checks whether a valid connection between the application and the database exists. If it does, this method executes the data query using that connection. But if no valid

connection exists, the Fill() method will first open a connection and then perform the data populating job. The difference is that the data queries including INSERT, DELETE, and UPDATE statements do not have a functionality to open a new connection if no valid connection is available. We need to create a new connection if no valid connection is available to make sure that the data insertion job can be continued.

- E. The global variable InsertFacultyFlag is set to indicate that the Insert Faculty Form window is called.
- F. The coding for the Back button event procedure is easy: one method, Me.Hide(), is executed to hide the Insert Faculty Form window. Hiding a window is different from closing a window; after the former method is executed, the form window disappears from the screen but is still in the memory. But after the latter method is executed, the form window is removed from the memory and from the project. After the Insert Faculty Form window is hidden, the Faculty form will be displayed to allow us to perform the data validation. Finally, the Insert Faculty Form window will be closed by executing a Me.Close() method in the Faculty form code window.
- G. The coding for the Cancel button event procedure is also easy. The class method String.Empty is assigned to each text box's Text property to clean them up. Note that this cleaning job does not include the Faculty ID text box. The reason for that is because after one new record is inserted into the database, the Insert button will be disabled to avoid multiple insertions of the same data. The data is identified by the Faculty ID that is a primary key in the Faculty table. In order to insert a different new record that has a different Faculty ID, we need to enable the Insert button to allow users to do that. In other words, as long as the content of the Faculty ID text box is changed, it means that a new, different record needs to be inserted and therefore the Insert button should be enabled. A cleaning of the Faculty ID text box is equivalent to changing its content, so in order to avoid enabling the Insert button mistakenly, we will not clean up its content in this Cancel button event procedure.

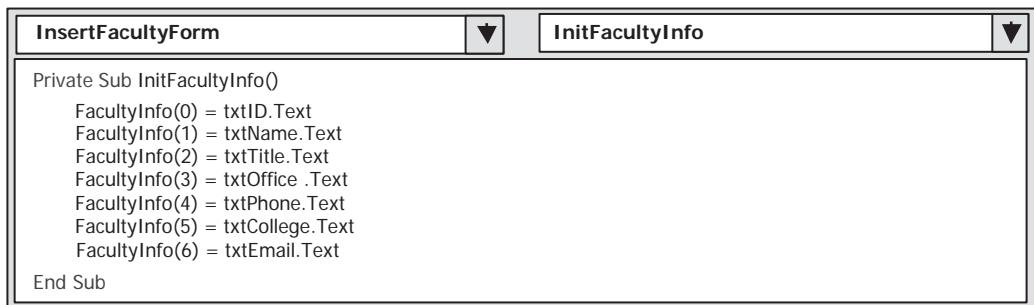
Now let's do our coding for the data validation before the data insertion.

This data validation can be performed by calling one subroutine and one function. The subroutine is named InitFacultyInfo() and is used to set up a mapping relationship between each item in the string array FacultyInfo and each text box. The function is named CheckFacultyInfo() and is used to scan and check all text boxes to make sure that none of them is empty.

Open the code window of the Insert Faculty Form, and enter the code shown in Figure 5.48 to create a user-defined subroutine InitFacultyInfo().

The FacultyInfo is a zero-based string array, and it starts its index from 0. All seven text boxes related to faculty information are assigned to this array. In this way, it is easier for us to scan and check each text box later to make sure that none of them is empty.

Now open the code window of the Insert Faculty Form, and enter the code shown in Figure 5.49 to create a user-defined function CheckFacultyInfo().



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "InsertFacultyForm" and the tab bar has "InitFacultyInfo". The code in the editor is:

```

Private Sub InitFacultyInfo()
    FacultyInfo(0) = txtID.Text
    FacultyInfo(1) = txtName.Text
    FacultyInfo(2) = txtTitle.Text
    FacultyInfo(3) = txtOffice.Text
    FacultyInfo(4) = txtPhone.Text
    FacultyInfo(5) = txtCollege.Text
    FacultyInfo(6) = txtEmail.Text
End Sub

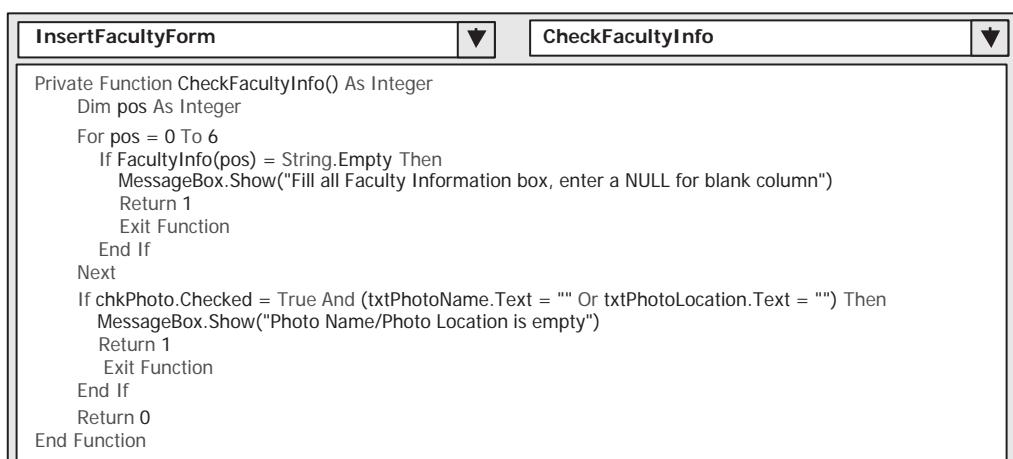
```

**Figure 5.48.** The coding for the InitFacultyInfo subroutine.

The functionality of this code is as follows:

- A For loop is used to scan each text box in the FacultyInfo string array to check whether any of them is empty. A message will be displayed if this situation occurs, and the function is exited to allow the user to fill all text boxes.
- If the user checks the Faculty Photo check box, which means that the user wants to include a faculty photo with this new data insertion, then he or she needs to check both text boxes' contents, Photo Name and Photo Location, to make sure that the necessary photo information is contained in those two text boxes. If either of them is empty, a message will be displayed to indicate this situation, and the function is exited to allow the user to enter the information the associated text box. If no text box is empty, the function returns a zero to indicate that this validation is successful.

Now let's develop the code for the Insert button event procedure to call the subroutine and the function to perform the data validation before the data insertion. Open the Insert button event procedure by double-clicking the Insert button from the form window of the Insert Faculty Form, and enter the code shown in Figure 5.50 into this event procedure.



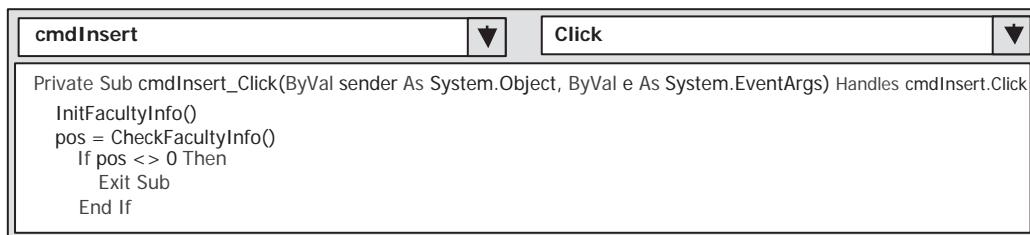
The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "InsertFacultyForm" and the tab bar has "CheckFacultyInfo". The code in the editor is:

```

Private Function CheckFacultyInfo() As Integer
    Dim pos As Integer
    For pos = 0 To 6
        If FacultyInfo(pos) = String.Empty Then
            MessageBox.Show("Fill all Faculty Information box, enter a NULL for blank column")
            Return 1
            Exit Function
        End If
    Next
    If chkPhoto.Checked = True And (txtPhotoName.Text = "" Or txtPhotoLocation.Text = "") Then
        MessageBox.Show("Photo Name/Photo Location is empty")
        Return 1
        Exit Function
    End If
    Return 0
End Function

```

**Figure 5.49.** The coding for the function CheckFacultyInfo().



The screenshot shows a Windows application's code editor window. At the top, there are two dropdown menus: 'cmdInsert' on the left and 'Click' on the right. Below the menu bar, the code is displayed:

```
Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    InitFacultyInfo()
    pos = CheckFacultyInfo()
    If pos <> 0 Then
        Exit Sub
    End If
```

**Figure 5.50.** The first coding for the Insert button event procedure.

The functionality of this coding is straightforward and easy to understand. First, the user-defined subroutine `InitFacultyInfo()` is called to set up the mapping relationship between each item in the string array `FacultyInfo` and each text box. Then the user-defined function `CheckFacultyInfo()` is executed to check and make sure that no text box is empty. If any text box is empty, the function returns a nonzero value and the procedure is exited to allow the user to enter information in the associated text boxes until all of them are filled with the desired information.

At this point, we have completed the coding for the data validation before the data insertion and the startup coding. Now let's do our coding for the data insertion.

#### 5.6.1.2.2 Insert Data into the Faculty Table

The main coding job is performed in the Insert button event procedure. We have already developed some code for the beginning of this procedure in the last section. Now let's complete this coding.

First, let's develop the coding in the Faculty form to trigger the Insert Faculty Form window. To insert new records into the faculty table in the database, the Insert Faculty Form window should be utilized, and this form will show up as soon as the user clicks the Insert button in the Faculty form window. To display the Insert Faculty Form window from the Faculty form, we need to perform the following three steps:

1. Create a form-level instance variable for the Insert Faculty Form class since we need to open that form window when the user clicks the Insert button.
2. Develop the code to display the Insert Faculty Form window when the Insert button is clicked.
3. Develop the code to close the Insert Faculty Form window when the user clicks the Back button on the Faculty form.

Now let's first complete the coding for the Faculty form. Open the code window of the Faculty form and create a form-level instance of the Insert Faculty Form window class by typing the following code in the Form's General Declaration section:

---

**Private** InsertFaculty **As New** InsertFacultyForm

---

```

cmdInsert Click
Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    InsertFaculty.Show()
End Sub

Private Sub cmdBack_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdBack.Click
    InsertFaculty.Close()
    Me.Close()
End Sub

```

**Figure 5.51.** The coding for the Insert and Back button event procedures.

Then enter the code shown in Figure 5.51 into two event procedures in the Faculty form, the Insert button event procedure and the Back button event procedure.

The Show() method of the Insert Faculty Form instance is called to display that form when the user clicks the Insert button from the Faculty form window. Similarly, the Close() method is executed to close that form when the user clicks the Back button. The code with a gray background was developed earlier.

Now let's do the coding for the Insert Faculty Form window, that is, for the Insert button event procedure in that form. Open that event procedure and enter the code shown in Figure 5.52 into the event procedure.

```

cmdInsert Click
Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim cmdString As String = "INSERT INTO Faculty (faculty_id, name, office, phone, college, title, email) " & _
        "VALUES (@faculty_id,@name,@office,@phone,@college,@title,@email)"
    Dim FacultyDataAdapter As New SqlDataAdapter
    Dim sqlCommand As New SqlCommand
    Dim pos, intInsert As Integer
    FacultyName = txtName.Text
    sqlCommand.Connection = LogInForm.sqlConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    InsertParameters(sqlCommand)
    FacultyDataAdapter.InsertCommand = sqlCommand
    intInsert = FacultyDataAdapter.InsertCommand.ExecuteNonQuery()
    intInsert = sqlCommand.ExecuteNonQuery()
    If intInsert = 0 Then
        MessageBox.Show("The data insertion is failed")
        Exit Sub
    End If
    cmdCancel.PerformClick()
    cmdInsert.Enabled = False
End Sub

```

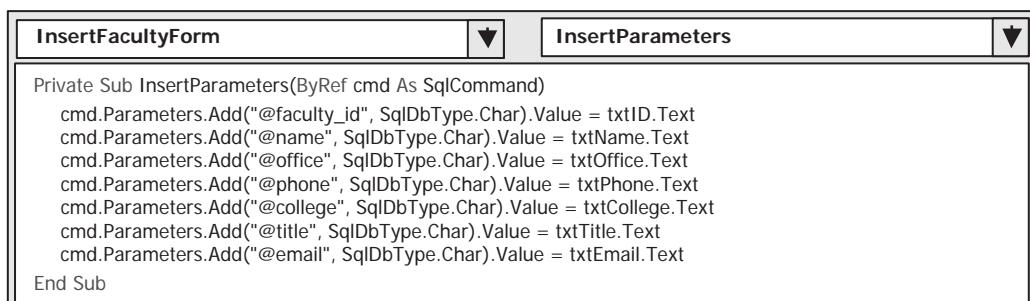
'reserve faculty name for validation

'clean up all faculty information  
'disable the Insert button

**Figure 5.52.** The coding for the Insert button event procedure.

The code we developed before for this event procedure is indicated with a gray background. Let's look at this piece of code to see how it works.

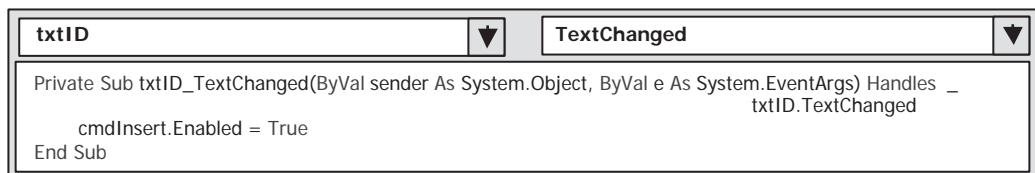
- A. The SQL INSERT statement is declared first, and this insertion query contains seven parameters followed the command VALUES. Each parameter is prefixed by an @ symbol since this is required by the SQL Server database.
- B. The data components used to perform the data insertion are declared here, which include the SqlDataAdapter and SqlCommand. Two local integer variables, pos and intInsert, are also declared in this part. The pos is used to hold the value returned the calling of the function CheckFacultyInfo(), and the intInsert is used to hold the value returned by executing the ExecuteNonQuery() method of the Command class.
- C. Since the faculty name text box will be cleaned up after the data insertion, we need to reserve it to perform the data validation later in the Faculty form.
- D. The Command instance is initialized with the Connection, CommandType, and CommandText properties of the Command class.
- E. Another user-defined subroutine, InsertParameters(), is called to fill parameters to the Parameters collection of the Command instance. Figure 5.53 shows the detailed coding for this subroutine later.
- F. 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.
- G. The ExecuteNonQuery() method returns an integer as the feedback to indicate whether this calling is successful or not. The value of this returned integer equals the number of newly inserted records in 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 is displayed and the procedure is exited if this situation occurs.
- H. A cleaning command is issued to clean up the contents of all text boxes that contain the newly inserted faculty information, except the Faculty ID.
- I. The Insert button is disabled after this data insertion to avoid multiple insertions of the same data. This button will be enabled again when the content of the Faculty ID text box is changed, which means that a new record is ready to be inserted into the Faculty table.



The screenshot shows a Microsoft Word document window titled "InsertFacultyForm". The title bar has two dropdown arrows on the right. The main content area displays the following VBA code:

```
Private Sub InsertParameters(ByRef cmd As SqlCommand)
    cmd.Parameters.Add("@faculty_id", SqlDbType.Char).Value = txtID.Text
    cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text
    cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text
    cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text
    cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text
    cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text
    cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text
End Sub
```

Figure 5.53. The coding for the user-defined subroutine InsertParameters().



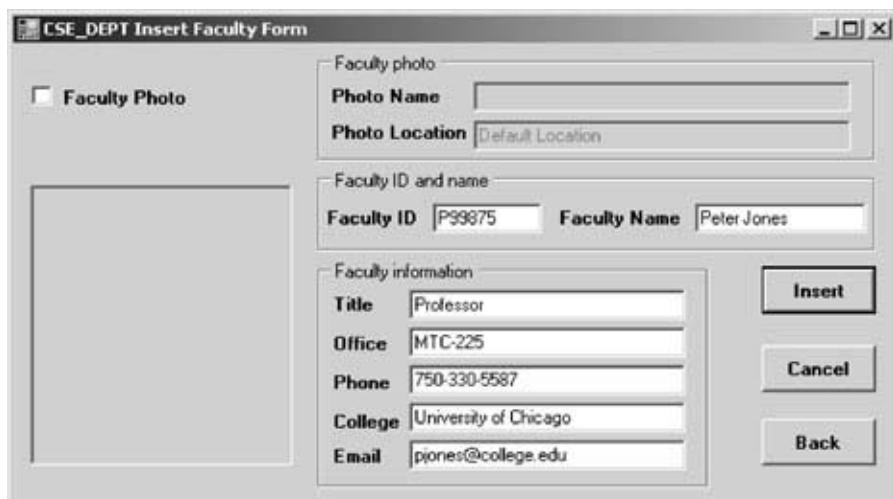
**Figure 5.54.** The coding for the TextChanged event procedure.

The detailed coding for the user-defined subroutine `InsertParameters()` is shown in Figure 5.53.

This piece of coding is easy. Each piece of faculty-related information stored in the associated text box is assigned to each matched parameter by using the `Add()` method. One point to be noted is that the @ symbol must be prefixed to each parameter since this is required by the SQL Server database.

Another coding job is for the Faculty ID text box, that is, for the `TextChanged` event procedure of the Faculty ID text box. As mentioned, in order to avoid multiple insertions of the same data, the `Insert` button will be disabled after one record is inserted into the database. This `Insert` button will be enabled again when the content of the Faculty ID text box is changed, which means that a different new record is ready to be inserted into the database. The coding for that event procedure is shown in Figure 5.54.

Now let's first test the coding we have developed to run the project to insert a new record into the Faculty data table in the database. Start the project by clicking the Start Debugging button, enter the correct username and password to the `LogIn` form, and then select on Faculty Information from the Selection form window to open the Faculty form. Click the `Insert` button on the Faculty form to open the Insert Faculty Form window, which is shown in Figure 5.55.



**Figure 5.55.** The running status of the Insert Faculty Form window.

Enter the following information the associated text box in as the information for a new faculty member:

- |                         |                       |
|-------------------------|-----------------------|
| ■ P99875                | Faculty ID text box   |
| ■ Peter Jones           | Faculty Name text box |
| ■ Professor             | Title text box        |
| ■ MTC-225               | Office text box       |
| ■ 750-330-5587          | Phone text box        |
| ■ University of Chicago | College text box      |
| ■ pjones@college.edu    | Email text box        |

Keep the Faculty Photo check box unchecked. Your finished information should match that shown in Figure 5.55.

Click the Insert button to insert this new record into the Faculty data table in the database. Immediately all text boxes, except the Faculty ID, become empty, and the Insert button is disabled. Click the Back button to return to the Faculty form window, and then click the Back and Exit buttons on the Faculty and the Selection forms to terminate the project. Our data insertion is successful!

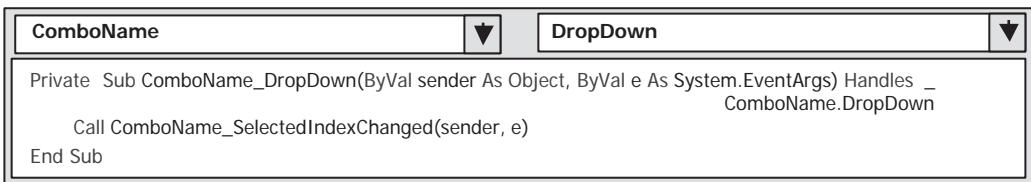
Wait a moment. How do you know this data insertion is successful? We have to find a way to confirm this. Well, an easy way to confirm it is that the newly inserted data can be read back from the database and displayed in a form if this insertion is successful. We have a very good candidate form to perform this data validation, the Faculty form, because we developed the code for this form to perform the data query before. Now let's add some code to this Faculty form to perform the data validation for this newly inserted data.

#### 5.6.1.2.3 Validate Data After the Data Insertion

To validate the newly inserted faculty record, the Faculty form window is used. The functionality 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 code we developed for the Select button event procedure in Chapter 4 to perform this data query, and we also need to add the following two additional steps to complete the data validation:

1. The newly inserted faculty name, which is stored in a global variable, FacultyName, and is inserted to the database from the Insert Faculty Form, should be added to the ComboName combo box on the Faculty form to allow us to select it to make a query to retrieve the inserted data from the database if this insertion is successful. This addition is necessary and should be performed when the drop-down arrow of the ComboName control is clicked to select a faculty name to execute a new query.
2. The global variable InsertFacultyFlag should be used to determine whether a faculty photo is included in the data insertion. The faculty photo should be loaded and displayed if this variable is True, which means that a faculty photo is involved in the data insertion.

To perform step 1, there is a little trick. As the user clicks the drop-down arrow of the ComboName combo box to select a faculty name a ComboName.DropDown



```

ComboName  DropDown 

```

```

Private Sub ComboName_DropDown(ByVal sender As Object, ByVal e As System.EventArgs) Handles _
    ComboName.DropDown
    Call ComboName_SelectedIndexChanged(sender, e)
End Sub

```

**Figure 5.56.** The coding for the ComboName DropDown event procedure.

event is created that will trigger the ComboName\_DropDown event procedure. The problem is that a new item can be added to the combo box only when a combo box's SelectedIndexChanged event occurs, and no new item can be added to the combo box with any other events. That is tricky! To solve this problem, we need to call this SelectedIndexChanged event procedure from the DropDown event procedure to add the newly inserted faculty name into the combo box.

Now let's develop our code for these two steps to complete this data validation.

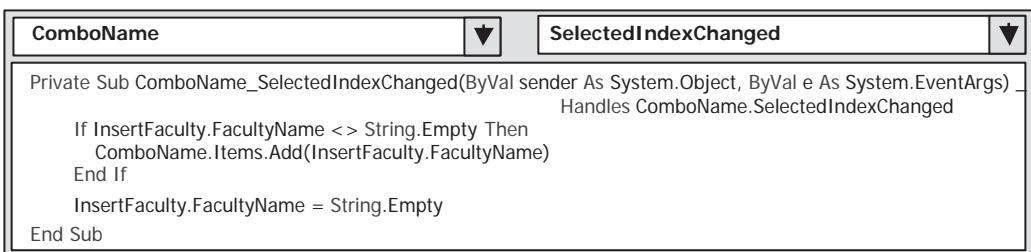
Open the code window of the Faculty form. Then select the ComboName item from the Class Name combo box, select the DropDown item from the Method Name combo box to open the ComboName\_DropDown event procedure, and enter the code shown in Figure 5.56 into this event procedure.

The coding is simple. It calls the SelectedIndexChanged event procedure of the ComboName combo box, which is shown in Figure 5.57, to add the newly inserted faculty name into this combo box.

The functionality of the coding for the SelectedIndexChanged event procedure is as follows: First, it checks the global variable FacultyName to confirm that it is not an empty string, and then the Items.Add() method of the combo box is used to add the faculty name into the ComboName combo box. Finally, an empty string is assigned to the FacultyName variable to avoid having that faculty name added into the combo box again when the user clicks the drop-down arrow of the combo box to select another faculty name in the future. If the FacultyName contains an empty string, it means either that the Insert Faculty Form has not been opened and no new faculty information has been inserted into the database or that the faculty name has already been added to the combo box. In either case, no action is needed.

Since the global variable FacultyName is defined in the Insert Faculty Form, that form's instance name, InsertFaculty, should be prefixed to that global variable.

Now let's handle the coding for step 2.



```

ComboName  SelectedIndexChanged 

```

```

Private Sub ComboName_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _
    ComboName.SelectedIndexChanged
    If InsertFaculty.FacultyName <> String.Empty Then
        ComboName.Items.Add(InsertFaculty.FacultyName)
    End If
    InsertFaculty.FacultyName = String.Empty
End Sub

```

**Figure 5.57.** The coding for the SelectedIndexChanged event procedure.

The functionality of this piece of coding is to determine whether a faculty photo is included in this data insertion. The user can select whether or not to include a faculty photo the data insertion. To include a photo, the user should check the Faculty Photo check box in the Insert Faculty Form when performing the data insertion. When the data validation is performed, we need to check this check box to find out if a photo is involved for that data insertion. Also, two text boxes, Photo Name and Photo Location, should be checked to see whether a valid photo name and a valid photo location have been entered. Multiple possible selections may be made by the user for this faculty photo issue, and they are summarized as follows:

1. If the user wants to include a faculty photo with the data insertion, the chkPhoto check box in the Insert Faculty Form window should be checked, and both the Photo Name and Photo Location text boxes should be filled with the desired faculty photo name and the location. If the chkPhoto check box is unchecked, this means that the user does not want to include faculty photo with the data insertion.
2. It is possible that the user just wants to perform a normal data query in this Faculty form without inserting data into the database. In that case, the global variable InsertFacultyFlag should be False.
3. If situation 2 occurs and no valid faculty photo file can be found, a warning message should be displayed to indicate this situation.

In order to combine this piece of code with the code we developed before, we will insert this piece of code into the user-defined subroutine ShowFaculty() that we developed before. The newly inserted code is shown in Figure 5.58.

The code we developed before is indicated with a gray background. Let's have a close look at the new code to see how it works.

- A. The MessageBox is removed from the old coding since we do not want to display this warning message until we complete our photo detection issue.
- B. If the FacultyImage contains an empty string, two possibilities exist: First, new faculty data has been added to the database using the Insert Faculty Form window and the user wants to include a faculty photo for that data insertion (step **C**). The faculty photo's name and location are provided in two text boxes, txtPhotoName and txtPhotoLocation, respectively. Second (step **F**), another two possibilities exist for this situation: either new data has been added to the database and the user does not want to include a faculty photo for that data insertion (in that case, the global variable InsertFacultyFlag should be True – step **G**), or the user just wants to perform a normal data query in the Faculty form without data insertion (in that case, the global variable InsertFacultyFlag should be False – step **H**). Let's handle these cases one by one.
- C. If the user wants to include a faculty photo for data insertion, we need to find the photo location and the photo name and assign them to the FacultyImage variable that will be used later to display the faculty photo.
- D. If the content of the Photo Location text box is not equal to “Default Location”, it means that the user has provided a special location for the photo. In that case, we need to pick up that location and combine it with the content of

```

FacultyForm
ShowFaculty

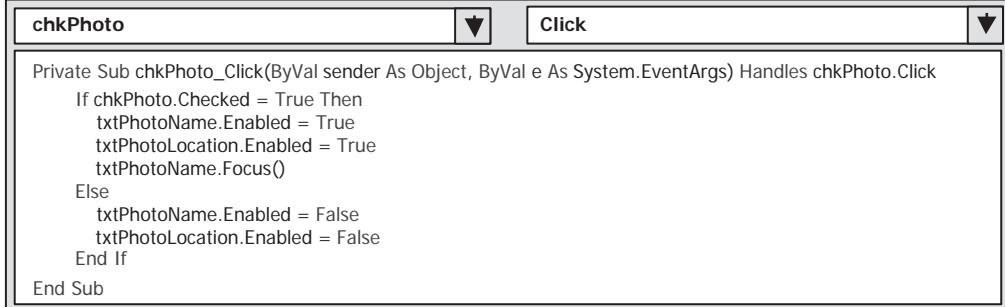
Private Sub ShowFaculty(ByVal fName As String)
    Dim FacultyImage As String
    Select Case fName
        Case "Ying Bai"
            FacultyImage = "Bai.jpg"
        Case "Satish Bhalla"
            FacultyImage = "Satish.jpg"
        Case "Black Anderson"
            FacultyImage = "Anderson.jpg"
        Case "Steve Johnson"
            FacultyImage = "Johnson.jpg"
        Case "Jenney King"
            FacultyImage = "King.jpg"
        Case "Alice Brown"
            FacultyImage = "Brown.jpg"
        Case "Debby Angles"
            FacultyImage = "Angles.jpg"
        Case "Jeff Henry"
            FacultyImage = "Henry.jpg"
        Case Else
            FacultyImage = ""
    End Select
    If FacultyImage = "" Then
        If InsertFaculty.chkPhoto.Checked = True Then
            If InsertFaculty.txtPhotoLocation.Text <> "Default Location" Then
                FacultyImage = InsertFaculty.txtPhotoLocation.Text & "\& InsertFaculty.txtPhotoName.Text
            Else
                FacultyImage = InsertFaculty.txtPhotoName.Text
            End If
        Else
            If InsertFaculty.InsertFacultyFlag = True Then
                PhotoBox.Image = System.Drawing.Image.FromFile("Default.jpg")
                Exit Sub
            Else
                MessageBox.Show("No match faculty image found!")
                Exit Sub
            End If
        End If
        PhotoBox.Image = System.Drawing.Image.FromFile(FacultyImage)
    End Sub

```

**Figure 5.58.** The added coding for the ShowFaculty subroutine.

the Photo Name text box to build a full name of the photo. Then we assign this full name of the photo to the variable FacultyImage to make it ready to be displayed later.

- E. If the user selects the default location for the photo, we only need to pick up that name from the Photo Name text box and assign it to the variable FacultyImage to make it ready to be displayed later.
- F. If the chkPhoto check box in the Insert Faculty Form window is unchecked, two possibilities exist: First, new data has been inserted into the database and the user does not want to include faculty photo for that data insertion. If this happens, the global variable InsertFacultyFlag should be set to True (step **G**). Second, no new data insertion has been performed and the Insert Faculty Form is not touched, which means that the user just wants to perform a normal data query using the Faculty form (step **H**).



The screenshot shows the Visual Studio code editor with the following code:

```

chkPhoto Click
A Private Sub chkPhoto_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles chkPhoto.Click
B     If chkPhoto.Checked = True Then
         txtPhotoName.Enabled = True
         txtPhotoLocation.Enabled = True
         txtPhotoName.Focus()
     Else
         txtPhotoName.Enabled = False
         txtPhotoLocation.Enabled = False
     End If
End Sub

```

**Figure 5.59.** The coding for the Faculty Photo check box.

- G. For the first possibility, we do not need to display any faculty photo for that data insertion. But here, in order to make our project neat, we use a beautiful photo as our default photo and display it for any data insertion that does not include photo to be displayed.
- H. For the second possibility, no correct photo file can be found for the normal data query. A warning message is displayed to indicate this error, and the subroutine is exited.
- I. Finally, the faculty photo is displayed by calling the system method.

Now we have finished all the coding for this data insertion job in our Faculty form. But before we can test this functionality, we need to add one more functionality to display a faculty photo with the data insertion.

### 5.6.2 Insert a New Faculty Photo

Three controls on the Insert Faculty Form window are used to help users insert a new faculty photo: the Faculty Photo check box and the Photo Name and Photo Location text boxes. These two text boxes will not be enabled for users to enter information for the faculty photo until the Faculty Photo check box is checked. This functionality can be realized by using the code shown in Figure 5.59. Open this event procedure by first opening the code window of the Insert Faculty Form and then selecting the chkPhoto item from the Class Name combo box and selecting the Click item from the Method Name combo box. Enter the code shown in Figure 5.59 into this event procedure.

The functionality of this piece of coding is as follows:

- A. When this check box is clicked, we first need to see if it check box is checked. If it is, which means that the user wants to include a faculty photo for the data insertion, Photo Name and Photo Location text boxes are enabled and a focus is set to the Photo Name text box to allow users to enter the photo information into those two text boxes.
- B. If this check box is unchecked, which means that no photo will be involved in the data insertion, then both text boxes are disabled.

At this point, we have finished all coding for both the data insertion and the data validation after data insertion. Let's run the project to test the coding. Since we want to add a faculty photo for this data insertion, make sure that the desired

faculty photo file has already been saved into the desired location. For this test, we want to display a faculty photo named Mhamed.jpg, which we have stored in our default folder, C:\Chapter 5\SQLInsertRTOBJECT\bin\Debug.

Now start the project. After the project begins to run, enter a suitable username and password, such as jhenry and test, in the LogIn form, and then select the Faculty Information item from the Selection form to open the Faculty form window. Click the Insert button to open the Insert Faculty Form window, and enter the following information into this form as new faculty information:

- |                       |                       |
|-----------------------|-----------------------|
| ■ A99875              | Faculty ID text box   |
| ■ Ali Mhamed          | Faculty Name text box |
| ■ Associate Professor | Title text box        |
| ■ MTC-235             | Office text box       |
| ■ 750-330-3387        | Phone text box        |
| ■ University of Main  | College text box      |
| ■ amhamed@college.edu | Email text box        |

Then check the Faculty Photo check box and enter Mhamed.jpg into the Photo Name text box as the photo file name. Keep Default Location unchanged in the Photo Location text box. Your finished information for this new faculty member is shown in Figure 5.60.

Now click the Insert button to insert this new faculty record into the database. Immediately the Insert button is disabled after this insertion. Click the Back button to return to the Faculty form to perform the data validation.

In the opened Faculty form window, click the drop-down arrow of the ComboBox combo box and you will find that the newly inserted faculty name, Ali Mhamed, is in there. Click that name to select it, and then click the Select button to try to read back that newly inserted record from the database and display it in this Faculty form window.



Figure 5.60. The running status of the Insert Faculty Form window.



Figure 5.61. An example of the data validation result.

Immediately you will find that all information in that newly inserted faculty record, including the faculty photo, is displayed in the associated boxes, shown in Figure 5.61. Our data insertion is successful because the newly inserted data has been retrieved from the database and successfully displayed in this Faculty form.

One potential bug exists in this data validation. Each time you enter a new piece of faculty information into the database using this Insert Faculty Form window, the faculty name must not be identical. Some readers may argue with me on this: different faculty members are identified by the faculty ID, not by name, and the faculty ID is the primary key in the Faculty table. One can enter multiple identical faculty names into the database as long as their IDs are different. Yes, that is true. But the issue is that in this application, we use the faculty name, not the faculty ID, as the criterion to perform this SELECT query. This means that the query criterion is based on the faculty name, and multiple records will be returned if multiple faculty members have the same name even they have different faculty IDs.

Recall that when we developed the coding for the user-defined subroutine FillFacultyTable() in Section 4.18.4 in Chapter 4, the returned faculty information was stored in a label array named FacultyLabel(), and the size of this array is 5 since we only need to retrieve five columns from the Faculty table. The potential bug is that this FacultyLabel array can only hold five pieces of information returned from the Faculty table. A runtime error may occur if multiple records that have the same faculty name but different faculty IDs are returned and assigned to that FacultyLabel array because this label array cannot hold more than five pieces of information!. For the detailed coding of this issue, refer to the inner For Each... loop in Figure 4.106 in Chapter 4.

Two solutions to this bug are as follows: always use different faculty names for the new data when performing the data insertion into the Faculty table in the database, or enlarge the FacultyLabel array to allow it to hold more records.

The completed project SQLInsertRTOBJECT is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) the folder **DBProjects\Chapter 5**.

Basically, there are no significant differences between inserting data into the SQL Server, Microsoft Access, and Oracle databases. The only difference is in the query strings, such as the Connection string and the SELECT query string used in the LogIn form, the SELECT query string used in the Faculty form, and the INSERT query string and Parameter strings used in the Insert Faculty Form. All other coding is identical. In Section 5.6.3, we will show those differences and discuss how to insert data into the Microsoft Access database, and in Section 5.6.4 we will discuss how to insert data into the Oracle database.



One possible problem with testing your project by inserting more data into the Faculty table is that too many records are added to the database.

To remove those unused records, you can open the Faculty table from the SQL Server Management Studio Express and delete those records from the table.

### **5.6.3 Insert Data into the Microsoft Access Database Using Runtime Objects**

As we mentioned at the end of the last section, the only difference in data insertion between the different databases is in the query strings used in the different form windows. All other parts of the coding are identical, without modifications. So we can use most of the code in the project SQLInsertRTOBJ 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 SQLInsertRTOBJ. Open Windows Explorer and create a new folder, such as **Chapter 5**, and then copy the project SQLInsertRTOBJ from the folder **DBProjects\Chapter 5** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) to our new folder, **Chapter 5**. Change the name of this project to AccessInsertRTOBJ, which includes the following files:

- AccessInsertRTOBJ.sln
- AccessInsertRTOBJ.vbproj
- AccessInsertRTOBJ.vbproj.user
- AccessInsertRTOBJ.exe
- AccessInsertRTOBJ.pdb
- AccessInsertRTOBJ.vshost.exe
- AccessInsertRTOBJ.xml

To rename the last four files, you need to use the Project Properties window. 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 database. We can modify the project SQLInsertRTOBJ to get our new project, AccessInsertRTOBJ, to perform our data insertion job using the runtime object method. Basically, we need to modify the following items:

1. Imports commands – 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 string – select the correct faculty information
5. To other forms – change the connection object

Modification items 2 and 3 are included in the LogIn form window with the LogIn data table, and modification item 4 is located in the Faculty form with the Faculty data table in the Microsoft Access database. Let's do these modifications one by one now.

#### 5.6.3.1 Modify the Imports Commands

First, let's open the code window of the LogIn form by clicking the View Code button from the Solution Explorer window. In 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 finish the modification for the Imports commands in the LogIn form window. Make the same modification to the rest of the form windows:

- Faculty Form
- Course Form
- Insert Faculty Form

Since we will not use the Student form for our data insertion, no modification should be made to it. Now let's modify the Connection string for the LogIn form.

#### 5.6.3.2 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, we first need to open the Form\_Load event procedure of the LogIn form since the connection string is defined in there.

Open the code window of the LogIn form if it has not been opened. Select the item (LogInFormEvents) from the Class Name combo box and select the item Load from the Method Name combo box to open the Form\_Load event procedure. Change the name and the content of the Connection string, which is shown in Figure 5.62.

Let's have a close look at these modifications.

- A. The modifications to the Imports commands are shown here.
- B. 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.
- C. Change the name and the content of the connection string as shown in here.

Make the modifications shown in Figure 5.62 steps **D** to **K**. All modifications are shown in bold.

```

(LogInForm Events) ▾ Load ▾
Imports System.Data
Imports System.Data.OleDb
Public Class LogInForm
    Public accConnection As OleDbConnection
    Private Sub LogInForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        Dim accString As String = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
            "Data Source=C:\database\CSE_DEPT.mdb;"
        accConnection = New OleDbConnection(accString)
        If accConnection.State = ConnectionState.Open Then
            accConnection.Close()
        End If
        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
    End Sub
End Sub

```

Figure 5.62. The modifications to the Connection string.

Go to File|Save All to save those modifications. Next, let's modify the login query strings in the LogIn form.

### 5.6.3.3 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 modify only one method, the TableAdapter method. Open this event procedure by double-clicking the TabLogIn button from the LogIn form window, and make the modifications shown in Figure 5.63 to this event procedure.

Let's take a look at these modifications.

1. Most parts of this query string work with the Microsoft Access database, so the only modifications are the LIKE symbols used in the WHERE clause. Change these two LIKE symbols to equal symbols before the two parameters @Param1 and @Param2, respectively. This is the syntax used in the Microsoft Access database.
2. In steps **B** to **D**, change the prefix for all OleDb classes used in this event procedure from Sql to OleDb. All modifications are shown in bold.
3. In step **E** and steps **G** to **R**, change the prefix for all OleDb objects used in this event procedure from sql to acc. All modifications are shown in bold.
4. In step **F**, change the prefix for both OleDb classes and objects from sql to acc, and from Sql to OleDb, respectively.

You can perform similar modifications to the code in the ReadLogIn and the Cancel button event procedures.

```

Private Sub TabLogIn_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TabLogIn.Click
    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)"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim paramUserName As New OleDbParameter
    Dim paramPassWord As New OleDbParameter
    Dim LogInTableAdapter As New OleDbDataAdapter
    Dim accDataTable As New DataTable
    Dim accCommand As New OleDbCommand
    Dim selForm As New SelectionForm

    paramUserName.ParameterName = "@Param1"
    paramUserName.Value = txtUserName.Text
    paramPassWord.ParameterName = "@Param2"
    paramPassWord.Value = txtPassWord.Text
    accCommand.Connection = accConnection
    accCommand.CommandType = CommandType.Text
    accCommand.CommandText = cmdString
    accCommand.Parameters.Add(paramUserName)
    accCommand.Parameters.Add(paramPassWord)
    LogInTableAdapter.SelectCommand = accCommand
    LogInTableAdapter.Fill(accDataTable)
    If accDataTable.Rows.Count > 0 Then
        selForm.Show()
        Me.Hide()
    Else
        MessageBox.Show("No matched username/password found!")
    End If
    accDataTable.Dispose()
    accDataTable = Nothing
    accCommand.Dispose()
    accCommand = Nothing
    LogInTableAdapter.Dispose()
    LogInTableAdapter = Nothing
End Sub

```

Figure 5.63. The modifications to the LogIn query string.

Now let's go to the Faculty form to modify the Faculty table query string.

#### 5.6.3.4 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 5.6.3.1. Then open the Form\_Load event procedure and change the Connection object, which is located in the first line, from `sqlConnection` to `accConnection`, as shown below:

---

**If LogInForm.accConnection.State <> ConnectionState.Open Then**

---

Now open the Select button event procedure by double-clicking this button from the Faculty form window, and perform the modifications shown in Figure 5.64.

Let's have a look at these modifications.

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString1 As String = "SELECT office, phone, college, title, email FROM Faculty "
    Dim cmdString2 As String = "WHERE name=@facultyName"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim paramFacultyName As New OleDbParameter
    Dim FacultyDataAdapter As New OleDbDataAdapter
    Dim accCommand As New OleDbCommand
    Dim accDataReader As OleDbDataReader
    Dim accDataTable As New DataTable

    paramFacultyName.ParameterName = "@facultyName"
    paramFacultyName.Value = ComboName.Text
    accCommand.Connection = LogInForm.accConnection
    accCommand.CommandType = CommandType.Text
    accCommand.CommandText = cmdString
    accCommand.Parameters.Add(paramFacultyName)
    Call ShowFaculty(ComboName.Text)

    If ComboMethod.Text = "TableAdapter Method" Then
        FacultyDataAdapter.SelectCommand = accCommand
        FacultyDataAdapter.Fill(accDataTable)
        If accDataTable.Rows.Count > 0 Then
            Call FillFacultyTable(accDataTable)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        accDataTable.Dispose()
        accDataTable = Nothing
        FacultyDataAdapter.Dispose()
        FacultyDataAdapter = Nothing
    Else
        accDataReader = accCommand.ExecuteReader
        If accDataReader.HasRows = True Then
            Call FillFacultyReader(accDataReader)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        accDataReader.Close()
        accDataReader = Nothing
    End If
    accCommand.Dispose()
    accCommand = Nothing
End Sub

```

**Figure 5.64.** Modifications to the Faculty query string.

- 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 the LIKE, which is inside the cmdString2 and located before the dynamic parameter @facultyName, to the equal symbol “=” since this is the requirement of the Microsoft Access database.
- B. Change the prefix of all OleDb data classes from Sql to OleDb in steps **B** and **C**. All modifications have been indicated in bold.
- D. In steps **D** and **E**, change the prefix of all OleDb data classes and objects from Sql to OleDb, and from sql to acc. All modifications have been indicated in bold.
- F. Change the prefix of all OleDb objects from sql to acc in steps **F** to **W**.

Another modification is for the user-defined subroutine FillFacultyReader(). The data type of the argument should be changed from SqlDataReader to OleDbDataReader.

Before we can run the project to insert data into the database, we need to finish the modifications to the other forms. Basically, the only modification is to change the connection object in all other forms to match the Microsoft Access database connection.

#### 5.6.3.5 Modifications to Other Forms

The following four forms contain the connection object: Selection, Course, Student, and Insert Faculty Form. In this project we only need to use the Selection, Course, and Insert Faculty forms, so we only need to modify the connection object for those three forms. Open the code window of the Course and the Insert Faculty forms; to be precise, open the Form\_Load event procedure of those forms, and change the connection object name from sqlConnection to accConnection. Also open the Exit button event procedure on the Selection form to change the connection object there, too. Your finished modification for this connection object should match the one shown below:

---

```
If LogInForm.accConnection.State <> ConnectionState.Open Then
```

---

Besides the Form\_Load event procedure, the following event procedures also contain this connection object:

- The Select button event procedure and the Course list box's SelectedIndexChanged event procedure in the Course form
- The user-defined subroutine BuildCommand() in the Student form
- The Insert button event procedure in the Insert Faculty Form

Open those event procedures and subroutines and change the connection object. Since we will not use the Student form for this project, you can temporarily comment out the first line in the Student form, which contains the connection object in the BuildCommand() subroutine.

Also perform the following modifications to the associated items in all forms used in this project:

- Change the prefix for all OleDb classes and objects from Sql to OleDb, and from sql to acc, respectively.
- Change the data type for all parameters from SqlDbType to OleDbType, from SqlCommand to OleDbCommand, and from SqlDataReader to OleDbDataReader.

The final modification is for the user-defined subroutine InsertParameters() in the Insert Faculty Form. Change the data type of the inserting parameters from SqlDbType to OleDbType.

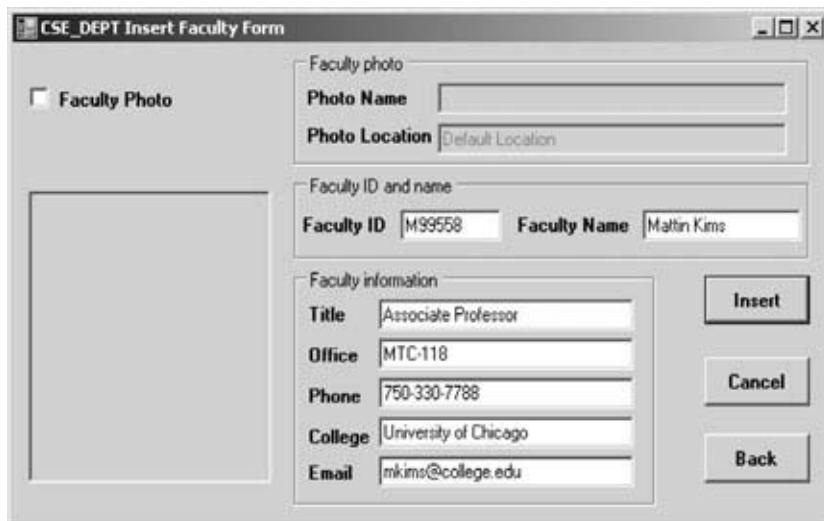


Figure 5.65. The running status of the Insert Faculty Form window.

Now let's run the project to test our data insertion functionality. Click the Start Debugging button to run the project, and enter a suitable username and password, such as jhenry and test, in the LogIn form window, and then select the Faculty Information item from the Selection form to open the Faculty form. Click the Insert button to open the Insert Faculty Form window, which is shown in Figure 5.65.

Enter the following data into the associated text boxes as a new faculty record:

- |                         |                       |
|-------------------------|-----------------------|
| ■ M99558                | Faculty ID text box   |
| ■ Mattin Kims           | Faculty Name text box |
| ■ Associate Professor   | Title text box        |
| ■ MTC-118               | Office text box       |
| ■ 750-330-7788          | Phone text box        |
| ■ University of Chicago | College text box      |
| ■ mkims@college.edu     | Email text box        |

Keep the Faculty Photo check box unchecked since we do not want to include a photo for this faculty record. Your finished window should match the one shown in Figure 5.65.

Click the Insert button to insert this new faculty record into the Faculty table in the database. Immediately the Insert button is disabled after this new data is inserted into the database. Now click the Back button to return to the Faculty form to validate this data insertion.

Click the drop-down arrow of the ComboName combo box on the Faculty form and you will see that the newly inserted faculty name, Mattin Kims, is in this box. Click it to select this faculty member, and then click the Select button to try to retrieve this newly inserted data from the database and display it in this form. Immediately you will see that all information about this new faculty member appears in this form, as shown in Figure 5.66.



Figure 5.66. The data validation process.

This evidence shows that our data insertion into the Microsoft Access database is successful! Click the Back and then the Exit buttons to close the project.

You can remove the 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 delete the records.

The completed project AccessInsertRTOBJECT is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**.

#### 5.6.4 Insert Data into the Oracle Database Using Runtime Objects

Similarly, as we did in the last section for the Microsoft Access database, we can modify the SQLInsertRTOBJECT project to make it work for the Oracle database.

First, open Windows Explorer and create a new folder, such as **Chapter 5**, and then copy the project SQLInsertRTOBJECT from [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5** to our new folder in **Chapter 5**. Change the name of this project to OracleInsertRTOBJECT, which includes the following files:

- OracleInsertRTOBJECT.sln
- OracleInsertRTOBJECT.vbproj
- OracleInsertRTOBJECT.vbproj.user
- OracleInsertRTOBJECT.exe
- OracleInsertRTOBJECT.pdb
- OracleInsertRTOBJECT.vshost.exe
- OracleInsertRTOBJECT.xml

To rename the last four files, you need to use the Project Properties window. 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 Oracle database. We will modify the project SQLInsertRTOBJECT to get a new project OracleInsertRTOBJECT to perform our data insertion job using the runtime object method. Basically, we need to modify the following items:

1. Imports commands – all Oracle data components are defined here
2. Database Connection string – connect it 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. Other forms – change the connection object

Items 2 and 3 are included in the LogIn form window with the LogIn data table, and item 4 is located in the Faculty form with the Faculty data table in the Oracle database. Let's do these modifications one by one now.

#### **5.6.4.1 Add the Reference and Modify the Imports Commands**

Unlike Microsoft Access and SQL Server databases, Visual Basic.NET does not set the Oracle namespace as a default data namespace for the database programming. So we first need to add this namespace as a reference to our new project. To do that, go to the Solution Explorer window, right-click the project, and select the Add Reference item from the popup menu to open the Add Reference dialog box. Browse down the list until you find the item System.Data.OracleClient, select it, and click the OK button to add this reference into the project.

To confirm this addition, click the Show All Files button from the Solution Explorer window, and then expand the Reference item to find the reference we just added to the project.

Next, let's open the code window of the LogIn form by clicking the View Code button from the Solution Explorer window. In the opened code window, move your cursor to the top and modify the two Imports commands to

---

```
Imports System.Data
Imports System.Data.OracleClient
```

---

In this way, we finish the modification for the Imports commands in the LogIn form window. Make the same modifications to these form windows:

- Faculty Form
- Course Form
- Insert Faculty Form

Since we will not use the Student form for our data insertion, no modification should be made to it. Now let's modify the Connection string for the LogIn form.

#### **5.6.4.2 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

```

(LogInForm Events) ▾ Load ▾
Imports System.Data
Imports System.Data.OracleClient
Public Class LogInForm
    Public oraConnection As OracleConnection
    Private Sub LogInForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        Dim oraString As String = "Data Source=XE;" + _
            "User ID=system;" + "Password=reback"
        oraConnection = New OracleConnection(oraString)
        If oraConnection.State = ConnectionState.Open Then
            oraConnection.Close()
        End If
        Try
            oraConnection.Open()
        Catch OracleExceptionErr As OracleException
            MessageBox.Show(OracleExceptionErr.Message, "Oracle Error")
        Catch InvalidOperationExceptionErr As InvalidOperationException
            MessageBox.Show(InvalidOperationExceptionErr.Message, "Oracle Error")
        End Try
        If oraConnection.State <> ConnectionState.Open Then
            MessageBox.Show("Database connection is Failed")
            Exit Sub
        End If
    End Sub

```

Figure 5.67. Modifications to the Connection string in LogIn form.

modification, we first need to open the Form\_Load event procedure of the LogIn form since the connection string is defined in there.

Open the code window of the LogIn form if it has not been opened. Select the item LogInEvents from the Class Name combo box, and select the item Load from the Method Name combo box to open this Form\_Load event procedure. Change the name and the content of the Connection string, as shown in Figure 5.67.

Let's have a close look at these modifications.

1. The modifications to the Imports commands are shown in step **A**.
2. 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.
3. Change the name and the content of the connection string as shown in step **C**.

Also make the modifications shown in Figure 5.67 to the items from steps **D** to **K**. All modifications have been indicated in bold.

Go to File|Save All to save the modifications. Next, let's modify the login query strings in the LogIn form.

#### 5.6.4.3 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, the TableAdapter method. Open this event procedure by double-clicking the TabLogIn button from

```

Private Sub TabLogIn_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles TabLogIn.Click
    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"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim paramUserName As New OracleParameter
    Dim paramPassWord As New OracleParameter
    Dim LogInTableAdapter As New OracleDataAdapter
    Dim oraDataTable As New DataTable
    Dim oraCommand As New OracleCommand
    Dim selForm As New SelectionForm
    paramUserName.ParameterName = "Param1"
    paramUserName.Value = txtUserName.Text
    paramPassWord.ParameterName = "Param2"
    paramPassWord.Value = txtPassWord.Text
    oraCommand.Connection = oraConnection
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add(paramUserName)
    oraCommand.Parameters.Add(paramPassWord)
    LogInTableAdapter.SelectCommand = oraCommand
    LogInTableAdapter.Fill(oraDataTable)
    If oraDataTable.Rows.Count > 0 Then
        selForm.Show()
        Me.Hide()
    Else
        MessageBox.Show("No matched username/password found!")
    End If
    oraDataTable.Dispose()
    oraDataTable = Nothing
    oraCommand.Dispose()
    oraCommand = Nothing
    LogInTableAdapter.Dispose()
    LogInTableAdapter = Nothing
End Sub

```

**Figure 5.68.** Modifications to the login query string in LogIn form.

the LogIn form window, and make the modifications shown in Figure 5.68 to this event procedure.

Make the following modifications:

1. Most parts of the query string in step **A** work with the Oracle database, and the only modification is the LIKE symbol used in the WHERE clause. Change the two LIKE symbols to the assignment operator “:=” before the two parameters @Param1 and @Param2, respectively. This is the syntax used in the Oracle database.
2. In steps **B** to **D**, change the prefix for all Oracle classes used in this event procedure from Sql to Oracle. All modifications have been indicated in bold.
3. In step **E** and steps **G** to **R**, change the prefix for all Oracle objects used in this event procedure from sql to ora. All modifications are indicated in bold.
4. In step **F**, change the prefix for both Oracle classes and objects from sql to ora, and from Sql to Oracle, respectively.

You can perform similar modifications to the code in the ReadLogIn and the Cancel button event procedures.

Now let's go to the Faculty form to modify the Faculty table query string.

```

cmdSelect_Click
Click

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString1 As String = "SELECT office, phone, college, title, email FROM Faculty "
    Dim cmdString2 As String = "WHERE name=:facultyName"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim paramFacultyName As New OracleParameter
    Dim FacultyDataAdapter As New OracleDataAdapter
    Dim oraCommand As New OracleCommand
    Dim oraDataReader As OracleDataReader
    Dim oraDataTable As New DataTable

    paramFacultyName.ParameterName = "facultyName"
    paramFacultyName.Value = ComboName.Text
    oraCommand.Connection = LogInForm.oraConnection
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add(paramFacultyName)
    Call ShowFaculty(ComboName.Text)

    If ComboMethod.Text = "TableAdapter Method" Then
        FacultyDataAdapter.SelectCommand = oraCommand
        FacultyDataAdapter.Fill(oraDataTable)
        If oraDataTable.Rows.Count > 0 Then
            Call FillFacultyTable(oraDataTable)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        oraDataTable.Dispose()
        oraDataTable = Nothing
        FacultyDataAdapter.Dispose()
        FacultyDataAdapter = Nothing
    Else
        oraDataReader = oraCommand.ExecuteReader
        If oraDataReader.HasRows = True Then
            Call FillFacultyReader(oraDataReader)
        Else
            MessageBox.Show("No matched faculty found!")
        End If
        oraDataReader.Close()
        oraDataReader = Nothing
    End If
    oraCommand.Dispose()
    oraCommand = Nothing
End Sub

```

Figure 5.69. Modifications to the query string in the Faculty form.

#### 5.6.4.4 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 5.6.4.1. Then open the Form\_Load event procedure and change the Connection object, which is located in the first line, from the sqlConnection to the oraConnection, as shown below:

---

If LogInForm.oraConnection.State <> ConnectionState.Open Then

---

Now open the Select button event procedure by double-clicking this button from the Faculty form window, and perform the modifications shown in Figure 5.69.

Let's have a look at these modifications.

1. The first modification is to the query string in step **A**. As in the last section, most parts of this query string work for the Oracle database, and the only modification is to change the LIKE, which is inside the cmdString2 and located before the dynamic parameter @facultyName, to the Oracle assignment operator “:=” since this is a requirement of the Oracle database. Also remove the @ symbol before the parameter facultyName.
2. Change the prefix of all Oracle data classes from Sql to Oracle in steps **B** and **C**. All modifications have been indicated in bold.
3. In steps **D** and **E**, change the prefix of all Oracle data classes and objects from Sql to Oracle, and from sql to ora. All modifications have been indicated in bold.
4. Change the prefix of all Oracle objects from sql to ora in step **F** and steps **G** to **W**.
5. In step **Z**, remove the @ symbol before the dynamic parameter facultyName. This is the syntax for Oracle database operations.

Another modification is for the user-defined subroutine FillFacultyReader(). The data type of the argument should be changed from SqlDataReader to OracleDataReader.

Before we can run the project to insert data into the database, we need to finish the rest of the modifications to other forms. Basically, the modification for all other forms is to change the connection object to match to the Oracle database connection.

#### **5.6.4.5 Modifications to Other Forms**

The following four forms contain this connection object: Selection, Course, Student, and Insert Faculty Form. In this project we only need to use the Selection, Course, and Insert Faculty forms, so we need to modify the connection object only for those three forms. Open the code window of the Course and Insert Faculty forms, that is, open the Form\_Load event procedure of those forms, and change the connection object name from SqlConnection to oraConnection. Also open the Exit button event procedure of the Selection form to change the connection object there, too. Your modified connection object should match the one shown below:

---

If LogInForm.oraConnection.State <> ConnectionState.Open Then

---

Besides the Form\_Load event procedure, the following event procedures also contain this connection object:

- The Select button event procedure and the Course list box SelectedIndexChanged event procedure in the Course form
- The user-defined subroutine BuildCommand() in the Student form
- The Insert button event procedure in the Insert Faculty Form

Open those event procedures and subroutines and change the connection object. Perform the following modifications to the event procedures or subroutines in the Course form.

The modification to the Select button event procedure in the Course form is to change the joined table query string. The current string used for the SQL Server database is a new version, but it cannot be recognized by the Oracle database. Change this string to

---

```
Dim cmdString1 As String = "SELECT Course.course_id, Course.course  
FROM Course, Faculty"  
Dim cmdString2 As String = "WHERE (Course.faculty_id = Faculty.  
faculty_id) AND (Faculty.name = :name)"
```

---

The modification to the query string cmdString2, which is located in the Course List box's SelectedIndexChanged event procedure, is to replace the LIKE @ symbol before the parameter courseid with the Oracle assignment operator "=:".

The modification to the user-defined subroutine FillCourseReader() is to replace the method GetSqlString() to GetOracleString().

Since we will not use the Student form for this project, you can temporarily comment out the first line in the Student form which contains the connection object in the BuildCommand() subroutine. Also, you can comment out the If... Then block in the Form\_Load event procedure in the Student form. You can remove the SP Form from this project if you like since this form will not be used for this data insertion.

Perform the following modifications to the associated items in all forms used in this project:

- Change the prefix for all data classes and objects from Sql to Oracle, and from sql to ora, respectively.
- Change the data type for all parameters from SqlDbType to OracleType, from SqlCommand to OracleCommand, and from SqlDataReader to OracleDataReader, respectively.

Modifications to the Insert Faculty Form include the following:

- Remove the @ symbol before each parameter, and change the data type of the inserting parameters from SqlDbType to OracleType in the user-defined subroutine InsertParameters().

Change the Insert query string to

---

```
Dim cmdString As String =  
"INSERT INTO Faculty (faculty_id, name, office, phone, college, title,  
email) " &  
"VALUES (:faculty_id,:name,:office,:phone,:college,:  
title,:email)"
```

---

Now we have finished the modifications to our new project OracleInsertRTOBJECT, and you can run the project to test the data insertion to the Oracle database. The completed project OracleInsertRTOBJECT is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**.

## 5.7 INSERT DATA INTO THE DATABASE USING STORED PROCEDURES

In this section, we discuss how to insert data into the database using stored procedures. We provided a very detailed introduction to stored procedures and illustrated how to use this method to perform data queries for the Student form and Student table in Section 4.18.8 in Chapter 4. Refer to that part to get more detailed information on the stored procedures.

We now use the Course form and Course table to illustrate how to insert a new course record based on a selected faculty member into the Course data table. 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 same thing for the Oracle database. Some readers may have noted that we spent a lot of time modifying the code in the Course form in the last project, OracleInsertRTOBJECT, 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.

### 5.7.1 Insert Data into the SQL Server Database Using Stored Procedures

To save time, we can modify the project SQLInsertRTOBJECT to create a new project named SQLInsertRTOBJECTSP and add one more form window to perform the data insertion using stored procedures. Recall that when we developed that project, an Insert button was added to the Course form window. We can use this button to trigger a new form to perform the data insertion job using the stored procedures. First, let's add one more form window into this new project. The name of this new form is Insert Course Form.

#### 5.7.1.1 Add an Insert Data Form Window – Insert Course Form

The functionality of this Insert Course form is as follows: As the project runs, after the user has finished the login process and selected the Course Information item from the Selection form, the Course form window will be displayed. When the user clicks the Insert button, the Insert Course Form window will show up. This form allows users to insert data into the Course data table in the database using stored procedures. The form also allows users to enter information into the appropriate text boxes for the newly inserted course. By clicking the Insert button, a new course record related to the selected faculty member is inserted into the database. However, if the user wants to reenter the pieces of information before finishing this insertion, the Cancel button can be used and all information entered will be erased. The Back button is used to return to the Course form to perform the validation to confirm that the data insertion was successful.

Go to the File|Project|Add Windows Form menu item to open the Add New Item dialog box. Keep the default template, Windows Form, selected, and enter Insert Course Form.vb into the Name box as the name for this new form. Then click the Add button to add this form into the project.

To save time, we can copy all controls of this Course form from the project SQLInsertWizard Project we developed in this chapter. Open that project, which is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**, and open the Insert Course Form window, then go to the Edit|Select All menu item to select all controls on that form window. Go to the Edit|Copy menu item to copy those items. Now open our project SQLInsertRTOBJECTSP and our new form Insert Course Form window, enlarge it to an appropriate size, and go to Edit|Paste to paste those controls into this form.

One important issue to note is that the project SQLInsertWizard Project is developed using the Visual Basic.NET design tools and wizards, so some objects related to those design tools and wizards such as the Data BindingSource will be added to this form as you paste those controls. Because we don't need those objects in this runtime object method, delete all of them from the new Insert Course Form window. To do that, right-click the CourseBindingSource at the bottom of this form window and select the Delete item from the popup menu to remove it.

In addition to removing the components related to the design tools and wizards, you need to perform the following modifications to this form:

- Remove the ComboMethod combo box control from this form since we only use one method, the ExecuteNonQuery method of the Command class, to execute the stored procedure to perform this data insertion.
- Remove the Select button from this form since we will not perform the data validation until we click the Back button to return to the Course form window. In other words, the data validation is performed in the Course form.
- Make sure that the following properties of the form are set up:
  - AcceptButton: cmdInsert (set Insert button as default button)
  - StartPosition: CenterScreen (set the form in the center)

Your finished form window, Insert Course Form, should match the one shown in Figure 5.70.

Detailed descriptions of the functionality of each control on this form can be found in Section 5.3.2. Simply speaking, the Faculty Name combo box is used to allow users to select the desired faculty member to insert a new course. Seven text boxes allow users to enter information for the new course to be inserted into the Course table. The Course ID text box is a key text box since the Insert button will be enabled if this text box's content is changed, which means that a new course will be inserted. The Cancel button allows users to clean up the contents of the text boxes (except the Course ID) to reenter the course information. A new course record will be inserted into the database when the Insert button is clicked. The Back button is used to return to the Course form to perform the data validation for the newly inserted course.

Our GUI design is done; next, we will develop the code for this form. But first let's take care of our stored procedures.

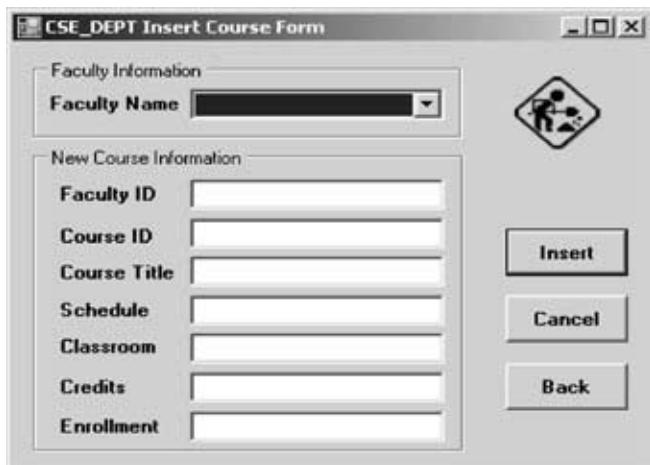


Figure 5.70. The finished Insert Course Form window.

### 5.7.1.2 Develop Stored Procedures for the SQL Server Database

Recall that when we built the sample database CSE\_DEPT in Chapter 2, there was no faculty name column in the Course table, and the only relationship existing between the Faculty and Course tables was the faculty\_id, which is a primary key in the Faculty table but a foreign key in the Course table. As the project runs and the Insert Course Form window is displayed, 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 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.

Instead of the stored procedure, another solution to avoid performing two queries is to use a joined table query to combine these two queries to complete a course query, as we did for the Course form in Section 4.18.6 in Chapter 4. But 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 procedure 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, click the plus symbol next to CSE\_DEPT database folder to connect to our database if this database was added to the Server Explorer before. Otherwise, you need to right-click on the Data Connections folder to add and connect to the database. Refer to Section 4.18.8.3 in Chapter 4 for detailed information on adding and connecting the database.

Right-click on the Stored Procedures folder and select the Add New Stored Procedure item to open the Add Procedure dialog box, and then enter the code shown in Figure 5.71 into this new procedure.

The functionality of this stored procedure is as follows:

```

ALTER PROCEDURE dbo.InsertFacultyCourse
    @FacultyName VARCHAR(50),
    @CourseID VARCHAR(50),
    @Course text,
    @Schedule text,
    @Classroom text,
    @Credit int,
    @Enroll int

AS
    DECLARE @FacultyID VARCHAR(50)
    SET @FacultyID = (SELECT faculty_id FROM Faculty
                      WHERE name LIKE @FacultyName)
    INSERT INTO Course VALUES (@CourseID,@Course,@Credit,@Classroom,@Schedule,@Enroll,@FacultyID)
    RETURN

```

Figure 5.71. The stored procedure dbo.InsertFacultyCourse.

- All input parameters are listed in this part. The @FacultyName is selected by the user from the ComboName combo box, and all other input parameters should be entered by the user in the associated text boxes in the Insert Course Form window.
- A local variable, @FacultyID, is declared and is used to hold the returned value from the execution of the first query to the Faculty table in step C.
- The first query is executed to pick up the matched faculty\_id from the Faculty table based on the first input parameter, @FacultyName.
- The second query is used to insert a new course record into the Course table. The last parameter in the VALUES parameter list is the @FacultyID, which is obtained from the first query.

The coding for this stored procedure is simple and easy to understand. One point you should note is the order of parameters in the VALUES parameter list. This order must be identical to the column order in the Course table. Otherwise, an error may be encountered when this stored procedure is saved.

Go to File|Save StoredProcedure1 to save this stored procedure. Now let's test this stored procedure in the Server Explorer environment to make sure that it works.

Right-click our new stored procedure dbo.InsertFacultyCourse from the Server Explorer window, and click the Execute item from the popup menu to open the Run Stored Procedure dialog box. Enter the input parameters into the associated box for a new course record. The finished parameters dialog box is shown in Figure 5.72.

Click the OK button to run this stored procedure. The running result is displayed in the Output window, which is shown in Figure 5.73.

To confirm this data insertion, open the Course table by first expanding the Tables folder in the Server Explorer window and then right-clicking the Course folder, and select the item Show Table Data. Browse to the last row of the table, and you will see that a new course, CSE-538: Advanced Robotics, has been inserted into this table. Our stored procedure is successful!

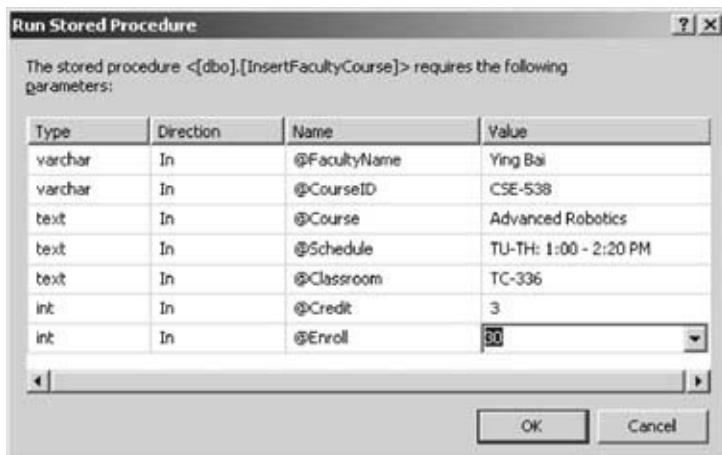


Figure 5.72. The Run Stored Procedure dialog box.

Next, we need to develop the code in Visual Basic.NET environment to call this stored procedure to insert a new course record into the database from our user interface.

### 5.7.1.3 Develop Code to Call Stored Procedures to Insert Data into the Course Table

The coding for this data insertion is divided into three steps: data validation before data insertion, data insertion using the stored procedure, and data validation after data insertion. The purpose of the first step is to confirm that the data in each text box is complete and valid. In other words, all text boxes 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 for the first step now.

#### 5.7.1.3.1 Validate Data Before the Data Insertion and Startup Coding

First, let's take care of the startup coding. The startup coding includes adding the Imports commands, the coding for the Form\_Load() event procedure, global and form-level variable declarations, and coding for the Cancel and Back buttons. Open

Output
<pre>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].</pre>

Figure 5.73. The running result of the stored procedure.

**(InsertCourseForm Events)** **Load**

```

Imports System.Data
Imports System.Data.SqlClient
Public Class InsertCourseForm
    Private CourseInfo(5) As String
    Private Sub InsertCourseForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.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
    End Sub
    Private Sub InitCourseInfo()
        CourseInfo(0) = txtFacultyID.Text
        CourseInfo(1) = txtCourseID.Text
        CourseInfo(2) = txtSchedule.Text
        CourseInfo(3) = txtClassRoom.Text
        CourseInfo(4) = txtCredits.Text
        CourseInfo(5) = txtEnroll.Text
    End Sub
    Private Function CheckCourseInfo() As Integer
        Dim pos As Integer
        For pos = 1 To 5
            If CourseInfo(pos) = String.Empty Then
                MessageBox.Show("Fill all Course Information box, enter a NULL for blank column")
                Return 1
                Exit Function
            End If
        Next
        Return 0
    End Function
    Private Sub cmdCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdCancel.Click
        txtFacultyID.Text = String.Empty
        txtCourse.Text = String.Empty
        txtSchedule.Text = String.Empty
        txtClassRoom.Text = String.Empty
        txtCredits.Text = String.Empty
        txtEnroll.Text = String.Empty
    End Sub
    Private Sub cmdBack_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdBack.Click
        Me.Hide()
    End Sub

```

Figure 5.74. The startup coding.

the code window of the Insert Course Form window and enter the code shown in Figure 5.74 into this code window.

The coding in the Form\_Load event procedure has been discussed in detail in the previous project. The purpose of this coding is to add all faculty names into the ComboName control to allow users to select one when performing the new course insertion.

Let's take a look at these pieces of code to see how they work:

- A string array, CourseInfo(), is created first, and this array is used to store all information related to the new course to be inserted into the database.

- B. The user-defined subroutine InitCourseInfo() is used to set up a one-to-one relationship between each item in the CourseInfo string array and each text box that contains a piece of new course information. In this way, it is easier to scan and check each text box to make sure that none of them is empty when the user-defined function CheckCourseInfo() is executed later.
- C. To check each text box, a For loop is utilized to scan the CourseInfo array. A warning message is displayed and the function returns a nonzero value to the calling procedure to indicate that this checking has failed if any text box (except the Faculty ID) is empty. Otherwise, a zero is returned to indicate that this checking is successful. A trick is that the For loop starts from 1, not 0, which means that this check does not include the Faculty ID text box. That is correct because at this moment we do not have any knowledge of this information.
- D. The Cancel button event procedure is used to clean up all text boxes' content, except the Course ID. The reason for this is that after a new course is inserted into the database, the Insert button is disabled to avoid multiple insertions of the same course. But this button will be enabled again as soon as the content of the Course ID text box is changed, which means that a different new course will be inserted. In order to keep this button from being enabled mistakenly, we do not want the Course ID text box to be cleaned or changed.
- E. The coding for the Back button event procedure is easy. A Me.Hide() method is used to hide the Insert Course Form window. This window should be closed when the user clicks the Back button from the Course form later.

Now let's do our coding 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 event procedure. Open the Insert button event procedure by double-clicking the Insert button on the Insert Course Form window, and enter the code shown in Figure 5.75 into this event procedure.

The functionality of this piece of coding is straightforward and easy to understand. First, the subroutine InitCourseInfo() is called to set up an one-to-one relationship between each item in the CourseInfo() array and each text box that stores a piece of course information. Next, the function CheckCourseInfo() is executed to

cmdInsert	▼	Click	▼
<pre>Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click     Dim pos As Integer     InitCourseInfo()     pos = CheckCourseInfo()     If pos &lt;&gt; 0 Then         Exit Sub     End If End Sub</pre>			

**Figure 5.75.** The first coding for the Insert button event procedure.

```

cmdInsert Click
Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim cmdString As String = "dbo.InsertFacultyCourse"
    Dim intInsert As Integer
    Dim sqlCommand As New SqlCommand
    InitCourseInfo()
    pos = CheckCourseInfo()
    If pos <> 0 Then
        Exit Sub
    End If
    sqlCommand.Connection = LogInForm.sqlConnection
    sqlCommand.CommandType = CommandType.StoredProcedure
    sqlCommand.CommandText = cmdString
    InsertParameters(sqlCommand)
    intInsert = sqlCommand.ExecuteNonQuery()
    sqlCommand.Dispose()
    sqlCommand = Nothing
    If intInsert = 0 Then
        MessageBox.Show("The data insertion is failed")
        Exit Sub
    End If
    cmdCancel.PerformClick()      'clean up all course information
    cmdInsert.Enabled = False    'disable the Insert button
End Sub

```

**Figure 5.76.** The modifications to the Insert button's event procedure.

make sure that the new course information is complete and valid, or in other words, that no text box is empty.

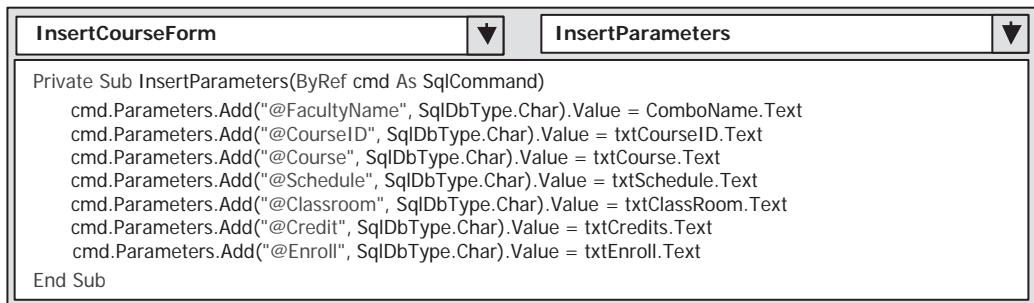
Now let's develop the code to call the stored procedure to perform the new course insertion.

#### 5.7.1.3.2 Develop Code to Call Stored Procedures

Open the Insert Course Form window by clicking the View Designer button from the Solution Explorer window, and then double-click the Insert button to open its event procedure. Add the code shown in Figure 5.76 into this event procedure. The code we developed in the last section has been highlighted with a gray background.

Let's take a look at the newly added code to see how it works.

- The query string is assigned with the name of the stored procedure we developed in Section 5.7.1.2. One of the most important points about calling stored procedures is that the query string must be exactly identical to the name of the stored procedure to be called. The Visual Basic.NET project would not find the stored procedure and a runtime error would be encountered if the query string did not match the name of the stored procedure.
- Some other components and variables used in this procedure are declared here. The local integer variable intInsert is used to hold the value returned by executing of the ExecuteNonQuery () method. The Command object is created here, too.
- The Command object is initialized with suitable components. Two important points to be noted are related to CommandType and CommandText. The former must be assigned with the property of StoredProcedure to indicate



```

Private Sub InsertParameters(ByRef cmd As SqlCommand)
    cmd.Parameters.Add("@FacultyName", SqlDbType.Char).Value = ComboName.Text
    cmd.Parameters.Add("@CourseID", SqlDbType.Char).Value = txtCourseID.Text
    cmd.Parameters.Add("@Course", SqlDbType.Char).Value = txtCourse.Text
    cmd.Parameters.Add("@Schedule", SqlDbType.Char).Value = txtSchedule.Text
    cmd.Parameters.Add("@Classroom", SqlDbType.Char).Value = txtClassRoom.Text
    cmd.Parameters.Add("@Credit", SqlDbType.Char).Value = txtCredits.Text
    cmd.Parameters.Add("@Enroll", SqlDbType.Char).Value = txtEnroll.Text
End Sub

```

**Figure 5.77.** The coding for the subroutine InsertParameters.

that the command type of this Command object is Stored Procedure and a stored procedure will be called when this Command is executed. The name of the stored procedure to be called must be assigned to the CommandText property of the Command object to indicate to the Visual Basic.NET where to find it.

- D. The user-defined subroutine InsertParameters(), whose detailed coding is shown in Figure 5.77, 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 the new data insertion.
- F. The Command object is cleaned up after the data insertion.
- G. The ExecuteNonQuery() method will return an integer to indicate whether this calling is successful or not. The returned value equals the number of rows or records that have been successfully added to 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. Finally, after the data insertion, the information stored in all text boxes, except the Course ID, is cleaned up to make it ready for the next data insertion. Also, the Insert button is disabled to avoid multiple insertions of the same data into the database.

The coding for the user-defined subroutine InsertParameters() is shown in Figure 5.77. The functionality of this subroutine is to assign each piece of information stored in each text box to the associated input parameter we defined in the stored procedure dbo.InsertFacultyCourse. One key point of this coding is that the name of each parameter, which is represented as a string and located at the first argument's position, must be identical to 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 to the input parameter's name @FacultyName in the input parameter list we defined at the beginning of the stored procedure dbo.InsertFacultyCourse. A runtime error would be encountered if the name of a parameter was not matched with the associated parameter's name in the stored procedure as the project runs. Refer to Figure 5.71 for a detailed list of all parameter names defined in the stored procedure.

Now we have finished the coding for this data insertion operation. Before we can run the project to test the data insertion, we need to figure out how to open and start the Insert Course Form to perform this new course insertion. There are two ways to open the Insert Course Form; one way is to directly start this form and insert data, and the other is to use the Course form to trigger this form. We prefer to use the second way since we need to use the Course form to perform the data validation after a new record is inserted in the Course table.

Let's make three modifications to the Course form to trigger the Insert Course Form.

- The first modification is to create a form-level object of the Insert Course Form class, and this object is used to start the Insert Course Form window when the Insert button in the Course form is clicked. To do that, add the following command under the class header:

---

**Private InsertCourse As New InsertCourseForm**

---

- The second modification is to add a command to close the Insert Course Form window when the Back button in the Course form is clicked. To do that, add the following command line into the Back button event procedure, cmdBack\_Click():

---

**InsertCourse.Close()**

---

- The third modification is to call the Show() method of the Insert Course Form object to display that form when the Insert button in the Course form is clicked. To do that, add the following command into the Insert button event procedure in the Course form, cmdInsert\_Click():

---

**InsertCourse.Show()**

---

Now let's run the project to test the new data insertion using the stored procedure. Click the Start Debugging button to start the project, enter a suitable user-name and password, such as jhenry and test, the LogIn form, and select the Course Information item from the Selection form to open the Course form window. Click the Insert button to open the Insert Course Form window to perform the new course insertion.

Enter the following data into the associated text boxes as the information for a new course. Leave the Faculty ID text box empty since we don't know that piece of information at this moment and this information will be retrieved by the stored procedure.

- |            |                        |
|------------|------------------------|
| ■ Ying Bai | Faculty Name combo box |
| ■ CSE-668  | Course ID text box     |

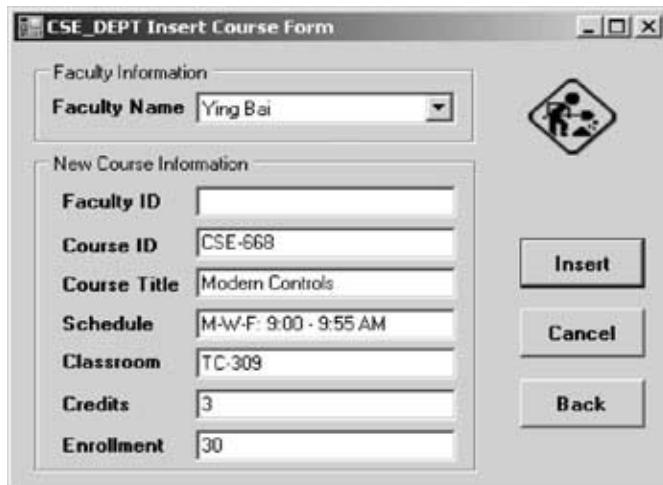


Figure 5.78. The running status of the Insert Course Form window.

- |                      |                       |
|----------------------|-----------------------|
| ■ Modern Controls    | Course Title text box |
| ■ M-W-F: 9:00–9:55AM | Schedule text box     |
| ■ TC-309             | Classroom text box    |
| ■ 3                  | Credits text box      |
| ■ 30                 | Enrollment text box   |

Your finished information window should match the one shown in Figure 5.78.

Click the Insert button to call the stored procedure to insert this new course record into the database. Immediately the Insert button is disabled after this data insertion. Was our data insertion successful? To answer this question, we need to perform the data validation in the next section.

#### 5.7.1.3.3 Validate Data After the Data Insertion

Now click the Back button to return to the Course form window to validate this data insertion.

Click the drop-down arrow on the ComboName combo box control and select Ying Bai from the list since we inserted a new course for this faculty member in the last section. Click the Select button to try to retrieve the newly inserted course record and display it in the Course form window. All courses taught by the selected faculty member are displayed in the Course List box. The last item is the course we just added to the Course table in the last section. Click that item and all information related to that new course is displayed in this form, as shown in Figure 5.79. This is the evidence that our data insertion using the stored procedure was successful!

A completed project SQLInsertRTOBJECTSP that includes the data insertion using the stored procedure can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**.

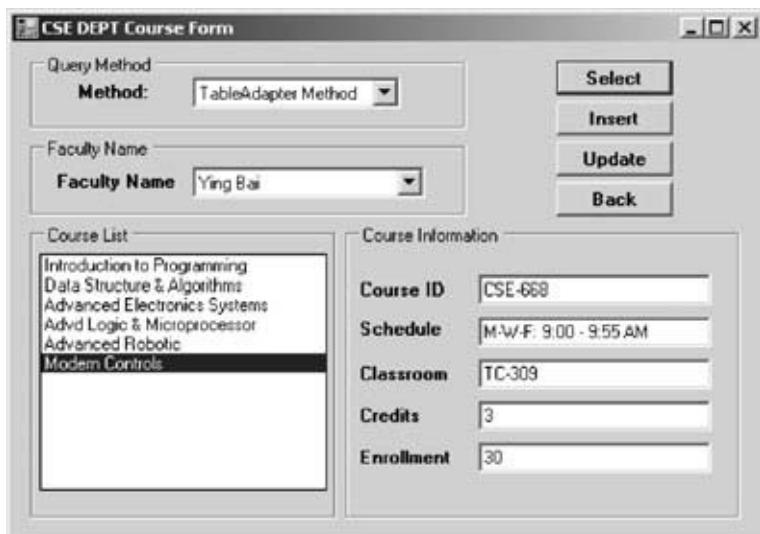


Figure 5.79. The data validation process.

## 5.7.2 Insert Data into the Oracle Database Using Stored Procedures

There is no significant difference between inserting data into an SQL Server database and an Oracle database using stored procedures. One of the most important differences is the stored procedure itself. A package component must be used to contain the stored procedure in the Oracle database if returned data is desired from the stored procedure.

In this section, we use the Course form and Course table to show readers how to insert a new course record into the Course table using a stored procedure in the Oracle database environment.

To illustrate how to insert data into an Oracle database using stored procedures or packages, we will utilize the following steps:

1. Develop a user interface – Insert Course Form window
2. Develop the package to contain stored procedures to perform data insertion into the Oracle database
3. Develop the code to call the package developed in step 2 to complete the data insertion
4. Validate the data insertion using the Course form window

Step 1 is similar to the step we did for the last project, SQLInsertRTOObjectSP; refer to Section 5.7.1.1 to get a more detailed description of how to develop a user interface. The only difference is that the Faculty ID text box is removed since it does not contain any input information. Steps 3 and 4 are very similar to the steps we developed for the same project in the last section. We will emphasize and highlight the different coding when we develop those steps below.

To save time and space, we modify the project OracleInsertRTOObject that we developed in Section 5.6.4 and create a new project named OracleInsertRTOObjectSP. Refer to Section 5.6.4 to get more detailed information on how to modify the current project to create the new project.

Now let's start from step 2 and develop a stored procedure for the Oracle database.

### 5.7.2.1 Develop Stored Procedures in the Oracle Database

A very detailed discussion of creating and manipulating packages and stored procedures in Oracle databases is provided in Section 4.19.7 in Chapter 4. Refer to that section to get more detailed information on creating Oracle stored procedures.

The topic discussed in this section is the insertion of data into the database, so no returned data is needed for this section. Therefore we only need to create stored procedures in the Oracle database, not packages, to perform the data insertion functionality.

As discussed in Section 4.19.7 in Chapter 4, different methods can be used to create Oracle stored procedures. In this section, we will use the Object Browser page provided by Oracle Database 10g XE to create our stored procedures.

Open the Oracle Database 10g XE home page by going to Start>All Programs|Oracle Database 10g Express Edition|Go To Database Home Page. Finish the login process by entering the correct username and password (in our case, it is SYSTEM and reback). Click the Object Browser and select the Create|Procedures item to open the Create Procedure window. Click the Create button and select the Procedure icon from the list to open this window. The opened window is shown in Figure 5.80.

Enter InsertFacultyCourse into the Procedure Name box, keep the Include Argument check box checked, and click the Next button to go to the next page.

The next window 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 gets the faculty\_id from the Faculty table based on the faculty name that is

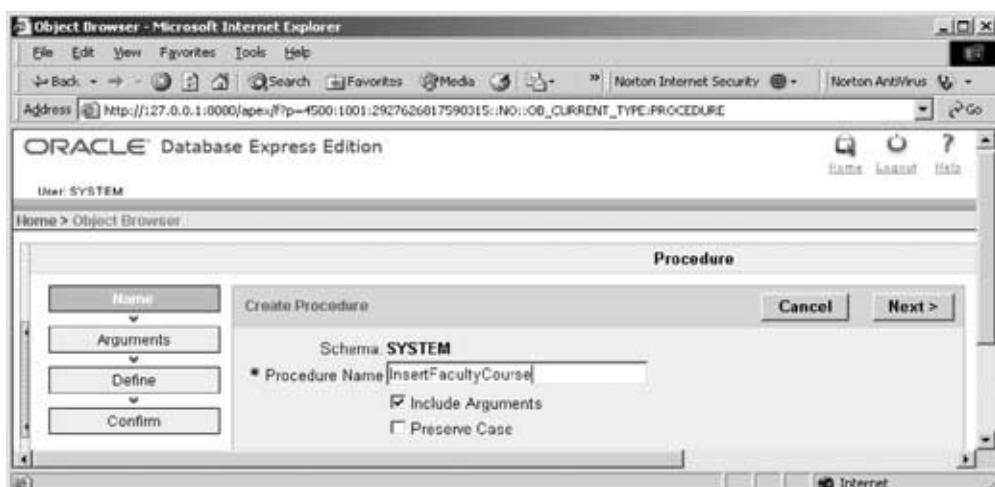


Figure 5.80. The opened Create Procedure window.

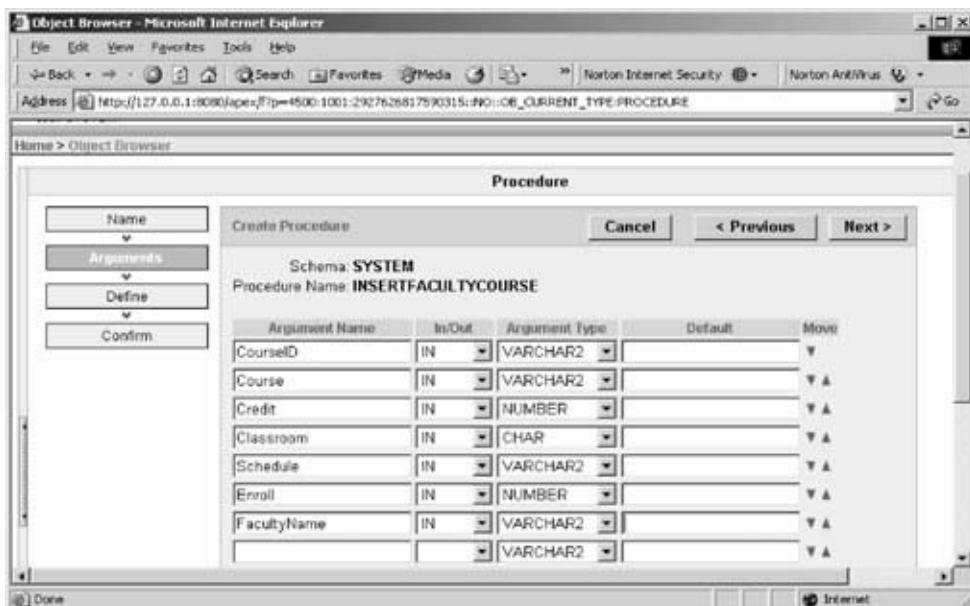


Figure 5.81. The finished argument list.

selected by the user from the ComboName combo box control in the Course form window, and the second query inserts a new course record that contains six pieces of information into the Course table based on the faculty\_id that is obtained from the first query. The seven input parameters are Faculty Name, Course ID, Course Title, Credit, Classroom, Schedule, and Enrollment. The first input parameter, Faculty Name, is used by the first query, and the following six input parameters are used by the second query.

Enter those input parameters one by one into the argument box. The data type of each input parameter must be identical to the data type of each data column used in the Course table. Refer to Section 2.11.3 in Chapter 2 to get a detailed list of data types used for those data columns in the Course data table.

For the Input/Output 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 shown in Figure 5.81.

Click the Next button to go to the procedure-defining page. Enter the code shown in Figure 5.82 as the body of the new procedure using Procedural Language Extension for SQL, or PL/SQL. Then click the Next and the Finish buttons to confirm creating this procedure. Your finished stored procedure should match the one shown in Figure 5.83.

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 must be identical to 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.

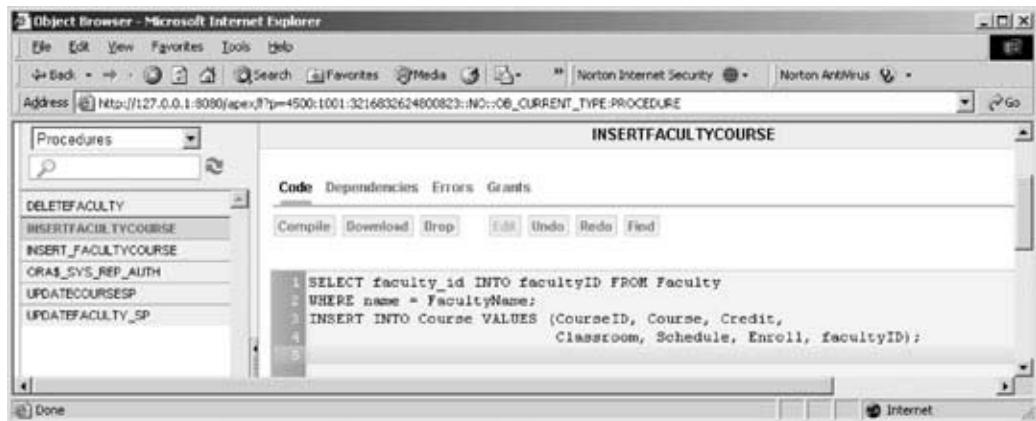


Figure 5.82. The stored procedure body.

Two queries are included in this procedure. The first query gets the faculty.id from the Faculty table based on the input parameter FacultyName, and the second query inserts seven input parameters into the Course table based on the faculty.id obtained from the first query. A semicolon must be used 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 5.83. This coding has been highlighted. Click the Edit button to add this local variable. This local variable is used to hold the faculty.id returned from executing the first query.

Another important issue is that the order of the input parameters or arguments in the INSERT command must be identical to the order of the columns in the associated data table. For example, in the Course table, the order of the data columns is as follows: course\_id, course, credit, classroom, schedule, enrollment, and faculty\_id. Accordingly, the order of the input parameters in the INSERT argument list must be identical to the data column order displayed above.

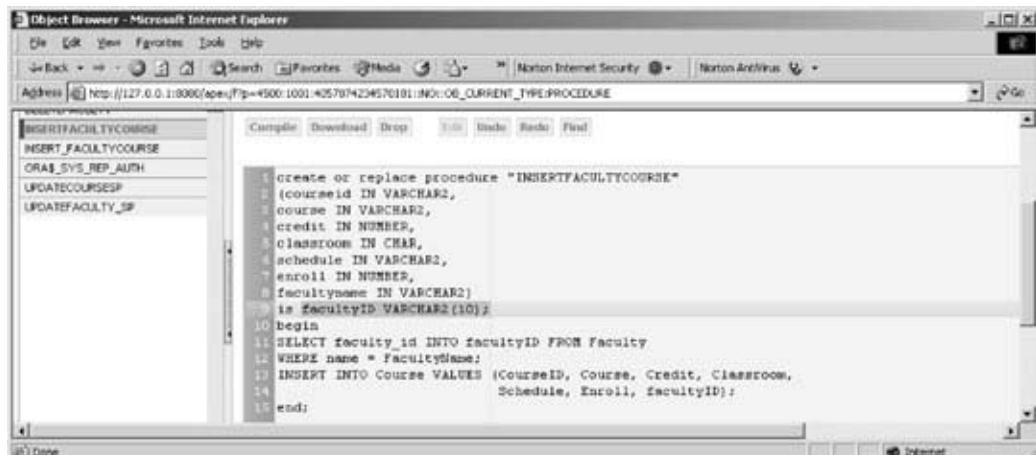


Figure 5.83. The completed stored procedure.

To make sure that this procedure works properly, we need to compile it first. Click the Compile button to compile and check the procedure. A successful compilation message will be displayed if the procedure is a bug-free stored procedure.

Close Oracle Database 10g XE by clicking the Close button. Next, we need to develop the code in our Visual Basic.NET project to call this stored procedure to perform the data insertion.

### **5.7.2.2 Develop Code to Call Stored Procedures to Insert Data into the Course Table**

Basically, the coding in this section is very similar to the coding we developed in Section 5.7.1.3. The functionality of this coding is to develop a new form, the Insert Course Form, to allow users to enter seven pieces of information related to a new course into the Course table. In the following sections, we emphasize and highlight only the important and different parts of the coding for the Oracle database.

#### **5.7.2.2.1 Validate Data Before the Data Insertion and Startup Coding**

This section's coding is identical to the coding in Section 5.7.1.3.1. The only difference is in the Imports commands. Since we are using the Oracle database, the Imports commands should include the System.Data.OracleClient namespace. In Section 5.7.2, we modified the project OracleInsertRTOBJECT and created a new project OracleInsertRTOBJECTSP. Now open this new project and open the code window of the Insert Course Form. Add the following two Imports commands to the top of this code window:

---

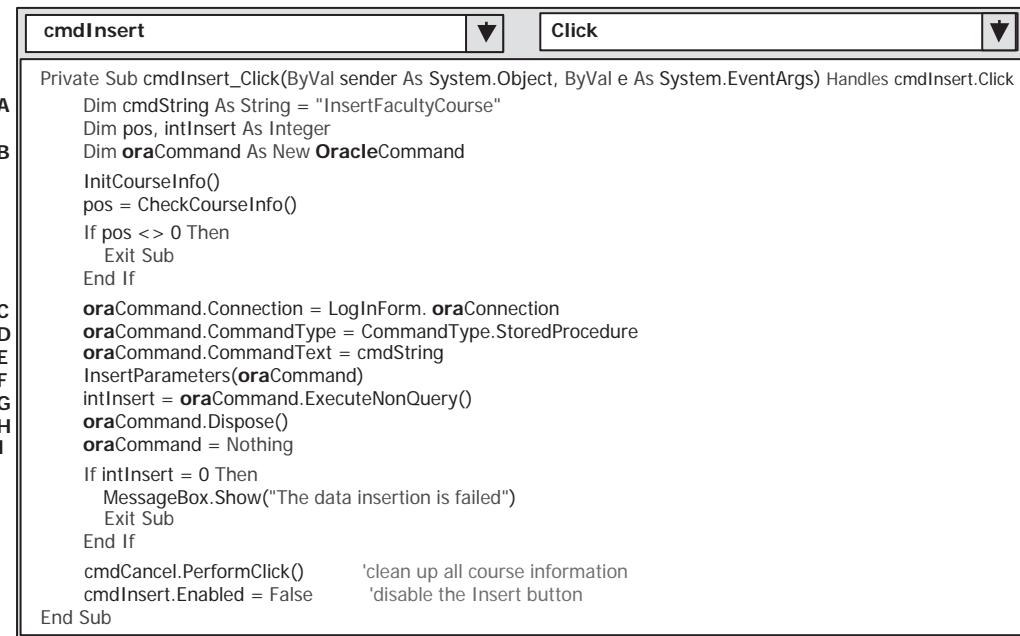
```
Imports System.Data
Imports System.Data.OracleClient
```

---

Also remove the coding related to the Faculty ID text box. Three procedures contain the coding related to that text box: the InitCourseInfo(), CheckCourseInfo(), and Cancel button event procedures. Reorder the index for the CourseInfo() array after the Faculty ID text box is removed from the InitCourseInfo(), and reduce the upper bound of the loop number to 4 for the subroutine CheckCourseInfo().

#### **5.7.2.2.2 Develop Code to Call Stored Procedures**

The main coding job to call the stored procedure is developed inside the Insert button event procedure in the Insert Course Form window. The coding for this event procedure is very similar to the coding we did for the same event procedure in the last project, SQLInsertRTOBJECTSP. Open that project and that event procedure, copy the coding from that procedure, and paste it into our Insert button event procedure. Also copy the user-defined subroutine InsertParameters() and paste it into our code window.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdInsert" and the dropdown menu says "Click". The code is written in VB.NET and handles the Click event for the button "cmdInsert". The code uses Oracle objects like "oraCommand" and "oraConnection". It includes modifications marked with bold letters A through I. The code performs an insert operation into a stored procedure named "InsertFacultyCourse". It checks if the insertion was successful, performs cleanup, and disables the insert button.

```

Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim cmdString As String = "InsertFacultyCourse"
    Dim pos, intInsert As Integer
    Dim oraCommand As New OracleCommand
    InitCourseInfo()
    pos = CheckCourseInfo()
    If pos <> 0 Then
        Exit Sub
    End If
    oraCommand.Connection = LogInForm.oraConnection
    oraCommand.CommandType = CommandType.StoredProcedure
    oraCommand.CommandText = cmdString
    InsertParameters(oraCommand)
    intInsert = oraCommand.ExecuteNonQuery()
    oraCommand.Dispose()
    oraCommand = Nothing
    If intInsert = 0 Then
        MessageBox.Show("The data insertion is failed")
        Exit Sub
    End If
    cmdCancel.PerformClick()      'clean up all course information
    cmdInsert.Enabled = False    'disable the Insert button
End Sub

```

Figure 5.84. Modifications to the coding of the Insert button event procedure.

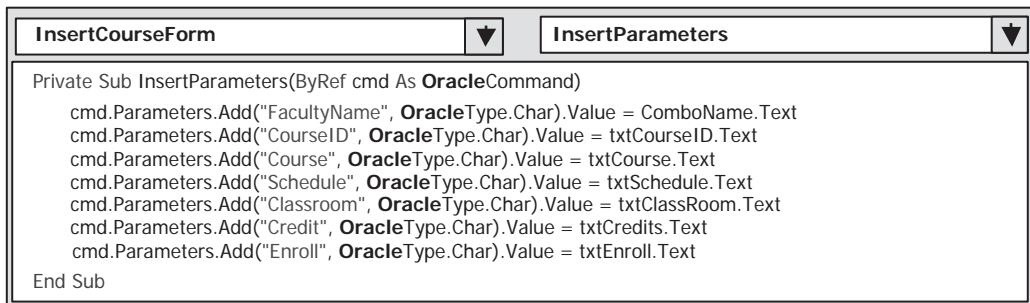
Some modifications are made to these two procedures to make them work for the Oracle database. All modifications are indicated in bold in Figure 5.84. Let's have a detailed discussion of those modifications one by one based on the steps in Figure 5.84.

First, let's concentrate on the modifications for the Insert button event procedure.

1. In step **A**, change the content of the query string, which is the name of the stored procedure, to the procedure name that we defined when we created this stored procedure using the Object Browser page in Oracle Database 10g XE in Section 5.7.2.1.
2. Change the prefix of the Oracle classes and objects to Oracle and ora.
3. Change the prefix of all Oracle objects from sql to ora in steps **C** to **I**.

Now let's take care of the modifications to the user-defined subroutine `InsertParameters()`. All modifications are indicated in bold in Figure 5.85.

The functionality of this subroutine is to assign each piece of information stored in each text box to the associated input parameter defined in the Oracle stored procedure `InsertFacultyCourse`. One key point of this coding is that the name of each parameter, which is represented as a string and located at the first argument's position, must be identical to each input parameter name defined in the stored procedure. For example, the name of the parameter `FacultyName` used here must be identical to the input parameter name `FacultyName` existing in the input parameter list we defined at the beginning of the stored procedure `InsertFacultyCourse`.



**Figure 5.85.** Modifications to the subroutine InsertParameters.

A runtime error would be encountered if a parameter name was not matched with the associated parameter name in the stored procedure as the project runs. Refer to Figure 5.81 for a detailed list of all parameter names defined in the stored procedure.

Now we have finished the coding for this data insertion operation. Before we can run the project to test the functionality of this data insertion, we need to figure out how to open and start the Insert Course Form to perform this new course insertion. There are two ways to open the Insert Course Form: to directly start this form and insert data or to use the Course form to trigger this form. We prefer to use the second way since we need to use the Course form to perform the data validation after a new record is inserted in the Course table.

Let's make three modifications to the Course form to trigger the Insert Course Form.

- The first modification is to create a form-level object of the InsertCourseForm class, and this object is used to start the Insert Course Form window when the Insert button in the Course form is clicked. To do that, add the following command under the class header:

---

#### Private InsertCourse As New InsertCourseForm

---

- The second modification is to add a command to close the Insert Course Form window when the Back button in the Course form is clicked. To do that, add the following command line into the Back button event procedure, cmdBack\_Click():

---

#### InsertCourse.Close()

---

- The third modification is to call the Show() method of the Insert Course Form object to display that form when the Insert button in the Course form is clicked.

To do that, add the following command into the Insert button event procedure in the Course form, cmdInsert\_Click():

---

**InsertCourse.Show()**

---

Now let's run the project to test the new data insertion using the stored procedure. Click the Start Debugging button to start the project, enter a suitable user-name and password, such as jhenry and test, in the LogIn form, and select the Course Information item from the Selection form to open the Course form window. Click the Insert button to open the Insert Course Form window to perform the new course insertion.

Enter the following data into the associated text boxes as the information for a new course:

- Ying Bai Faculty Name combo box
- CSE-668 Course ID text box
- Modern Controls Course Title text box
- M-W-F: 9:00–9:55AM Schedule text box
- TC-309 Classroom text box
- 3 Credits text box
- 30 Enrollment text box

Your finished information window should match the one shown in Figure 5.86.

Click the Insert button to call the stored procedure to insert this new course record into the database. The Insert button is disabled immediately after this data insertion. Was our data insertion successful? To answer this question, we need to perform the data validation in the next section.

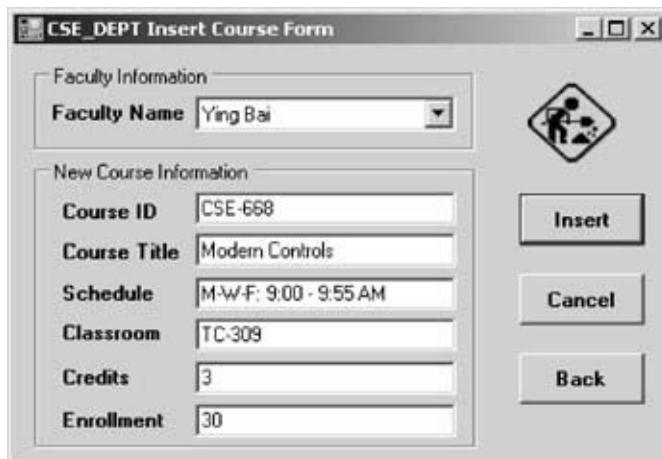


Figure 5.86. The running status of the Insert Course Form window.

The screenshot shows a Windows application window titled "CSE DEPT Course Form". On the left, there's a "Query Method" section with a dropdown menu set to "TableAdapter Method". Below it is a "Faculty Name" section with a dropdown menu showing "Faculty Name: Ying Bai". To the right of these are four buttons: "Select", "Insert", "Update", and "Back". The main area is divided into two sections: "Course List" and "Course Information". The "Course List" section contains a list box with several items: "Introduction to Programming", "Data Structure & Algorithms", "Advanced Electronics Systems", "Advd Logic & Microprocessor", and "Modern Controls". The last item, "Modern Controls", is highlighted with a black background. The "Course Information" section contains five text boxes with the following data:

Course ID	CSE-668
Schedule	M-W-F: 9:00 - 9:55 AM
Classroom	TC-309
Credits	3
Enrollment	30

Figure 5.87. The data validation process.

#### 5.7.2.2.3 Validate Data After Data Insertion

Now click the Back button to return to the Course form window to validate this data insertion.

Click the drop-down arrow of the ComboName combo box control and select Ying Bai from the list since we inserted a new course for this faculty member in the last section. Click 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 member are displayed in the Course List box. The last item is the course we just added to the Course table in the last section. Click that item and all information related to that new course is displayed in this form, as shown in Figure 5.87. This is the evidence that our data insertion using the stored procedure is successful!

A completed project OracleInsertRTOBJECTSP that includes the data insertion using the stored procedure can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**.

## 5.8 CHAPTER SUMMARY

Three popular data insertion methods were discussed and analyzed using 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 to the DataTable in the DataSet
3. Using the Command object's ExecuteNonQuery() method

Method 1 was developed using Visual Basic.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 from 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, insertions, and deletions) to the database.

A good habit is to try to use the TableAdapter.Insert() method when your application uses objects to store data (for example, when you are using text boxes to store your data), or when you want finer control over the creation of 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 to the data table located in the DataSet, and second, the TableAdapter.Update() method can be executed to update the whole table in the DataSet to the associated table in the database.

Method 3 is a runtime object method. This method is more flexible and convenient and allows users to insert data into multiple data tables with the different functionalities.

This chapter was divided into two parts. Part I provided a detailed discussion and analysis of inserting data into three different databases using Visual Basic.NET design tools and wizards. It is simple and easy to develop a data insertion project with these tools and wizards. The disadvantage of using these tools and wizards is that data can only be inserted to limited destinations, for example, a certain data table. Part II presented the runtime object method, which improves the efficiency of the data insertion and provides more flexibility in data insertion.

Nine real projects were provided in this chapter to give readers a clear and direct picture of developing professional data insertion applications in the Visual Basic.NET environment.

## 5.9 HOMEWORK

### I. True/False Selections

- \_\_\_\_\_1. Three popular data insertion methods are TableAdapter.Insert(), TableAdapter.Update(), and ExecuteNonQuery() of the Command class.
- \_\_\_\_\_2. Unlike the Fill() method, a valid database connection must be set before 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 from 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 these two collections start

with a different index; the former starts from 1, and 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 to the names of the input parameters defined in the stored procedures.

## II. Multiple Choices

- 1. To insert data into the database using the TableAdapter.Insert() method, one needs to use the \_\_\_\_\_ to build the \_\_\_\_\_.
  - a. Data Source, Query Builder
  - b. TableAdapter Query Configuration Wizard, Insert query
  - c. Runtime object, Insert query
  - d. Server Explorer, Data Source
- 2. To insert data into the database using the TableAdapter.Update() method, one needs first to add new data into the \_\_\_\_\_ and then update that data into the database.
  - a. Data table
  - b. Data table in the database
  - c. DataSet
  - d. Data table in the DataSet
- 3. To insert data into the database using the TableAdapter.Update() method, one can update \_\_\_\_\_.
  - a. One data row only
  - b. Multiple data rows
  - c. The whole data table
  - d. Any of the above
- 4. Because 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 an Oracle database.
  - a. :=
  - b. LIKE
  - c. =
  - d. @
6. To confirm the stored procedure built in the Object Browser page in an 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 an 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.InsertCommand
  - b. The content of the CommandType.Text
  - c. The name of the Insert command
  - d. The name of the stored procedure

### III. Exercises

- Following is a stored procedure developed in the SQL Server database. Please develop a piece of code 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
```

---

- Following is a piece of code developed in Visual Basic.NET. This coding is used to call a stored procedure in the Oracle database to insert a new record into the database. Please create the associated stored procedure in the Oracle database using the PL/SQL language.
- 

```
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()
```

---

3. Use 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 is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5\InsertWizard**).
4. Use runtime objects to complete the data insertion query for the Student form by using the project AccessInsertRTOBJECT (the project file is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5\AccessInsertRTOBJECT**).
5. Use a stored procedure to complete the data insertion query for the Student form to the Student table by using the project OracleInsertRTOBJECTSP (the project file is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 5**).

---

# 6

---

## Data Updating and Deleting with Visual Basic.NET

In this chapter, we will discuss how to update and delete data from databases. Basically, many different methods are provided and supported by Visual Basic.NET and the .NET Framework to help users update and delete data from a 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 in the database
2. Using the TableAdapter.Update() method to update and execute the associated TableAdapter's properties, such as UpdateCommand or DeleteCommand, to update changes made to the table in the DataSet to the table in the database
3. Using the runtime object method to develop and execute the Command object's ExecuteNonQuery() method to update or delete data from the database directly

Both methods 1 and 2 need to use Visual Basic.NET design tools and wizards to create and configure suitable TableAdapters, build associated queries using Query Builder, and call those queries from Visual Basic.NET applications. The difference between methods 1 and 2 is that method 1 can be used to directly access a database to perform data updating or deleting in a single step, but method 2 needs two steps to finish data updating or deleting. First, data updating or deleting is performed on the associated tables in the DataSet, and then that updated or deleted data is updated in the tables in the database by executing the TableAdapter.Update() method.

This chapter is divided into two parts: Part I discusses of data updating and deleting using methods 1 and 2, or, in other words, using the TableAdapter.Update() and TableAdapter.Delete() methods developed with Visual Basic.NET design tools and wizards. Part II presents data updating and deleting using the runtime object method to develop command objects to execute the ExecuteNonQuery() method dynamically.

When you have finished this chapter, you will

- Understand the working principle and structure of updating and deleting data from a database using Visual Basic.NET design tools and wizards
- Understand the procedures of configuring a TableAdapter object by using the TableAdapter Query Configuration Wizard and build a query to update and delete data from the database
- Design and develop special procedures to validate data before and after data updating and deleting
- Understand the working principle and structure behind updating and deleting data from a database using the runtime object method
- Design and build stored procedures to perform data updating and deleting

To successfully complete this chapter, you need to understand topics such as the fundamentals of databases, introduced in Chapter 2, and ADO.NET, discussed in Chapter 3. Also, the sample database CSE\_DEPT, which was developed in Chapter 2, will be used throughout this chapter.

Three kinds of databases will be used in the example projects in this chapter to illustrate how to perform data updating and deleting. These databases are Microsoft Access, SQL Server 2005, and Oracle Database 10g XE.

In order to save time and avoid repetition, we will use sample projects, such as InsertWizard, SQLInsertWizard, AccessInsertRTOBJECT, SQLInsertRTOBJECT, and OracleInsertRTOBJECT, which we developed in the previous chapters, and modify them to create new associated projects 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 to be exact, the event procedures related to those buttons, will be developed and built in this chapter. In this chapter, we will concentrate only on the coding for the Update and Delete buttons.

## PART I DATA UPDATING AND DELETING WITH VISUAL BASIC.NET DESIGN TOOLS AND WIZARDS

In this part, we will discuss updating and deleting data from a database using Visual Basic.NET design tools and wizards. We will develop two methods to perform these data actions. First, we will use the TableAdapter DBDirect methods, TableAdapter.Update() and TableAdapter.Delete(), to directly update or delete data from the database. Second, we will show readers how to update or delete data from the database by first updating or deleting records from 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 will use is the Access database, CSE\_DEPT.mdb, which was developed in Chapter 2 and is located in the folder **database** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). You can also use other databases, such as Microsoft SQL Server 2005 or Oracle Database 10g XE. The only issue is that you need to select and connect the correct database with your applications when you use the Data Sources window to set up your data source for your Visual Basic.NET data-driven applications.

## 6.1 UPDATE OR DELETE DATA FROM DATABASES

We have already provided a very detailed discussion of TableAdapter DBDirect methods in Section 5.1.1 in Chapter 5. To use these methods to directly access a database to make the desired manipulations to the data stored in the database, you need to use Visual Basic.NET design tools and wizards to create and configure the associated TableAdapter. These DBDirect methods have some limitations. For example, each TableAdapter is associated with a unique data table in the DataSet; therefore, data updating or deleting by using the associated TableAdapter can be executed only for that data table. In other words, the specified TableAdapter cannot update or delete data from any other data tables except the data table that is related to it.

### 6.1.1 Updating and Deleting Data from Related Tables in a DataSet

When updating or deleting data from 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 DataRowCollection in the DataSet. To prevent data integrity errors, the best practice is to update or delete data from a 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 the sample database CSE\_DEPT, all five tables are related with different primary and foreign keys. For example, faculty\_id works as a key between the LogIn, Faculty, and Course tables to relate them together. Here, faculty\_id is the primary key in the Faculty table but a foreign key in both the LogIn and Course tables. In order to update or delete data from any of these tables, you need to follow the sequence mentioned earlier. When updating or deleting a record from the database, the following data operations need to be performed:

1. First, the record should be deleted from the child tables, LogIn and Course.
2. Then the record should be updated in or deleted from the parent table, Faculty.
3. Finally, the updated record should be inserted into the child tables to update the data. There is no further action for deleting data from the child tables.

It would be terribly complicated if you tried to update a completed record (including updating the primary key) for existing data in the sample database, and in practice it is unnecessary to update the primary key for any record since the primary key has the same lifetime as the database. The popular way is to remove the undesired records and then insert new records with new primary keys. So in this chapter, we will concentrate on either updating or deleting a whole record (including the primary key) from the database for newly inserted data or updating the existing data in the sample database without touching the primary key.

**Table 6.1. TableAdapter DBDirect Methods**

<b>TableAdapter DBDirect Method</b>	<b>Description</b>
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 a database on the basis of the original column values passed in as method parameters.

### **6.1.2 Update or Delete Data from a Database by Using TableAdapter DBDirect Methods**

Three typical TableAdapter DBDirect methods are listed in Table 5.1. For your convenience, we repeat that table in this section as Table 6.1.

Both, TableAdapter.Update() and TableAdapter.Delete(), when executed, need the original column values as the parameters. The TableAdapter.Update() method needs both the original and the new column values to perform data updating. Another point to note is that when the application uses the object to store data – for instance, in the sample project we use text box objects to store data – you should use this DBDirect method to perform data manipulations in the database.

### **6.1.3 Update or Delete Data from a Database by Using the TableAdapter.Update Method**

You can use the TableAdapter.Update() method to update or delete records from a database. The TableAdapter.Update() method provides several overloads that perform different operations depending on the parameters passed. It is important to understand the results of calling these different method signatures.

To use the TableAdapter.Update() method to update or delete data from a database, you need to perform the following two steps:

1. Change or delete records from the desired DataTable on the basis of 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 in the database. You can control the amount of data to be updated by passing an entire DataSet, a DataTable, an array of DataRows, or a single DataRow.

**Table 6.2.** Variations of the TableAdapter.Update() Method

Update Method	Description
TableAdapter.Update(DataTable)	Attempts to save all changes in the DataTable to the database. This includes removing any rows deleted from the table, adding rows inserted into the table, and updating any rows in the table that have changed.
TableAdapter.Update(DataSet)	Attempts to save all changes in the TableAdapter's associated DataTable to the database, although the parameter takes a dataset, the TableAdapter. This includes removing any rows deleted from the table, adding rows inserted into the table, and updating any rows in the table that have changed.
TableAdapter.Update(DataRow)	Attempts to save changes in the indicated DataRow to the database.
TableAdapter.Update(DataRows())	Attempts to save changes in any row in the array of DataRows to the database.
TableAdapter.Update("new column values", "original column values")	Attempts to save changes in a single row that is identified by the original column values.

Table 6.2 describes the behavior of the various TableAdapter.Update() methods.

Different parameters or arguments can be passed to 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 in those tables in the database.

The last variation of this method is to pass the original and the new columns of a data table to perform updating. In fact, this method can be used as a DBDirect method to access the database to manipulate data.

In order to provide a detailed discussion and explanation of how to use these two methods to update or delete records from a database, a real example will be very helpful. Let's first create a new Visual Basic.NET project to handle these issues.

## 6.2 UPDATE AND DELETE DATA FROM AN ACCESS DATABASE BY USING THE SAMPLE PROJECT ACCESSUPDATEDELETEWIZARD

We have provided a very detailed introduction to the design tools and wizards in Visual Basic.NET in Section 4.2 in Chapter 4. The popular design tools and wizards include DataSet, BindingSource, TableAdapter, the Data Source window, the Data Source Configuration Wizard, and DataSet Designer. You need to use these to develop your data updating and deleting sample project based on the InsertWizard project developed in the previous chapter. First, let's copy that project

and make some modifications to it to get the new project. The advantage of creating the new project in this way is that you don't need to redo the data source connection and configuration because those jobs were performed in the previous chapter.

### 6.2.1 Create a New Project Based on the InsertWizard Project

Open Windows Explorer and create a new folder, **Chapter 6**. Then browse to the project InsertWizard that is in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and copy this project to your new folder, **Chapter 6**. Change the name of the solution and the project from InsertWizard to AccessUpdateDeleteWizard. Double-click AccessUpdateDeleteWizard Project.vbproj to open this project.

On the opened project, perform the following modifications to get the desired project:

1. Go to the Project|AccessUpdateDeleteWizard Project Properties menu to open the project's property window. Change the Assembly name from InsertWizard Project to AccessUpdateDeleteWizard Project and the Root namespace from InsertWizard\_Project to AccessUpdateDeleteWizard\_Project, respectively.
2. Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to AccessUpdateDeleteWizard Project. Click OK to close this dialog box.
3. Go to File|Save All to save these modifications.

Now you are ready to develop your graphical user interfaces (GUIs) based on the InsertWizard Project you developed before.

### 6.2.2 Application User Interfaces

Recall that when we developed the project InsertWizards, there were five command buttons located in the Faculty form window: Select, Insert, Update, Delete, and Back. In this section, we need to use both the Update and Delete buttons, or these two buttons' event procedures, to perform data updating and deleting actions on the database. Unlike adding a new record to the database, for the update and delete operations we don't need to develop new form window as the user interface to collect the new data. Instead, we can use the Faculty form with some modifications.

#### 6.2.2.1 Modify the Faculty Form Window

Since we have two methods available to perform data updating or deleting, we need to add one more control to the Faculty form window to enable us to select between these two methods as the project runs. We will add a combo box control named ComboMethod to the upper left corner of the Faculty form window.

Also, we need to change all five label controls that are located inside the Faculty Information group box to five text box controls because we need to change the faculty information by entering new data into those text boxes. Besides these five

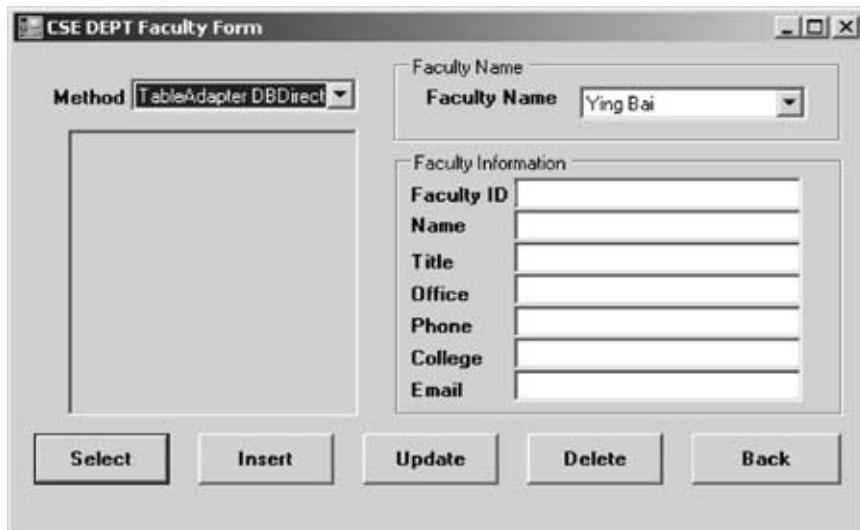


Figure 6.1. The modified Faculty form window.

text box controls, we will add two more text boxes into the group box and name them txtID for faculty\_id and txtName for Faculty Name, respectively. These two new text boxes will be used to hold the faculty ID and faculty name information.

The modified Faculty form window is shown in Figure 6.1.

Another advantage of using this Faculty form as the user interface is that we don't even need to use a search process to find the data items to be updated or deleted from the database. Instead, we can first get the data to be updated or deleted by performing a data query using the Select button event procedure. Then we can easily update or delete that data by modifying any piece of information that is related to that data and stored in the associated text box in the Faculty form window.

### 6.2.2.2 Bind Data for All Text Boxes of the Faculty Form Window

This data binding is necessary since we need to call the Select button event procedure to execute the data query when we perform data validations for data inserting, updating, and deleting later.

1. Open the Faculty form window if it is not opened, and select the first text box, txtID, which you just added.
2. Go to the Properties window, and expand the DataBindings item to the Text property.
3. Click the drop-down arrow next to the Text property to open the Add Project Data Source dialog box.
4. Select the faculty\_id from the list by clicking it, as shown in Figure 6.2a. In this way, a binding relationship between the txtID text box in the Faculty form and the faculty\_id column in the DataSet is set up.

In a similar way, you can finish the data bindings for all of these seven text boxes. An example of binding for the Email text box is shown in Figure 6.2b.

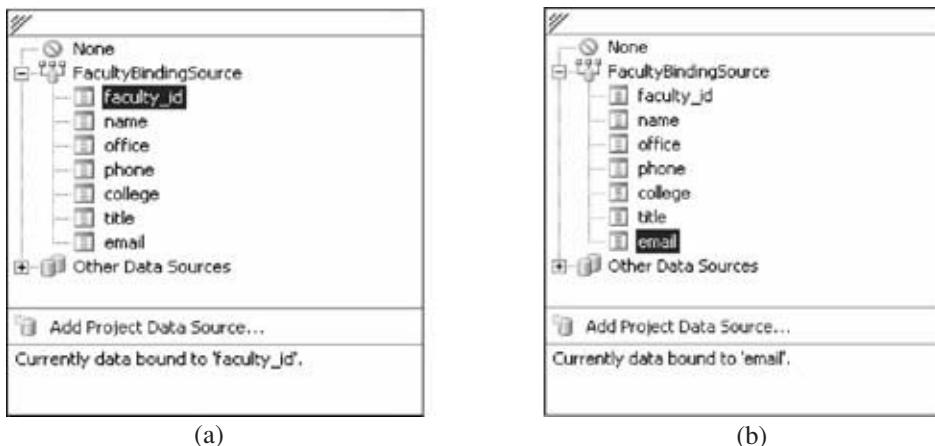


Figure 6.2. The data binding process.

### 6.2.3 Validate Data Before Data Updating and Deleting

Data validation can also be neglected because when we perform a data query by clicking the Select button, the retrieved data is complete data and can be displayed in the Faculty form window. This means that all text boxes have been filled with the related faculty information; whether we make some modifications or not, all text boxes are full. So data validation before data updating or deleting can be avoided.

### 6.2.4 Build Update and Delete Queries

As we mentioned, two methods will be discussed in this part: one is to update or delete records by using the TableAdapter DBDirect method, and the other is to use the TableAdapter.Update() method to update modified records from the DataSet to the database. First, let's concentrate on the first method.

Let's build data updating and deleting queries using the TableAdapter Query Configuration Wizard and Query Builder.

#### 6.2.4.1 Configure TableAdapter and Build the Data Updating Query

1. Open the Data Sources window by going to the Data|Show Data Sources menu.
2. In the Data Sources window, click the Edit the DataSet with Designer button, second from the left on the toolbar, to open the designer.
3. Then right-click the last item in the Faculty table, and select the Add Query item from the pop-up menu to open the TableAdapter Query Configuration Wizard. Keep the default selection “Use SQL statements” unchanged. Click Next to go to the next window.
4. Select the UPDATE item in this window since you need to perform a data updating query, and then click Next to continue.
5. Click the Query Builder button since you want to build an updating query. The opened Query Builder window is shown in Figure 6.3.

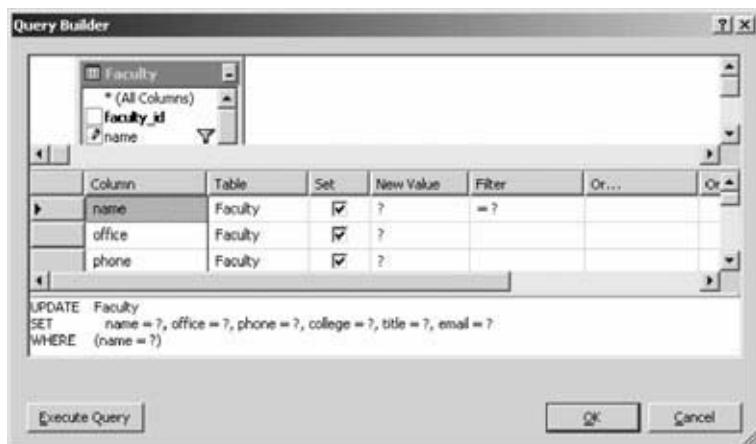


Figure 6.3. The Query Builder for the Update query.

6. Remove the default question mark from the faculty\_id row under the column Filter, and enter a question mark in the name row under the column Filter. Press the Enter key on the keyboard.
7. Remove all rows under the last row, email, and your finished Query Builder should match the one shown in Figure 6.3.
8. Click OK to go to the next window. Click Next to confirm this query and continue to the next step.
9. Modify the query function name from the default one to UpdateFaculty, and click Next to go to the last window.
10. Click Finish to complete the query building, and close the wizard. Immediately you will find that a new query function has been added to the Faculty TableAdapter as the last item.

Now let's continue to build the Delete query function, using Query Builder.

#### 6.2.4.2 Build the Data Deletion Query

1. Reopen Edit DataSet with the Designer window.
2. Right-click the last item in the Faculty table, and select the Add Query item to open the TableAdapter Query Configuration Wizard if it is not already open.
3. In the opened wizard, keep the default selection “Use SQL statements” unchanged. Click Next to go to the next window.
4. Select the DELETE item from this window since you need to perform a data deleting query, and click Next to continue.
5. Click the Query Builder button since you want to build a deleting query. The opened Query Builder window is shown in Figure 6.4.
6. Remove the question mark from the faculty\_id row under the column Filter.
7. Go to the middle pane, and then type name into the first column just under the faculty\_id row.

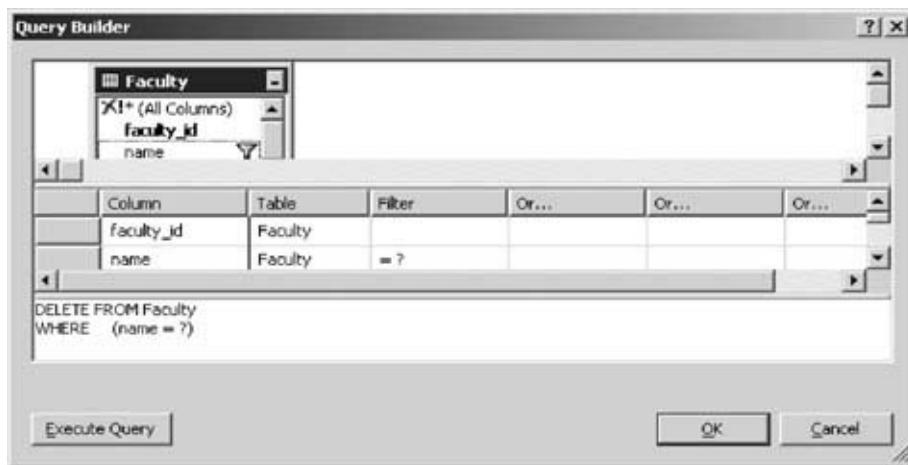


Figure 6.4. The Query Builder for the Delete query.

8. Move the cursor to the Filter column along the name row, and enter a question mark into that field. Press the Enter key on the keyboard. Your finished query builder should match the one shown in Figure 6.4.
9. Click OK to go to the next window. Click Next to confirm this query and continue to the next step.
10. Modify the query function name from the default one to DeleteFaculty, and click Next to go to the last window.
11. Click Finish to complete this query building and close the wizard. Immediately you will find that a new query function has been added to the Faculty TableAdapter as the last item.

### **6.2.5 Develop Code to Update Data by Using the TableAdapter DBDirect Method**

To perform data updating using this method, some startup coding and modifications to the original coding are necessary. We have divided the coding job into three subsections: coding modifications, startup coding, and update coding.

#### **6.2.5.1 Coding Modifications**

Because the target data to be updated or deleted from the database is newly inserted data, first some modifications need to be performed to the code in the Insert Faculty form.

1. Open the code window of the Insert Faculty form.
2. Change the scope of the form-level variable FacultyName to a global variable by replacing the accessing mode keyword Private with Public at the top of the code window.
3. Change the code for the Back button event procedure – replace Me.Close() with Me.Hide().

### 6.2.5.2 Startup Coding

The startup coding is developed inside the Faculty form. We will use this form to perform data updating, data deletion, and data validation functionalities.

1. Open the code window of the Faculty form.
2. Change the local object insertFaculty, which previously was created inside the Insert button event procedure cmdInsert\_Click(), to a form-level object, and place it in the top section of this code window.
3. Create two form-level variables, FacultyNameFlag (Boolean) and FacultyName (String). These two variables work as a flag and temporary storage unit for the faculty name selected.
4. Add one piece of code, insertFaculty.Close(), to the Back button event procedure, cmdBack\_Click(), and place it as the first command inside that event procedure. This code is used to close the Insert Faculty Form window when the Back button on the Faculty form is clicked.
5. Add one piece of code, FacultyNameFlag = True, to the Faculty Name Changed event procedure txtName\_TextChanged(). This code is used to indicate that the faculty name has been updated and that the old faculty name located in the ComboName combo box should be replaced by this updated faculty name.
6. Call the system event procedure ComboName\_SelectedIndexChanged() from the system event procedure ComboName\_DropDown() to indicate that a selected index in the ComboName combo box has been changed, and an associated event is created. The ComboName\_DropDown() event procedure is triggered by a ComboName\_DropDown event that is created when the user clicks the drop-down arrow of the ComboName combo box. The purpose of calling that event procedure is to add the updated faculty name into this combo box and remove the old faculty name from this box.
7. Add the coding shown in Figure 6.5 into the ComboName\_SelectedIndexChanged() event procedure. Two IF blocks are used for this coding. The first IF block is used to detect whether a new faculty name involved in a new faculty record has been added to the database from the Insert Faculty Form window. If yes, the newly inserted faculty name will be added into the ComboName combo box in the Faculty form. The second IF block is used to detect whether the faculty name involved in a faculty record has been updated. If yes, the updated faculty name will be added into the ComboName combo box and the old one will be removed from this box.

Figure 6.5 shows all these coding steps.

Now let's start the coding for our data updating.

### 6.2.5.3 Update Coding

The main coding to perform this data updating is developed inside the Update button event procedure. Open the project AccessUpdateDeleteWizard and the Faculty form window, then double-click the Update button to open its event procedure. Enter the code shown in Figure 6.6 into this event procedure.

**FacultyForm** (Declarations)

```

Public Class FacultyForm
    Private insertFaculty As New InsertFacultyForm
    Private FacultyNameFlag As Boolean
    Private FacultyName As String

    Private Sub cmdBack_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdBack.Click
        A insertFaculty.Close()
        Me.Close()
    End Sub

    Private Sub txtName_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles txtName.TextChanged
        B FacultyNameFlag = True
    End Sub

    Private Sub ComboName_DropDown(ByVal sender As Object, ByVal e As System.EventArgs) Handles _  
E ComboName.DropDown
        C ComboName_SelectedIndexChanged(sender, e)
    End Sub

    Private Sub ComboName_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles _  
F comboName.SelectedIndexChanged
        If insertFaculty.FacultyName <> String.Empty Then
            D ComboName.Items.Add(insertFaculty.FacultyName)
            insertFaculty.FacultyName = String.Empty
        End If
        If FacultyNameFlag = True Then
            E ComboName.Items.Remove(FacultyName)
            ComboName.Items.Add(txtName.Text)
        End If
    End Sub

```

**Figure 6.5.** The startup coding.

**cmdUpdate** Click

```

Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles_  
A cmdUpdate.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim FacultyRow As CSE_DEPTDataSet.FacultyRow
    Dim intUpdate As Integer
    Dim strFacultyID As String

    B If ComboMethod.Text = "TableAdapter DBDirect" Then
        intUpdate = FacultyTableApt.UpdateFaculty(txtName.Text, txtOffice.Text, txtPhone.Text, _  
txtCollege.Text, txtTitle.Text, txtEmail.Text, ComboName.Text)
    C Else
        'TableAdapter Update method selected
    End If
    D If intUpdate = 0 Then
        MessageBox.Show("Faculty Table Updating is failed!")
        Exit Sub
    End If
End Sub

```

**Figure 6.6.** Coding for the Update command button event procedure.

Let's take a look at this piece of newly added code to see how it works:

- A. All objects and variables used in this event procedure are declared here. An instance of the FacultyTableAdapter class is created first since you need to use it to perform the data updating. A new row object of the FacultyRow class is also created here since you need this object to update the data in the DataSet to the table in the database later when you use another method, TableAdapter.Update(), to perform the data updating. The local integer variable intUpdate is used to hold the value returned by the TableAdapter DBDirect method to update the data, and the local String variable strFacultyID is used to hold the faculty.id value returned by the second method, TableAdapter.Update(), to update the data in the next step.
- B. If the user selects the first method, TableAdapter DBDirect, to perform data updating, the updating function built in Section 6.2.4.1 is called to update the selected faculty record. This function will return a data value to indicate whether this function calling was successful. The value returned is equal to the number of rows or records that have been updated in the database.
- C. If the user selects the second method, TableAdapter.Update(), to update data, the related coding that will be developed later is executed to first update data in the DataSet and then update data in the database.
- D. If the value returned by 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 develop the code for the second data updating method.

### **6.2.6 Develop Code to Update Data by Using the TableAdapter.Update Method**

Open the Update button event procedure if it is not already open, and add the code shown in Figure 6.7 to this event procedure. Let's take a look at this piece of newly added code to see how it works. The code we developed in the previous step is highlighted with a gray background.

- A. In order to update a selected row from the Faculty table in the DataSet, first you need to identify that row. Visual Basic.NET provides a default method, FindBy(), to do that. But this method needs a primary key as a criterion to perform a query to locate the desired row in a table. The primary key for the Faculty table is faculty\_id. To find the faculty\_id, you can use the query function FindFacultyIDByName() you built in Section 4.14 in Chapter 4, with the Faculty Name as a 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, you must first find the old faculty row based on the old name. So the combo box 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 in the Faculty table, and the desired data row is returned and assigned to the local variable FacultyRow.

The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdUpdate" and the dropdown menu says "Click". The code is written in VB.NET. On the left margin, there are labels A through F corresponding to specific lines of code.

```

Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim FacultyRow As CSE_DEPTDataSet.FacultyRow
    Dim intUpdate As Integer
    Dim strFacultyID As String
    If ComboMethod.Text = "TableAdapter DBDirect" Then
        intUpdate = FacultyTableApt.UpdateFaculty(txtName.Text, txtOffice.Text, txtPhone.Text, _
            txtCollege.Text, txtTitle.Text, txtEmail.Text, ComboName.Text)
    Else
        'TableAdapter Update method selected
        strFacultyID = FacultyTableApt.FindFacultyIDByName(ComboName.Text)
        FacultyRow = CSE_DEPTDataSet.Faculty.FindByfaculty_id(strFacultyID)
        FacultyRow = UPFacultyRow(FacultyRow)
        Me.Validate()
        FacultyBindingSource.EndEdit()
        intUpdate = FacultyTableApt.Update(CSE_DEPTDataSet.Faculty)
    End If
    If intUpdate = 0 Then
        MessageBox.Show("Faculty Table Updating is failed!")
        Exit Sub
    End If
End Sub

```

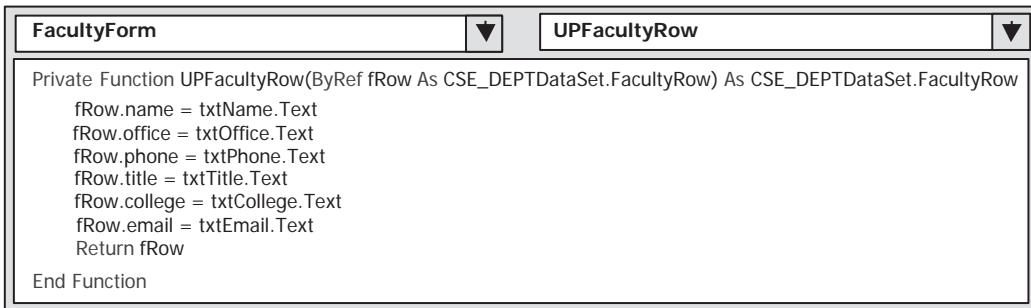
**Figure 6.7.** Coding for the second data updating method.

- C. A user-defined function UPFacultyRow() is called to assign all updated faculty information to the desired row. In this way, the faculty information – a row in the Faculty table in the DataSet – is updated.
- D. The Validate() command closes out the editing of a control. In our case, it closes any editing for text box controls 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 text box controls will be reflected in the associated column in the DataSet.
- F. Finally, the Update() method of the TableAdapter sends updated data back to the database. The argument of this method can be a whole DataSet, a DataTable in the DataSet, or a DataRow in a DataTable. In our case, we use the Faculty table as the argument for this method.

The detailed coding for the user-defined function UPFacultyRow() is shown in Figure 6.8. This function is straightforward and easy to understand.

The argument of this function is a DataRow object and is passed by a reference to the function. The advantage of passing an argument in this way is that any modifications performed to the DataRow object inside the function can be returned to the calling procedure without the need to create another returned variable. The updated faculty information that is stored in the associated text box 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 have finished the coding of two methods to update data in a database. Next, we will discuss how to delete data from a database.



```

FacultyForm
UPFacultyRow

Private Function UPFacultyRow(ByRef fRow As CSE_DEPTDataSet.FacultyRow) As CSE_DEPTDataSet.FacultyRow
    fRow.name = txtName.Text
    fRow.office = txtOffice.Text
    fRow.phone = txtPhone.Text
    fRow.title = txtTitle.Text
    fRow.college = txtCollege.Text
    fRow.email = txtEmail.Text
    Return fRow
End Function

```

Figure 6.8. Coding for the user-defined function UPFacultyRow.

### 6.2.7 Develop Code to Delete Data by Using the TableAdapter DBDirect Method

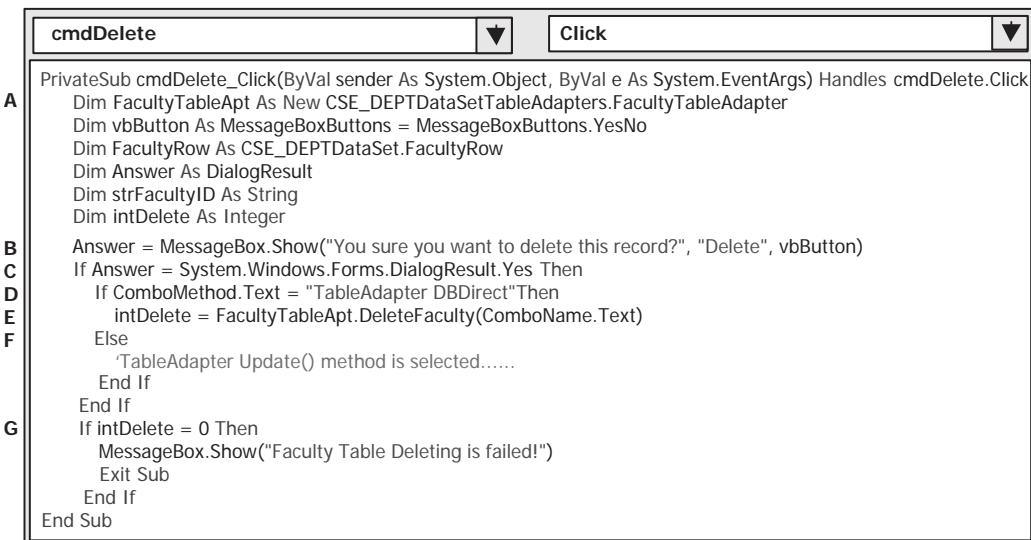
To delete data from a database, you can use either the TableAdapter DBDirect method TableAdapter.Delete() or the TableAdapter.Update() method. If your application does not use TableAdapters, you can use the runtime object method to create a command object to delete data from a database (e.g., ExecuteNonQuery).

The TableAdapter.Update() method is typically used when the application uses DataSets to store data, whereas the TableAdapter.Delete() method is typically used when the application uses objects, for example, the text boxes used here to store data.

Open the Faculty form window, and double-click the Delete button to open its event procedure. Enter the code shown in Figure 6.9 into this event procedure.

Let's take a look at this piece of code to see how it works:

- All data components and objects as well as variables used in this event procedure are declared and created here. The object of the FacultyTableAdapter



```

cmdDelete
Click

A PrivateSub cmdDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdDelete.Click
B     Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
C     Dim vbButton As MessageBoxButtons = MessageBoxButtons.YesNo
D     Dim FacultyRow As CSE_DEPTDataSet.FacultyRow
E     Dim Answer As DialogResult
F     Dim strFacultyID As String
G     Dim intDelete As Integer
H
I     Answer = MessageBox.Show("You sure you want to delete this record?", "Delete", vbButton)
J     If Answer = System.Windows.Forms.DialogResult.Yes Then
K         If ComboMethod.Text = "TableAdapter DBDirect"Then
L             intDelete = FacultyTableApt.DeleteFaculty(ComboName.Text)
M         Else
N             'TableAdapter Update() method is selected.....
O             End If
P         End If
Q         If intDelete = 0 Then
R             MessageBox.Show("Faculty Table Deleting is failed!")
S             Exit Sub
T         End If
U     End Sub
V
W End Sub

```

Figure 6.9. Coding for the Delete button event procedure.

class is created first since you need to use its Update() and Delete() methods to delete data later. A button object of the MessageBoxButtons class is created because you need to use these two buttons to confirm the data deletion later. The FacultyRow is used to locate the DataRow in the Faculty table and is used for the second deletion method. The local variable Answer is an instance of DialogResult, and it is used to hold the value returned by the MessageBox function. This variable can be replaced by an integer variable if you like.

- B. First, a MessageBox is called to confirm that data will be deleted from the Faculty table.
- C. If the value returned by this MessageBox is Yes, which means that the user has confirmed that this data deletion is fine, data deletion will be performed in the next step.
- D. If the user selects the first method, TableAdapter DBDirect, the query function you built in Section 6.2.4.2 will be called to perform the data deletion 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 selects the second method, TableAdapter.Update(), the associated coding 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 data returned by the TableAdapter.Delete() method or the TableAdapter.Update() method is an integer value, and it is stored in the variable intDelete. The value of this returned data is equal to the number of deleted data rows in the database or deleted DataRows in the DataSet. A zero value means that no data row has been deleted and that the data deletion failed. In that case, a warning message is displayed and the procedure is exited.

Now let's take a look at the coding for data deletion using the second method.

### **6.2.8 Develop Code to Delete Data by Using the TableAdapter.Update Method**

Add the code shown in Figure 6.10 into the Delete button event procedure in the Else block. The code we developed in the previous step is highlighted with a gray background. Let's take a close look at this piece of newly added code to see how it works.

- A. To identify the DataRow to be deleted from the DataTable, the default method FindBy() will be utilized. But this method needs to use faculty\_id as a criterion, so you need to first retrieve the faculty\_id from the Faculty table based on the faculty name selected by the user.
- B. After 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 table in the DataSet.

The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdDelete" and the tab says "Click". The code is written in VB.NET. There are four lines of code labeled A, B, C, and D on the left side of the code editor.

```

Private Sub cmdDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdDelete.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim vbButton As MessageBoxButtons = MessageBoxButtons.YesNo
    Dim FacultyRow As CSE_DEPTDataSet.FacultyRow
    Dim Answer As DialogResult
    Dim strFacultyID As String
    Dim intDelete As Integer

    Answer = MessageBox.Show("You sure you want to delete this record?", "Delete", vbButton)
    If Answer = System.Windows.Forms.DialogResult.Yes Then
        If ComboMethod.Text = "TableAdapter DBDirect" Then
            intDelete = FacultyTableApt.DeleteFaculty(ComboName.Text)
        Else
            strFacultyID = FacultyTableApt.FindFacultyIDByName(ComboName.Text)
            FacultyRow = CSE_DEPTDataSet.Faculty.FindByfaculty_id(strFacultyID)
            FacultyRow.Delete() 'delete data from the DataTable in DataSet
            intDelete = FacultyTableApt.Update(CSE_DEPTDataSet.Faculty)
        End If
    End If
    If intDelete = 0 Then
        MessageBox.Show("Faculty Table Deleting is failed!")
        Exit Sub
    End If
End Sub

```

**Figure 6.10.** Coding for the second method of data deletion.

- 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 coding for the data updating and deleting, let's first complete the coding for the data validation for those data actions.

### 6.2.9 Validate the Data After Data Updating and Deleting

As we mentioned in the previous section, we do not need to develop any code for these data validations but can use the coding developed for the Select button event procedure.

The last job we need to do before we can run the project to test our coding is to add some code to the Select button event procedure to handle the faculty photo issue. The reason is that after a faculty record is updated in the database, for example, a faculty name, the associated faculty photo should also be updated. To make this project simple, we will use a default photo, Default.jpg, to represent any updated faculty photo.

Open the Select button event procedure of the Faculty form, and add the code shown in Figure 6.11 into this event procedure. The code we developed in the previous step is highlighted with a gray background.

The functionality of this piece of coding is:

- If no matched faculty photo is found by the function FindName(), continue to check whether the faculty name has been changed.
- If the faculty name has been changed or updated, which is indicated by a True value of the Boolean variable FacultyNameFlag, the default faculty photo

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim FacultyTableApt As New CSE_DEPTDataSetTableAdapters.FacultyTableAdapter
    Dim strName As String
    strName = FindName(ComboName.Text)
    If strName = "No Match" Then
        If FacultyNameFlag = True Then
            strName = "Default.jpg"
            FacultyNameFlag = False
        Else
            MessageBox.Show("No Matched Faculty Image Found!")
            Exit Sub
        End If
    End If
    PhotoBox.SizeMode = PictureBoxSizeMode.StretchImage
    PhotoBox.Image = System.Drawing.Image.FromFile(strName)
    FacultyTableApt.ClearBeforeFill = True
    FacultyTableApt.FillByFacultyName(CSE_DEPTDataSet.Faculty, ComboName.Text)
    If CSE_DEPTDataSet.Faculty.Count = 0 Then
        MessageBox.Show("No matched faculty found!")
        Exit Sub
    End If
    FacultyName = ComboName.Text      'reserver the old faculty name
End Sub

```

**Figure 6.11.** Modified coding for the Select button event procedure.

Default.jpg is assigned to the photo name string variable strName, and FacultyNameFlag is reset to avoid multiple updates of the same faculty name.

- C. If no faculty name is updated, which means that no matched faculty photo has been found, a warning message is displayed.

Now let's run the project to test our coding for the data updating and deleting.

1. Make sure that the default faculty photo Default.jpg has been stored in the default location – in our case, it is the folder in which our Visual Basic.NET executable file is located (**C:\Chapter 6\AccessUpdateDeleteWizard\bin\Debug**).
2. Click the Start Debugging button to run the project, and enter a suitable username and password, such as jhenry and test, in the LogIn form.
3. Select the Faculty Information item from the Selection form window to open the Faculty form.
4. Click the Select button on the Faculty form to first test the data query functionality.
5. Then click the Insert button to open the Insert Faculty Form window to insert a new Faculty record, as follows:

■ G88765	Faculty ID text box
■ George Stone	Faculty Name text box
■ Associate Professor	Title text box
■ MTC-119	Office text box
■ 750-330-3377	Phone text box
■ University of Florida	College text box
■ gstone@college.edu	Email text box



Figure 6.12. Running status of the Insert Faculty Form window.

Your finished new faculty information window should match the one shown in Figure 6.12.

6. Click the Insert button to insert this new faculty record into the Faculty table.
7. Then click the Back button to return to the Faculty form to perform the data updating, data deletion, and data validation functionalities.
8. Click the drop-down arrow of the ComboName combo box. You will find the newly added faculty name in this box. Select this name and click the Select button to retrieve the newly inserted faculty record and display it in the associated text boxes in the Faculty form.
9. Now update this record by entering the following information:

- |                |                 |
|----------------|-----------------|
| ■ Professor    | Title text box  |
| ■ MTC-219      | Office text box |
| ■ 750-378-5577 | Phone text box  |

10. Keep the content of all other text boxes unchanged, and click the Update button to update this new record in the database. You can use either the TableAdapter DBDirect or the TableAdapter.Update method by selecting it from the Method combo box.
11. To validate this updating, make sure that the faculty name of the updated record is in the ComboName combo box. Then click the Select button to retrieve that updated record. You will find that the faculty record is indeed updated, as shown in Figure 6.13.

To delete this faculty record, do the following.

1. Click the Delete button with either the TableAdapter DBDirect or the TableAdapter.Update method selected. A message box is displayed to ask you to confirm this deletion.
2. Click Yes if you want to delete the record.
3. Validate the deletion by clicking the Select button to try to retrieve that deleted record. What happens after you click the Select button? A message



Figure 6.13. The updated faculty record.

saying “No matched faculty found” is displayed to indicate that that faculty record has been deleted from the database.

One point to be noted is that when you update the faculty name by changing the content of the Faculty Name text box, be sure that you go to the ComboName combo box to select the modified faculty name to perform the data validation by clicking the Select button after you finish updating that record. You need to perform the same operations to delete a record from the database. The key is that the content of the faculty name text box (named Name) may differ from the content of the combo box ComboName, and the former may be an updated faculty name and the latter an old faculty name if an updating of the faculty name has been performed.

Our project is successful!

The completed AccessUpdateDeleteWizard project is located in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

### **6.3 UPDATE AND DELETE DATA FROM AN SQL SERVER DATABASE BY USING THE SAMPLE PROJECT SQLUPDATEDELETEWIZARD**

To save time, we will modify an existing project SQLInsertWizard we developed in the previous chapter to create a new project named SQLUpdateDeleteWizard and use it in this chapter.

1. Open Windows Explorer, and create a new folder, **Chapter 6**, if you have not already created it.
2. Browse to the project SQLInsertWizard in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) using your Internet browser.
3. Copy this project to our new folder, **Chapter 6**.
4. Change the name of the solution and the project from SQLInsertWizard to SQLUpdateDeleteWizard.

5. Double-click SQLUpdateDeleteWizard Project.vbproj to open this project.

On the opened project, perform the following modifications to get the desired project:

1. Go to the Project|SQLUpdateDeleteWizard Project Properties menu to open the project's property window. Change the Assembly name from SQLInsertWizard Project to SQLUpdateDeleteWizard Project and the Root namespace from SQLInsertWizard\_Project to SQLUpdateDeleteWizard\_Project.
2. Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to SQLUpdateDeleteWizard Project. Click OK to close this dialog box.
3. Go to File|Save All to save these modifications.

Now we are ready to develop our GUIs based on our new project SQLUpdateDeleteWizard.

Because of the similarity between this project, SQLUpdateDeleteWizard, and the project AccessUpdateDeleteWizard we developed in the previous section, we will not duplicate all identical parts. Basically, both the GUIs and the coding of SQLUpdateDeleteWizard are identical to those of AccessUpdateDeleteWizard. Perform the following tasks to finish the modifications to this project:

1. Refer to Section 6.2.2.1 to modify the GUI, which includes modifications to the Faculty form window.
2. Refer to Section 6.2.2.2 to finish the data bindings between the text box controls on the Faculty form and the associated columns in the Faculty table in the DataSet.
3. Refer to Section 6.2.3 to finish the coding for data validation before the data updating and deleting.
4. Refer to Section 6.2.4 to finish building the Update and Delete queries.
5. Refer to Sections 6.2.5 and 6.2.6 to finish the coding for updating data by using the TableAdapter DBDirect and TableAdapter.Update() methods.
6. Refer to Sections 6.2.7 and 6.2.8 to finish the coding for deleting data by using the TableAdapter DBDirect and TableAdapter.Update() methods.
7. Refer to Section 6.2.9 to finish the coding for data validation after data updating and deleting.

Some important points related to these modifications are as follows.

To make the modifications in step 1 simple, first you can delete all controls from the Faculty form window in the project SQLUpdateDeleteWizard as follows:

1. Go to Edit|Select All to select all controls.
2. Go to Edit|Delete to delete all controls. Then open the project AccessUpdateDeleteWizard and its Faculty form window.
3. Go to Edit|Select All to select all controls from the Faculty form.
4. Go to Edit|Copy to copy all the controls to the Faculty form window in the project SQLUpdateDeleteWizard.

One point to be noted for this copy operation is that the FacultyBindingSource that belongs to the Faculty form in the project AccessUpdateDeleteWizard will also be copied to the new project. So you need to delete it from the new project after you finish this copy operation.

The modifications in step 4 are to build Update and Delete queries. There are a few differences between building these queries in an Access database and an SQL Server database. First, let's discuss how to build the Update query.

1. Open the Data Sources window and the TableAdapter Query Configuration Wizard.
2. Keep the default selection “Use SQL statements” unchanged, and click Next to go to the next window.
3. Select the UPDATE item from this window since you need to perform a data updating query, and click Next again to continue.
4. Click the Query Builder button since you want to build an updating query. The opened Query Builder window is shown in Figure 6.14.
5. Remove the item “=@Original” in the Filter column along the faculty\_id row, and place a question mark in the Filter column along the name row in the middle pane. Press the Enter key on the keyboard.
6. Clear the Set check box for the faculty\_id row to remove this item. Your finished Update Query Builder should match the one shown in Figure 6.14.
7. Click OK to continue.
8. In the window that opens, remove the SELECT query since you do not need it. Click Next to go to the next window.
9. Change the function name to UpdateFaculty, and click Next.
10. Click Finish to close the Query Builder.

Perform similar operations to build the Delete query, which is shown in Figure 6.15, and name the query function DeleteFaculty.



Figure 6.14. The Update Query Builder.



Figure 6.15. The Delete Query Builder.

To add the Name column to this Delete query as a criterion, click the second row from the middle pane, click the drop-down arrow, and select the Name column from the list. Also remove the item “=@Original” from the Filter column along the faculty\_id row in the middle pane.

Now you can try to run this project to test the data updating and deleting functionalities. You may encounter some errors when you run this project, but do not worry; they are easy to fix.

One possible error is that the modified project’s name will not be recognized by your typed DataSet generation file, CSE\_DEPTDataSet.Designer.vb, in which the complete DataSet definitions are included. This error can occur at lines 2808, 3369, 3953, 4368, and 4741 in that file. This error occurs because we modified this project after the DataSet generation file was created in the previous chapter. To fix this error, just change those items to the current project’s name, SQLUpdateDeleteWizard, in that file.

Another possible error is that some buttons in the Faculty form may not be connected to the associated event procedures, such as the Back, Select, and Insert buttons. To fix this error, just attach the associated event type after each event procedure. For example, for the Select button’s click event procedure, locate this event procedure, move the cursor to the end of the header of this event procedure, and type the keyword Handles followed by the event type cmdSelect.Click. Add the associated event types for all other event procedures that have this kind of error.

A complete project SQLUpdateDeleteWizard is located in the folder **DB-Projects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

## 6.4 UPDATE AND DELETE DATA FROM AN ORACLE DATABASE BY USING THE SAMPLE PROJECT ORACLEUPDATEDELETEWIZARD

To modify data in an Oracle database by using Update and Delete commands is very similar to the project we developed in the last section, and the only difference is the data source to be connected to your applications. Refer to Appendix E to

add and connect the sample Oracle 10g XE database CSE\_DEPT with the Visual Basic.NET application by using Design Tools and Wizards. Also refer to Appendix F to get detailed information on how to use the sample database. All user interfaces and coding are identical to those for the previous project. Appendix E also provides sample coding to use different query methods to perform data accessing and insertion functionalities.

## **PART II DATA UPDATING AND DELETING WITH RUNTIME OBJECTS**

To update or delete data from a database using the runtime object method is a flexible and professional way to perform data modification in the Visual Basic.NET environment. Compared with the method we discussed in Part I, in which Visual Basic.NET design tools and wizards were utilized to update or delete data from a database, the runtime object method provides more sophisticated techniques to do this job efficiently and conveniently even when a more complicated coding job is needed. Relatively speaking, the methods we discussed in Part I are easy to learn and code, but some limitations exist for those methods. First, each TableAdapter can 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 perform only a single query such as data updating or deleting. Third, after the query function is built, no modifications can be made to that function dynamically, which means that the only time you can modify that query function is either before or after the project runs. In other words, you cannot modify that query function during the time the project runs.

To overcome these shortcomings, we will discuss how to update or delete data by using the runtime object method in this part.

Basically, you need to use the TableAdapter to perform data actions in the database if you develop your applications using Visual Basic.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 object method.

### **6.5 THE RUNTIME OBJECT METHOD**

We provided a very detailed introduction to and discussion of the runtime object method in Section 4.16 in Chapter 4. For your convenience, we will highlight some important points and general methodology of this method here and provide some key notes on using this method to perform data updating and deleting in a database.

As you know, ADO.NET provides different classes to help users develop professional data-driven applications by using different methods to perform specific data actions such as updating and deleting data. Among them, two popular methods are as follows:

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 by using the Command object, and then call the Command object's ExecuteNonQuery() method to update or delete records from the database. You can even assign the built Command object to the UpdateCommand or DeleteCommand properties of a DataAdapter and call the ExecuteNonQuery() method from the UpdateCommand or DeleteCommand property.

The first method is to use the so-called DataSetDataAdapter 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 different properties of a DataAdapter, such as 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 make for each table in the DataSet. For example, if you delete rows from a table in the DataSet and then call the 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 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 provide any further discussion of this method in this part.

The second method enables 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 and 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 to that query for the newly created Command object. You can update or delete data from 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 will provide three sample projects named SQLUpdateDeleteRTOBJECT, AccUpdateDeleteRTOBJECT, and OracleUpdateDeleteRTOBJECT to illustrate how to update or delete records from three different databases by using the runtime object method. Because of the coding similarity between these three databases, we will concentrate on updating and deleting data from the SQL Server database by using the sample project SQLUpdateDeleteRTOBJECT first and then illustrate the coding differences between these databases by using the real codes for the other two sample projects.

Let's first develop the sample project SQLUpdateDeleteRTOBJECT to update and delete data from the SQL Server database by using the runtime object method. Recall that in Sections 4.18.4–4.18.8 in Chapter 4, we discussed how to select data for the Faculty, Course, and Student form windows by 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, stored procedures are used to perform the runtime data query.

Similarly, in this part, we will divide this discussion into two sections:

1. Update and delete data from the Faculty table in the Faculty form window by using the runtime object method.
2. Update and delete data from the Faculty table in the Faculty form by using the runtime stored procedure method.

In order to avoid duplication of coding, we will modify the existing project named SQLInsertRTOBJECT we developed in Chapter 5 to create our new project SQLUpdateDeleteRTOBJECT used in this section.

## **6.6 UPDATE AND DELETE DATA FROM AN SQL SERVER DATABASE BY USING RUNTIME OBJECTS**

1. Open Windows Explorer, and create a new folder **Chapter 6** if you have not already created it.
2. Open your Internet browser and browse to the folder **DBProjects\Chapter 5** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and copy the project SQLInsertRTOBJECT to the new folder **C:\Chapter 6**.
3. Change the name of the project from SQLInsertRTOBJECT to SQLUpdateDeleteRTOBJECT.
4. Double-click SQLUpdateDeleteRTOBJECT.vbproj to open this project.

On the opened project, perform the following modifications to get the desired project:

1. Go to the Project|SQLUpdateDeleteRTOBJECT Properties menu to open the project's property window.
2. Change the Assembly name from SQLInsertRTOBJECT to SQLUpdateDeleteRTOBJECT and the Root namespace from SQLInsertRTOBJECT to SQLUpdateDeleteRTOBJECT, respectively.
3. Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to SQLUpdateDeleteRTOBJECT.
4. Click the OK to close this dialog box.
5. Go to File|Save All to save those modifications.

Now we are ready to develop our GUIs based on our new project SQLUpdateDeleteRTOBJECT.

### **6.6.1 Update Data in the Faculty Table for the SQL Server Database**

Let's first discuss how to update data in the Faculty table for the SQL Server database. To update data in the Faculty table, we do not need to add any new Windows forms, and we can use the Faculty form as the user interface. We need to perform the following four steps to complete this new project:

1. Modify the current Faculty form window.
2. Modify the original coding in the Faculty form and the Insert Faculty form.
3. Develop the code to update data.
4. Validate the data updating.

First, we need to modify the Faculty form to make it suitable for our data updating.

#### 6.6.1.1 Modify the Faculty Form Window

Recall that when we developed the Faculty form for the project SQLInsertRTOBJ in the previous chapter, five labels were developed in that form to store faculty information. In order to update records in the Faculty table, we need a way to enter new faculty information into some controls and update the record later. The text box is a good candidate to receive and store a piece of new faculty information. Therefore, the first job we need to do is to replace those five labels with five text box controls and increase the number of text boxes to seven since, to update a record in the Faculty table, seven pieces of information are needed.

A good, simple way to modify this Faculty form window is to first remove all controls from the current Faculty form window and then copy all controls from the Faculty form window in the project SQLUpdateDeleteWizard we developed in this chapter to the current Faculty form window. To do this, take the following steps:

1. First, remove all controls from the current Faculty form window by clicking Edit|Select All and Edit|Delete.
2. Next, open the project SQLUpdateDeleteWizard and its Faculty form window.
3. Select Edit|Select All and then Edit|Copy to copy all controls from that Faculty form window.
4. Now open our current Faculty form window and go to Edit|Paste to paste the controls into our current form.

Your finished Faculty form window should match the one shown in Figure 6.16.

One point to note is that when you perform this copy operation, the object FacultyBindingSource, which belongs to the project SQLUpdateDeleteWizard, will also be copied into the Faculty form. Remove this object since you do not need it.

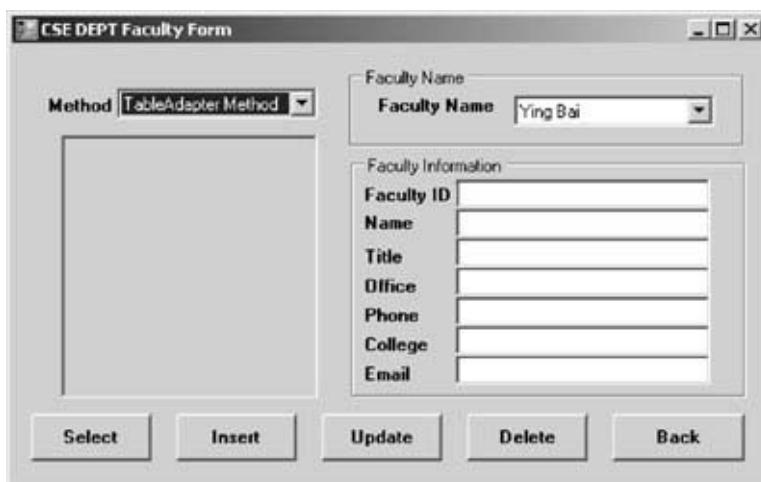


Figure 6.16. The modified Faculty form window.

### 6.6.1.2 Modify the Original Coding in the Faculty Form

This modification should be divided into two parts. The first part is to modify the coding in the Faculty form, and the second part is to modify the coding for the Insert Faculty Form window. Since there is no change in the coding for the Insert Faculty form, we will concentrate on the modification to the Faculty form only.

#### 6.6.1.2.1 Modify the Coding for the Faculty Form

The following modifications are needed for this part, and all modifications are highlighted in bold:

1. Replace the form-level label array FacultyLabel() with a form-level text box array FacultyTextBox(), and increase the size of this array to 7 since you need to use this array to store seven pieces of faculty information. Refer to step **D** in Figure 6.17 for these modifications.
2. Add three form-level variables, FacultyName (String), FacultyNameFlag (Boolean), and SelectFlag (Boolean), to the current Faculty form. You need to use these variables to identify whether the faculty name has been updated. If yes, you need to remove the old faculty name from the ComboName combo box control and add the new faculty name that is located in the txtName text box control into that combo box control. FacultyName is used to temporarily store the old faculty name. Refer to steps **A**, **B**, and **C** in Figure 6.17 for these modifications.

```

FacultyForm (Declarations)
Public Class FacultyForm
Private InsertFaculty As New InsertFacultyForm
A Private FacultyNameFlag As Boolean
B Private FacultyName As String
C Private SelectFlag As Boolean 'indicate a select query occurs
D Private FacultyTextBox(6) As TextBox 'Faculty table has 7 columns, we used all of them
Private Sub FacultyForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
If LogInForm.sqlConnection.State <> ConnectionState.Open Then
    E LogInForm.sqlConnection.Open()
End If
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("DataReader Method")
ComboMethod.SelectedIndex = 0
End Sub
F Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Dim cmdString1 As String = "SELECT faculty_id, name, office, phone, college, title, email FROM Faculty"
Dim cmdString2 As String = "WHERE name LIKE @facultyName"
G .....
SelectFlag =True

```

Figure 6.17. Modifications to the Faculty form – 1.

3. Replace all FacultyLabel array variables with the FacultyTextBox in the user-defined subroutines FillFacultyTable() and FillFacultyReader(). Also change the class name from Label to TextBox in those subroutines. Refer to steps B to D and F to H in Figure 6.18 for these modifications.
4. Change the upper bound of the FacultyTextBox array from the original value of 4 to 6 for both user-defined subroutines FillFacultyTable() and FillFacultyReader() since you need to store seven pieces of faculty information in that array. Refer to steps **A** and **E** in Figure 6.18 for these modifications.
5. Change the nominal variable of the user-defined subroutine MapFacultyTable() from fLabel to fText, and modify the coding for this subroutine as shown in step **I** in Figure 6.18.
6. Replace the two statements MessageBox.Show() and Exit Sub with the statement LogInForm.sqlConnection.Open() for the If block in the FacultyForm\_Load event procedure. This is because you need to connect the database with your application to perform data updating or deleting if this connection is broken. Refer to step **E** in Figure 6.17 for this modification.
7. Add two more query items, faculty\_id and name, to the query statement string cmdString1 in the cmdSelect\_Click event procedure because now you need to query seven pieces of faculty information from the database to validate the data updating. Refer to step **F** in Figure 6.17 for this modification.
8. Add a statement to set SelectFlag to True to indicate that the Select button's Click event procedure has been executed. This flag is used to distinguish the TextChanged event of the Faculty Name text box between the Update and Select buttons' Click events. Because both the Select and Update buttons' Click events can trigger the TextChanged event for the Faculty Name text box, you need to update the ComboName combo box's content by adding an updated faculty name and removing the old faculty name only when the Update button is clicked. For the TextChanged event triggered by the Select button's Click event, you do not need to update the ComboName combo box control. Refer to step **G** in Figure 6.17 for this modification.
9. Modify the coding for the SelectedIndexChanged event procedure of the ComboName combo box control; the resultant coding is shown in step **J** in Figure 6.18.
10. Add one more condition, FacultyNameFlag=True, to the If block in the user-defined subroutine ShowFaculty(). The purpose of this condition is to check whether the Faculty Name has been updated. If yes, a default faculty photo is displayed for that updated faculty name. Refer to step **K** in Figure 6.18 for this modification.
11. After step 9, the FacultyNameFlag is reset to avoid multiple identical operations for the updated faculty. Refer to step **L** in Figure 6.18 for this modification.

Another important point to note is that some event types may be lost for some command buttons' event procedures, such as Select.Click, Back.Click, Insert.Click, the combo box's Drop-Down, and the combo box's SelectedIndexChanged, in the code window of the Faculty form since we performed a copy operation for this form

```

FacultyForm ▼ (Declarations) ▼

Private Sub FillFacultyTable(ByVal FacultyTable As DataTable)
    Dim pos1 As Integer = 0
    Dim column As DataColumn
    Dim row As DataRow
    A For pos2 As Integer = 0 To 6           'Initialize the object array
    B   FacultyTextBox(pos2) = New TextBox()
    Next pos2
    C Call MapFacultyTable(FacultyTextBox)
    For Each row In FacultyTable.Rows
        For Each column In FacultyTable.Columns
            D   FacultyTextBox(pos1).Text = row(column)
            pos1 = pos1 + 1
        Next
    Next
    End Sub

Private Sub FillFacultyReader(ByVal FacultyReader As SqlDataReader)
    Dim intIndex As Integer
    E For intIndex = 0 To 6           'Initialize the object array
    F   FacultyTextBox(intIndex) = New TextBox()
    Next intIndex
    G Call MapFacultyTable(FacultyTextBox)
    While FacultyReader.Read()
        H   For intIndex = 0 To FacultyReader.FieldCount - 1
            FacultyTextBox(intIndex).Text = FacultyReader.Item(intIndex).ToString()
        Next intIndex
    End While
    End Sub

I Private Sub MapFacultyTable(ByRef fText As Object)
    fText(0) = txtID           'The order must be identical
    fText(1) = txtName         'with the real order in the query string cmdString
    fText(2) = txtOffice
    fText(3) = txtPhone
    fText(4) = txtCollege
    fText(5) = txtTitle
    fText(6) = txtEmail
End Sub

Private Sub ComboName_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
    If InsertFaculty.FacultyName <> String.Empty Then
        ComboName.Items.Add(InsertFaculty.FacultyName)
        InsertFaculty.FacultyName = String.Empty
    End If
    J If FacultyNameFlag = True Then
        ComboName.Items.Remove(FacultyName)
        ComboName.Items.Add(txtName.Text)
        FacultyNameFlag = False
    End If
End Sub

Private Sub ShowFaculty(ByVal fName As String)
    .....
    K Else
        If InsertFaculty.InsertFacultyFlag = True Or FacultyNameFlag = True Then
            PhotoBox.Image = System.Drawing.Image.FromFile("Default.jpg")
        L If FacultyNameFlag = True Then
            FacultyNameFlag = False
        End If
        Exit Sub
    .....

```

Figure 6.18. Modifications to the Faculty form – 2.

in Section 6.5.1.1 in this chapter. To fix these bugs, just open the code window of the Faculty form and add the associated event types for each event procedure. An example of adding the event type for the Select button's Click event procedure is shown below (the added part is highlighted in bold):

---

```
Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles cmdSelect.Click
```

---

Well, a lot of modifications have been made in this part. But that is required since we can save a lot of time when we develop the next project, OracleUpdateDeleteRTOBJECT, by just making a little modification to the current project.

Now let's develop the coding for the data updating and deleting.

#### 6.6.1.3 Develop Code to Update Data

As we mentioned in the previous sections, to update or delete data using this project, you can insert a new record into the database and then that inserted record can be updated or deleted completely (including updating the primary key). Another way is to update an existing record from the sample database without touching the primary key. To update or delete an existing record from the related tables, you must follow the three steps listed in Section 6.1.1.

Open the Update button's Click event procedure on the Faculty form by double-clicking the Update button on the Faculty form window, and enter the code shown in Figure 6.19 into this event procedure.

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

- A. The Update query string is defined first at the beginning of this procedure. All seven columns in the Faculty table are input parameters. The dynamic parameter @Param1 represents the old faculty name, which means the faculty name that has not been 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 data returned by the ExecuteNonQuery() method.
- C. Before the data updating occurs, you need to reserve the old faculty name that is located in the ComboName combo box control since you need to remove this old faculty name and add the updated faculty name into this control later if the faculty name is updated.
- D. The Command object is initialized and built using the connection object and parameter object.
- E A user-defined subroutine UpdateParameters() is called to add all updated parameters into the Command object.
- F. 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.
- G. The Command object is released after this data updating.
- H. The value returned by the ExecuteNonQuery() method is equal to the number of rows that have been updated in the Faculty table. A zero means that

```

Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click
    Dim cmdString As String = "UPDATE Faculty SET faculty_id = @faculty_id, name = @name, office = @office, " &
        "phone = @phone, college = @college, title = @title, email = @email " & _
        "WHERE (name LIKE @Param1)"
    Dim sqlCommand As New SqlCommand
    Dim intUpdate As Integer
    FacultyName = ComboName.Text
    sqlCommand.Connection = LogInForm.sqlConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    UpdateParameters(sqlCommand)
    intUpdate = sqlCommand.ExecuteNonQuery()
    sqlCommand.Dispose()
    sqlCommand = Nothing
    If intUpdate = 0 Then
        MessageBox.Show("The data updating is failed")
        Exit Sub
    End If
End Sub

Private Sub UpdateParameters(ByRef cmd As SqlCommand)
    cmd.Parameters.Add("@faculty_id", SqlDbType.Char).Value = txtID.Text
    cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text
    cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text
    cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text
    cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text
    cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text
    cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text
    cmd.Parameters.Add("@Param1", SqlDbType.Char).Value = ComboName.Text
End Sub

Private Sub txtName_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles txtName.TextChanged
    If SelectFlag = True Then
        SelectFlag = False
    Else
        FacultyNameFlag = True
    End If
End Sub

```

**Figure 6.19.** Coding for the data updating operation.

no row has been updated, an error message is displayed, and the procedure is exited.

- I. The detailed coding for the user-defined subroutine `UpdateParameters()` is shown in this step. Seven pieces of new faculty information are assigned to the associated columns in the Faculty table.
- J. One point to note is that two controls on the Faculty form can be used to store the Faculty Name information, the `ComboName` combo box control and the `txtName` text box control. The difference between these two controls is that the former is used to store the old faculty name, and the latter is used to hold the updated faculty name. The dynamic parameter `@Param1` in the `UPDATE` query string represents the old faculty name, so you must assign the old faculty name to that parameter. This is very important for this update query; the project may encounter errors if you use an updated faculty name as this parameter, because the project cannot find the updated name in the database if that faculty record has not been updated.

K. The coding for the TextChanged event procedure of the Faculty Name text box is very simple. If a TextChanged event occurs, which means that a new faculty name is entered into this text box and the user wants to update the faculty name, before you set the FacultyNameFlag, you must check and confirm whether this event is triggered by the Update button's or the Select button's Click event procedure. You take care of only the event triggered by the Update button's Click event procedure because that means an update occurs. The FacultyNameFlag is set only in that situation. Also, you reset the SelectFlag if it is set, to avoid multiple duplicated operations.

At this point, we have finished the coding for the data updating operation for the Faculty table. Next, let's take care of data validation after this data updating to confirm that the data updating is successful.

#### 6.6.1.4 Validate the Data Updating

We do not need to add a new form window to perform this data validation, and we can use the Faculty form to perform this job. By clicking 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. A small change to the coding of the Select button's Click event procedure is shown in step **G** in Figure 6.17.

Before we can run the project to test the data updating functionality, we should complete the coding for data deletion first.

#### 6.6.2 Delete Data from the Faculty Table for an SQL Server Database

As we mentioned in the previous section, to delete data from a database, we have two ways to go. One way is to first insert a new record into the Faculty table and then delete that newly inserted data since our sample database CSE\_DEPT is a relational database and all tables have been related by different keys. Another way is to delete existing data from our related tables by performing the following two steps:

1. First, delete records from the child tables (LogIn and Course tables).
2. Second, delete the record from the parent table (Faculty table).

The topic of deleting existing data from our related tables by calling stored procedures is discussed in Sections 6.8.4 and 7.7.4. An example of deleting a faculty member from the Faculty, LogIn, and Course tables is used to illustrate how to perform data deletion among related tables. The databases we will use are the Oracle database (Section 6.8.4) and the SQL Server database (Section 7.7.4) since they are very popular and are applied in most actual applications. In this section we will concentrate only on deleting a newly inserted record from our database.

The data deletion functionality can be performed by using the Delete button's Click event procedure in the Faculty form window. So the main coding for this functionality is developed inside that event procedure.

```

Private Sub cmdDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdDelete.Click
    Dim cmdString As String = "DELETE FROM Faculty WHERE (name LIKE @Param1)"
    Dim vbButton As MessageBoxButtons = MessageBoxButtons.YesNo
    Dim sqlCommand As New SqlCommand
    Dim Answer As DialogResult
    Dim intDelete As Integer
    Answer = MessageBox.Show("You sure you want to delete this record?", "Delete", vbButton)
    If Answer = System.Windows.Forms.DialogResult.Yes Then
        sqlCommand.Connection = LogInForm.sqlConnection
        sqlCommand.CommandType = CommandType.Text
        sqlCommand.CommandText = cmdString
        sqlCommand.Parameters.Add("@Param1", SqlDbType.Char).Value = ComboName.Text
        intDelete = sqlCommand.ExecuteNonQuery()
        sqlCommand.Dispose()
        sqlCommand = Nothing
    End If
    If intDelete = 0 Then
        MessageBox.Show("The data Deleting is failed")
        Exit Sub
    End If
    For intDelete = 0 To 6      'Clean up the Faculty textbox array
        FacultyTextBox(intDelete).Text = String.Empty
    Next intDelete
    End If
End Sub

```

Figure 6.20. Coding for the data deleting query.

### 6.6.2.1 Develop Code to Delete Data

Open the Delete button's Click event procedure by double-clicking the Delete button from the Faculty form window, and enter the code shown in Figure 6.20 into this event procedure.

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

- A. First, the deleting query string is declared at the beginning of this procedure. The only parameter is the faculty name. Although the primary key of the Faculty table is faculty\_id, in order to make it convenient to the user, the faculty name is used as the criterion for this data deleting query. A potential problem of using the name as a criterion in this query is that no duplicated faculty name can be inserted into the Faculty table for this application. In other words, each faculty name in the Faculty table must be unique. A solution to this problem is that we can use the simplified faculty\_id as the criterion for the data deleting query in the future.
- B. A MessageBox button's object is created, and this object is used to display both buttons in the MessageBox, Yes and No, when the project runs.
- C. Some useful components and local variables are declared here, too. The data type of the variable Answer is DialogResult, but you can use an integer instead.
- D. When the Delete button is clicked as the project runs, first a message box 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 message box is Yes, then the deleting operation is processed. The Command object is initialized and built by using the

Connection object and the command string defined at the beginning of this procedure.

- F. The dynamic parameter @Param1 is replaced by the real parameter, the faculty name stored in the ComboName combo box. A key point to note is that you must use the faculty name stored in the combo box control, which is an old name, and not the faculty name stored in the Faculty Name text box, since that is an updated name.
- G. The ExecuteNonQuery() method of the Command class is called to execute the data deleting query on the Faculty table. The running result of calling this method is stored in the local variable intDelete.
- H. The Command object is released after data deletion.
- I. The value returned by the ExecuteNonQuery() method is equal to the number of rows that have been successfully deleted from the Faculty table. If a zero is returned, which means that no row has been deleted from the Faculty table and this data deletion failed, an error message is displayed and the procedure is exited.
- J. After the data deletion is done, all faculty information stored in the seven text boxes should be cleaned up. A For loop is used to finish this cleaning job.

Finally, let's take care of the coding to validate the data deleting query.

#### 6.6.2.2 Validate the Data Updating and Deleting

As we did for the validation of the data updating in the last section, we do not need to create any new form window to do this validation, and we can use the Faculty form to perform it.

Now let's run the project to test both the data updating and data deletion operations.

1. Before you 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 6\SQL-UpdateDeleteRTOBJECT\bin\Debug**.
2. Click the Start Debugging button to start the project, and enter a suitable username and password in the LogIn form.
3. Select the item Faculty Information from the Selection form to open the Faculty form window.
4. Click the Insert button to open the Insert Faculty Form window to first insert a new faculty record, which is shown below, into the Faculty table.

■ P28262	Faculty ID text box
■ Peter Jones	Faculty Name text box
■ Associate Professor	Title text box
■ MTC-228	Office text box
■ 750-550-2266	Phone text box
■ University of Miami	College text box
■ pjones@college.edu	Email text box



Figure 6.21. Running status of the Insert Faculty Form window.

Your finished new faculty information window should match the one shown in Figure 6.21.

5. Click the Insert button to insert this new record into the Faculty table.
6. Click the Back button to return to the Faculty form to validate this new data insertion.
7. Click the drop-down arrow on the ComboName combo box control. You will find that that newly inserted faculty name Peter Jones is there.
8. Select the new name, and click the Select button to retrieve this newly inserted record and display it in this form.
9. To test the data updating functionality, perform the following modifications to this new faculty record:

- |                |                       |
|----------------|-----------------------|
| ■ Peter Steff  | Faculty Name text box |
| ■ Professor    | Title text box        |
| ■ MTC-358      | Office text box       |
| ■ 750-378-5577 | Phone text box        |

10. Click the Update button to update this record in the Faculty table.
11. To confirm this data updating, go to the ComboName combo box control, and try to find the updated faculty name in this combo box. Of course, you can find this updated name.
12. In order to test this data updating, first select another faculty name from the box, and click the Select button to show all information for that faculty member.
13. Then go to the combo box again, select the updated faculty name from the box, and click the Select button to retrieve the updated information for that faculty member. Immediately you will find that all updated information related to that faculty name is displayed in this form. This means that our data updating is successful. Your updated faculty information window is shown in Figure 6.22.



Figure 6.22. The updated faculty information window.

Now let's test the data deletion functionality by clicking the Delete button to try to delete this updated faculty record from the Faculty table. Click Yes to confirm, and all updated faculty information stored in the seven text boxes is gone. Is our data deletion successful? To answer this question, click the Select button again to try to retrieve that updated faculty information from the Faculty table. What happens after you click the Select button? A message "No matched faculty found" is displayed, which means that the updated faculty information has been successfully deleted from the Faculty table. Yes, our data deletion is successful.

A completed project SQLUpdateDeleteRTOBJECT is located in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

## 6.7 UPDATE AND DELETE DATA FROM AN ORACLE DATABASE BY USING RUNTIME OBJECTS

Because of the coding similarity between SQL Server and Oracle databases for data updating and deleting, we will show only coding that is different from the coding for an SQL Server database. The main differences between an SQL Server database and an Oracle database are the query strings for data deletion and updating. In this section, we will concentrate on these query strings.

First, let's modify an existing project to create our new project. We want to modify the project SQLUpdateDeleteRTOBJECT we developed in the last section to create the new project OracleUpdateDeleteRTOBJECT that we will use in this section. Open SQLUpdateDeleteRTOBJECT and perform the following operations to make it a new project:

1. Open Windows Explorer, and create a new folder, **Chapter 6**, if you have not already created it.

2. Open your Internet browser and browse to the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and copy the project SQLUpdateDeleteRTOBJECT to the new folder **C:\Chapter 6**.
3. Change the name of the project from SQLUpdateDeleteRTOBJECT to OracleUpdateDeleteRTOBJECT.
4. Double-click OracleUpdateDeleteRTOBJECT.vbproj to open this project.

On the opened project, perform the following modifications to get the desired project:

1. Go to the Project|OracleUpdateDeleteRTOBJECT Properties menu to open the project's property window. Change the Assembly name from SQLUpdateDeleteRTOBJECT to OracleUpdateDeleteRTOBJECT and the Root namespace from SQLUpdateDeleteRTOBJECT to OracleUpdateDeleteRTOBJECT.
2. Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to OracleUpdateDeleteRTOBJECT. Click OK to close this dialog box.
3. Go to File|Save All to save those modifications.

Now we are ready to develop the code for our new project OracleUpdateDeleteRTOBJECT.

We can use all GUIs from this modified project, and the only modifications we need to make are in the coding for each form window. Basically, we need to perform the following modifications on the coding:

1. Add the Oracle namespace reference to the project.
2. Modify the Imports commands.
3. Modify the connection string in the LogIn form.
4. Modify the SELECT query string for the LogIn button's Click event procedure in the LogIn form.
5. Modify the SELECT query string for the Select button's Click event procedure in the Faculty form.
6. Modify the UPDATE query string for the Update button's Click event procedure in the Faculty form.
7. Modify the DELETE query string for the Delete button's Click event procedure in the Faculty form.
8. Modify the parameters' names for the UPDATE and DELETE command objects in the Faculty form.
9. Modify the two SELECT query strings for the Select button's Click event procedure and the SelectedIndexChanged event procedure of the Course list box in the Course form.
10. Modify the INSERT query string for the Insert button's Click event procedure in the Insert Faculty form.
11. Modify the parameters' names for the INSERT command object in the Insert Faculty form.
12. Modify all prefixes for all Oracle classes and objects used in this project.

Well, it looks like there are many modifications that we need to do for this project, but it is easy to handle them. Let's begin our first modification.

### **6.7.1 Add the Oracle Namespace Reference and Modify the Imports Command**

Open the project and go to the Solution Explorer window, right-click the project, and select the Add Reference item to open the Add Reference dialog box. Browse along the list until you find the item System.Data.OracleClient. Select it and click OK to add this reference to our project.

Open the code windows of the following forms from the current project:

- LogIn
- Faculty
- Course
- Insert Faculty Form

Replace the Imports command Imports System.Data.SqlClient, which is located on the second line at the top of each code window, with the new command Imports System.Data.OracleClient for all the above four form windows. Since we will not use the Student and SP forms for this project, leave them unchanged.

### **6.7.2 Modify the Connection String and Query String for the LogIn Form**

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.

#### **6.7.2.1 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=system;" + "Password=reback"
```

---

Also change the prefixes of all data classes from Sql to Oracle and the prefixes of all data objects from sql to ora, respectively.

#### **6.7.2.2 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 and the prefixes of all data objects from sql to ora, respectively. Change the two dynamic parameters' names from @Param1 to Param1 and from @Param2 to Param2, respectively.

### **6.7.2.3 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 and the prefixes of all data objects from sql to ora, respectively. Change the two dynamic parameters' names from @name to name and from @word to word, respectively.

## **6.7.3 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.

### **6.7.3.1 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, name, office, phone,  
college, title, email FROM Faculty"
```

```
Dim cmdString2 As String = "WHERE name = :facultyName"
```

---

Also change the prefixes of all data classes from Sql to Oracle and the prefixes of all data objects from sql to ora, respectively. Change the dynamic parameter's name from @facultyName to facultyName.

### 6.7.3.2 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_id = :faculty_id, " &
    "name = :name, office = :office, phone = :phone, college = :college, " &
    "title = :title, email = :email WHERE (name = :Param1)"
```

---

Change the prefixes of all data classes from Sql to Oracle and the prefixes of all data objects from sql to ora, respectively. Also modify the data types and the names of the dynamic parameters inside the UpdateParameters() subroutine as follows:

- Change the data type for all parameters from SqlDbType to OracleType.
- Remove the @ symbol before all parameters' names.

### 6.7.3.3 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
    (name = :Param1)"
```

---

Change the prefixes of all data classes from Sql to Oracle and the prefixes of all data objects from sql to ora, respectively. Also change the dynamic parameter's name from @Param1 to Param1 and the data type from SqlDbType to OracleType.

## 6.7.4 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 List box's SelectedIndexChanged event procedure.

### 6.7.4.1 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 cmdString1 As String = "SELECT Course.course_id, Course.course
FROM Course, Faculty"
Dim cmdString2 As String = "WHERE (Course.faculty_id = Faculty.
faculty_id) AND (Faculty.name = :name)"
```

---

Change the prefixes of all data classes from Sql to Oracle and the prefixes of all data objects from sql to ora, respectively. Also change the dynamic parameter's name from @name to name and the data type from SqlDbType to OracleType.

Another modification is to change the method GetSqlString() to GetOracleString() in the user-defined subroutine FillCourseReader().

#### **6.7.4.2 Modify the SELECT Query String for the CourseList Event Procedure**

Open the Course List box's SelectedIndexChanged event procedure, and change the query string to

---

```
Dim cmdString1 As String = "SELECT course_id, 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 and the prefixes of all data objects from sql to ora, respectively. Also change the dynamic parameter's name from @courseid to courseid and the data type from SqlDbType to OracleType.

#### **6.7.5 Modify the Query Strings for the Insert Faculty Form**

Open the Insert button's Click event procedure from the Insert Faculty Form window, and change the query string to

---

```
Dim cmdString As String = "INSERT INTO Faculty (faculty_id, name, office, "&
"phone, college, title, email) VALUES(:faculty_id,:name,:office,:phone, " &
":college,:title,:email)"
```

---

Also modify the data types and the names of the dynamic parameters inside the InsertParameters() subroutine as follows:

- Change the data type of the passed argument command object from SqlCommand to OracleCommand.
- Change the data type for all parameters from SqlDbType to OracleType.
- Remove the @ symbol before all the parameters' names.

#### **6.7.6 Other Modifications**

Change the prefixes of all data classes from Sql to Oracle and the prefixes of all data objects from sql to ora, respectively. These modifications include the following procedures:

- The Cancel button's Click event procedure in the LogIn form
- The Form\_Load event procedure of the Faculty form

- The Form\_Load event procedure of the Course form
- The Form\_Load event procedure of the Insert Faculty form
- The Exit button's Click event procedure in the Selection form

These modifications also include the data type of nominal arguments passed into either subroutines or functions in this project.

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 functionalities. Click the Start Debugging button to run the project. You may encounter some debug errors that are introduced by some old coding in the Student or the SP form windows. Just comment out those lines at this moment because we will not use this coding in this project. Enter a suitable username and password, such as jhenry and test, to the LogIn form, and select the Faculty Information item from the Selection form to open the Faculty form window. Click the Insert button to insert a new faculty record into the database as shown:

- |                         |                       |
|-------------------------|-----------------------|
| ■ P33431                | Faculty ID text box   |
| ■ Peter Steff           | Faculty Name text box |
| ■ Associate Professor   | Title text box        |
| ■ MTC-235               | Office text box       |
| ■ 750-378-1130          | Phone text box        |
| ■ University of Florida | College text box      |
| ■ Psteff@college.edu    | Email text box        |

Click the Insert button to insert this new faculty record into the Faculty table in the database. Click the Back button to return to the Faculty form window to perform the data updating operation.

Click the drop-down arrow on the ComboName combo box control, and you will find that the newly inserted faculty name is there. Select it and click the Select button to retrieve this new record from the database and display it in this form, which is shown in Figure 6.23.



Figure 6.23. Running status of the Faculty form.

Change the faculty information as follows:

- Peter Jones Faculty Name text box
- Professor Title text box
- MTC-335 Office text box
- 750-330-5555 Phone text box

Click the Update button to update this record in the Faculty table in the database.

To confirm this data updating, click the drop-down arrow on the ComboName combo box control. First, select any other faculty name from the list, and click the Select button to show the information for that faculty member. Then select the updated faculty member from the ComboName combo box control, and click the Select button to try to retrieve this updated faculty information and display it in this form. Immediately you will see that the faculty information has been updated and displayed, as shown in Figure 6.24. Our data updating is successful.

Now let's test our data deletion functionality. Keep the updated faculty name unchanged in the ComboName combo box control, and click the Delete button to try to delete it from the Faculty table in the database. Click Yes to confirm, and you will find that all information related to that faculty member is removed from all text boxes. To confirm the data deletion, click the Select button to try to retrieve that deleted record from the Faculty table. The message "No matched faculty found" is displayed to indicate that that piece of faculty information has been deleted from the database. Yes, our data deletion is also successful.

A complete project OracleUpdateDeleteRTOBJECT can be found in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

## 6.8 UPDATE AND DELETE DATA FROM A DATABASE BY USING STORED PROCEDURES

As we mentioned in the previous sections, updating data among related tables is a very challenging task. But the real issue is that it is unnecessary to update the

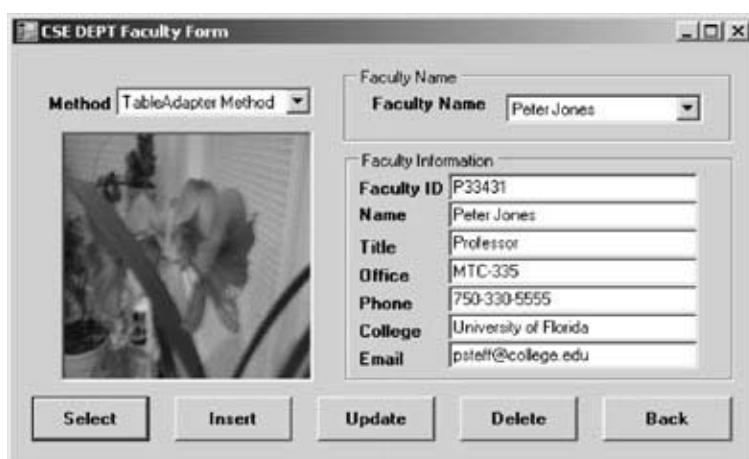


Figure 6.24. Confirmation of the data updating operation.

primary key, or 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 to update that record, because generally the primary key, or faculty\_id, is good for the lifetime of the database in actual applications. Therefore, based on this analysis, in the following sections, we will perform data updating for all columns in the Faculty table except faculty\_id.

To delete records from related tables, we need to perform two steps: First, we need to delete records from the child tables, and second, we need to 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 records related to the record to be deleted from the Faculty table from the LogIn and Course tables (child tables), and then we can delete the record from the Faculty table (parent table).

We will divide this discussion into four parts based on the three types of databases we used in this book: Access, SQL Server, and Oracle.

To save time and space, we will not duplicate any project and will modify the existing projects to create our desired projects.

### 6.8.1 Update Data in an Access Database by Using Stored Procedures

We want to modify the project SQLUpdateDeleteRTOBJECT to create our desired project AccessUpdateRTOBJECTSP to discuss updating data in the Faculty table by using stored procedures for the Access database.

Copy the project SQLUpdateDeleteRTOBJECT and change the name. We will perform the following tasks to finish this project:

1. Modify the existing project to access the Access database.
2. Create stored procedures in the Access database.
3. Call the stored procedures to update the faculty information.
4. Confirm the faculty information updating.

Now let's start with the first part, modifying the project.

#### 6.8.1.1 Modify the Existing Project

Open Windows Explorer, and create a new folder, **Chapter 6**, if you have not already created it. Then open your Internet browser and browse to the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and copy the project SQLUpdateDeleteRTOBJECT to the new folder **C:\Chapter 6**. Change the name of the project from SQLUpdateDeleteRTOBJECT to AccessUpdateRTOBJECTSP. Double-click AccessUpdateRTOBJECTSP.vbproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to the Project|AccessUpdateRTOBJECTSP Properties menu to open the project's property window. Change the Assembly name from SQLUpdateDeleteRTOBJECT to AccessUpdateRTOBJECTSP and the Root namespace from SQLUpdateDeleteRTOBJECT to AccessUpdateRTOBJECTSP.

- Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to AccessUpdateRTOBJECTSP. Click OK to close this dialog box.

Go to File|Save All to save these modifications. Now we are ready to modify our codes based on our new project AccessUpdateRTOBJECTSP.

The code modifications include the following parts:

1. Change the Imports commands.
2. Change the connection string in the LogIn form.
3. Change the query strings for the LogIn button's event procedure in the LogIn form.
4. Change the query strings for the Select and Update buttons' event procedures in the Faculty form.
5. Change the prefixes of all data classes from Sql to OleDb and the prefixes of all data objects from sql to acc, respectively, for the LogIn, Faculty, and Selection forms.

Let's start with the first modification – modify the Imports commands.

#### **6.8.1.1.1 Modify the Imports Command and Connection String**

Replace the second Imports command with the command Imports System.Data.OleDb at the top of the LogIn and Faculty forms.

Open the Form\_Load event procedure of the LogIn form, and change the connection string as follows:

---

```
Dim accString As String = "Provider = Microsoft.Jet.OLEDB.4.0;" &
                           "Data Source = C:\database\CSE_DEPT.mdb;"
```

---

Also change the prefixes of all data classes from Sql to OleDb and the prefixes of all data objects from sql to acc, respectively, for the Form\_Load event procedure.

#### **6.8.1.1.2 Modify the Query Strings for the LogIn Button Event Procedures**

There are two query strings located at two different LogIn buttons' event procedures, TabLogIn and ReadLogIn. Open these two event procedures, and modify these two query strings. This modification is very easy, and the only change is to replace the keyword LIKE in the WHERE clause with the equal symbol (=). Perform this modification on the two query strings.

Also change the prefixes of all data classes from Sql to OleDb and the prefixes of all data objects from sql to acc, respectively, for these two event procedures.

#### **6.8.1.1.3 Modify the Query Strings for the Select and Update Button Event Procedures**

Open the Select and the Update buttons' Click event procedures to modify the query strings. This modification is very easy, and the only change is to replace the keyword LIKE in the WHERE clause with the equal symbol (=) for both query strings.

Change the prefixes of all data classes from Sql to OleDb and the prefixes of all data objects from sql to acc, respectively, for these two event procedures.

Other modifications for this form include changing the data type of the passed argument FacultyReader from SqlDataReader to OleDbDataReader in the user-defined subroutine FillFacultyReader() and changing the data type of the passed argument cmd from SqlCommand to OleDbCommand, respectively, in the user-defined subroutine UpdateParameters(). Also change the data type of all parameters from SqlDbType to OleDbType in the same subroutine.

#### 6.8.1.1.4 Other Modifications

Change the prefixes of all data classes from Sql to OleDb and the prefixes of all data objects from sql to acc, respectively, for the following event procedures:

- The Cancel button's Click event procedure in the LogIn form
- The Form\_Load event procedure in the Faculty form
- The Delete button's Click event procedure in the Faculty form
- The Exit button's Click event procedure in the Selection form

Because we will not use other forms in this project such as the Course, Student, Insert Faculty, and SP forms, we do not need to make modifications to these forms. One possible problem is that you may encounter some debug errors when you run this project because of unmodified codes in these forms. To solve this problem, just comment out those codes that have not been modified.

Now let's create our stored procedure in the Access database.

#### 6.8.1.2 Create Stored Procedures in the Access Database

As we mentioned at the beginning of this section, this data updating is to update all columns of one existing faculty record except the faculty\_id column since it is unnecessary to update the primary key of the Faculty table. A better way to update a faculty\_id is to insert a new faculty record with a new faculty\_id, which is common sense.

Let's create the stored procedure to update one faculty record now.

Open our sample database CSE\_DEPT.mdb, which is located in the folder **database** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). Copy this database into a folder in your root drive, such as **C:\database**.

On the opened database, select the Faculty table from the list, and click the Queries tab from the Object list. Double-click the Create query in the Design view to open the Query Builder dialog box. Then click the Close button to close the Show Table dialog box.

Right-click on the top pane, and select the SQL View item from the pop-up menu to open the SQL window, which is shown in Figure 6.25. Enter the Update statement shown in Figure 6.25 into this window as our stored procedure.

Go to the File|Save menu to save this stored procedure as AccessUpdateSP.

To confirm this stored procedure, we can run this query inside the Access environment. Right-click the newly created stored procedure AccessUpdateSP from the list, select the Open item from the pop-up menu, and click Yes to begin to run this

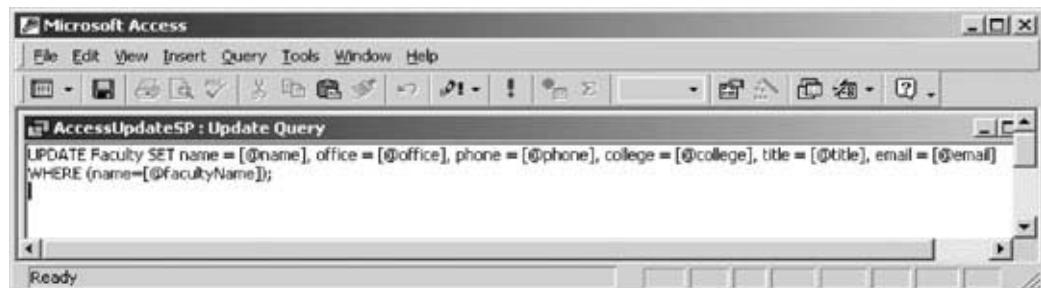


Figure 6.25. The stored procedure in the Access database.

query. Enter the following updated faculty information for each parameter input box:

- |                       |                               |
|-----------------------|-------------------------------|
| ■ Frank Tailor        | name parameter                |
| ■ MTC-228             | office parameter              |
| ■ 750-378-1220        | phone parameter               |
| ■ University of Miami | college parameter             |
| ■ Associate Professor | title parameter               |
| ■ ftaylor@college.edu | email parameter               |
| ■ Ying Bai            | facultyName dynamic parameter |

After finish entering this new data, click the Yes button to confirm that we want to perform this data update.

Now click Tables from the Object list, and then double-click the Faculty table to open it to confirm our data update. You will find that the old record for the faculty member named Ying Bai has been replaced by our updated record, which is shown in Figure 6.26.

faculty_id	name	office	phone	college	title	
A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	bandersc
A77587	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor	dangles@
B66750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor	abrown@
B78890	Frank Tailor	MTC-228	750-378-1220	University of Miami	Associate Professor	ftaylor@c
B86590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@
H99118	Jeff Henry	MTC-336	750-330-8650	Ohio State University	Associate Professor	jhenry@k
J33486	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor	sjohnson
K69880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor	jking@ct
K98766	Kim Collins	MTC-116	750-330-9987	University of Chicago	Associate Professor	kcollins@

Figure 6.26. Confirmation of the data updating.

Now recover the original record for this faculty member with the following information since we want to keep our data unique:

- |                               |                |
|-------------------------------|----------------|
| ■ Ying Bai                    | name column    |
| ■ MTC-211                     | office column  |
| ■ 750-378-1148                | phone column   |
| ■ Florida Atlantic University | college column |
| ■ Assistant Professor         | title column   |
| ■ ybai@college.edu            | email column   |

At this point, we have finished creating the stored procedure in the Access database. Click the File|Save menu to save our original database, and close the sample database. Next, let's develop the codes in Visual Basic.NET to call this stored procedure to perform the data updating action on the database.

### 6.8.1.3 Call the Stored Procedure to Update the Faculty Information

First, add one more form-level variable UpdateFlag as a monitor to indicate whether data updating has occurred. This variable will be used later by the Select button event procedure to validate the data updating action.

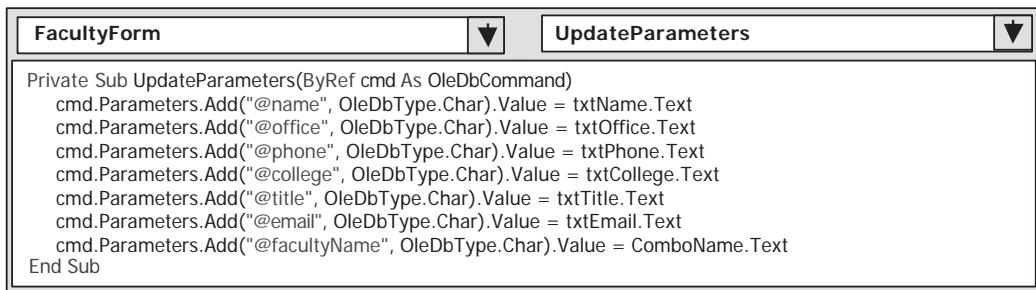
Then open the Update button's Click event procedure, and add the code shown in Figure 6.27 into this procedure.

Let's see how this piece of code works.

- The content of the query string is now equal to the name of the stored procedure we developed in the last section, since we need to call it to perform the data updating action. This name must be identical to the name we used when we developed this stored procedure in the Access database. Otherwise, the project would not find the stored procedure as the project runs.
- The UpdateFlag is set to indicate that data updating occurs. This flag will be used later by the Select button's Click event procedure to perform the data

	cmdUpdate	▼	Click	▼
A	<pre>Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click     Dim cmdString As String = "AccessUpdateSP"     Dim accCommand As New OleDbCommand     Dim intUpdate As Integer</pre>			
B	<pre>        UpdateFlag = True          'indicate an update occurs         FacultyName = ComboName.Text      'reserve old faculty name         accCommand.Connection = LogInForm.accConnection         accCommand.CommandType = CommandType.StoredProcedure         accCommand.CommandText = cmdString         UpdateParameters(accCommand)         intUpdate = accCommand.ExecuteNonQuery()         accCommand.Dispose()         accCommand = Nothing         If intUpdate = 0 Then             MessageBox.Show("The data updating is failed")             Exit Sub         End If     End Sub</pre>			

Figure 6.27. Coding for the Update button event procedure.



The screenshot shows the Microsoft Visual Studio IDE. A code editor window is open with the title 'FacultyForm' at the top left. To the right of the title bar are two dropdown arrows. The code itself is a Private Sub routine named 'UpdateParameters'. It uses an OleDbCommand object to set parameters based on text box values. The code is as follows:

```

Private Sub UpdateParameters(ByRef cmd As OleDbCommand)
    cmd.Parameters.Add("@name", OleDbType.Char).Value = txtName.Text
    cmd.Parameters.Add("@office", OleDbType.Char).Value = txtOffice.Text
    cmd.Parameters.Add("@phone", OleDbType.Char).Value = txtPhone.Text
    cmd.Parameters.Add("@college", OleDbType.Char).Value = txtCollege.Text
    cmd.Parameters.Add("@title", OleDbType.Char).Value = txtTitle.Text
    cmd.Parameters.Add("@email", OleDbType.Char).Value = txtEmail.Text
    cmd.Parameters.Add("@facultyName", OleDbType.Char).Value = ComboName.Text
End Sub

```

Figure 6.28. Modifications to the subroutine UpdateParameters.

validation for our data updating. This flag is used to determine whether a default faculty photo should be displayed if data updating occurs.

- C. During the time the Command object is initialized and built, the CommandType is set to StoredProcedure to tell the project that the query to be executed is a stored procedure, not a normal query. Also, the name of the stored procedure is assigned to the CommandText property to allow the project to locate the stored procedure as the project runs.
- D. Finally, the ExecuteNonQuery() method is called to call the stored procedure to perform the data updating action.

The detailed coding for the user-defined subroutine UpdateParameters() is shown in Figure 6.28.

Basically, the coding is identical to the coding we did for the last project. Two modifications have been made to this subroutine in order for it to match our data updating function in this project: First, the parameter faculty\_id has been removed from this subroutine since we do not need to update the faculty\_id, as mentioned at the beginning of Section 6.8. Second, the name of the dynamic parameter has been changed from @Param1 to @facultyName since the name used for the dynamic parameter in our stored procedure is @facultyName and the name used here must be identical to the name used in the stored procedure.

Now we have finished all coding for the data updating using stored procedures in the Access database. Before we can run the project to test this data updating functionality, we will complete the coding for the data validation for the updating. In that way, we can run the project to perform both the data updating and the data validation at the same time.

#### **6.8.1.4 Confirm the Faculty Information Updating**

As we did for the last project, we will still use the Faculty form, that is, the Select button's Click event procedure in the Faculty form, to perform the data validation for this data updating. The only modification to this form is that we add an Or condition to the If block in the subroutine ShowFaculty() to detect whether the UpdateFlag has been set. If yes, which means that data updating has occurred, we need to display a default faculty photo for that data updating. The modified coding is shown in Figure 6.29. The code we developed before is highlighted with a gray background.

```

Private Sub ShowFaculty(ByVal fName As String)
    .....
    If InsertFaculty.chkPhoto.Checked = True Then
        If InsertFaculty.txtPhotoLocation.Text <> "Default Location" Then
            FacultyImage = InsertFaculty.txtPhotoLocation.Text & "\" & InsertFaculty.txtPhotoName.Text
        Else
            FacultyImage = InsertFaculty.txtPhotoName.Text
        End If
    Else
        If InsertFaculty.InsertFacultyFlag = True Or FacultyNameFlag = True Or UpdateFlag = True Then
            PhotoBox.Image = System.Drawing.Image.FromFile("Default.jpg")
            If FacultyNameFlag = True Then
                FacultyNameFlag = False
            End If
        End If
        If UpdateFlag = True Then
            UpdateFlag = False
        End If
    .....

```

**Figure 6.29.** Modifications to the Select button's event procedure.

Now we can run the project to test the stored procedure to perform the data updating.

Click the Start Debugging button to start our project. Enter a suitable username and password in the LogIn form, and select the Faculty Information item from the Selection form window to open the Faculty form. Keep the default faculty name Ying Bai from the combo box control, and click the Select button to display the information for the selected faculty member.

To update this faculty information, enter the following information into the associated text boxes:

- |                       |                  |
|-----------------------|------------------|
| ■ Frank Tailor        | Name text box    |
| ■ Associate Professor | Title text box   |
| ■ MTC-228             | Office text box  |
| ■ 750-378-1220        | Phone text box   |
| ■ University of Miami | College text box |
| ■ ftailor@college.edu | Email text box   |

Click the Update button to call the stored procedure to update this faculty information in the Faculty table in the database.

To confirm this updating, first click the drop-down arrow of the ComboName combo box control, select any other faculty name from the box, and click the Select button to display the information related to that selected faculty member. Then reopen the ComboName combo box control, and select our newly updated faculty name, Frank Tailor, from the box. Click the Select button to retrieve that updated faculty information from the database and display it in this form. Immediately you will find that the updated faculty information is returned and displayed, as shown in Figure 6.30.

Our data updating action using the stored procedure is successful.

In order to keep the database neat, you can open the sample database and the Faculty table to recover the original faculty information by entering the following data into the associated columns:



Figure 6.30. Running status of the data validation process.

- |                               |                |
|-------------------------------|----------------|
| ■ Ying Bai                    | name column    |
| ■ MTC-211                     | office column  |
| ■ 750-378-1148                | phone column   |
| ■ Florida Atlantic University | college column |
| ■ Assistant Professor         | title column   |
| ■ ybai@college.edu            | email column   |

The completed project AccessUpdateRTOBJECT can be found in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

### **6.8.2 Update Data for an SQL Server Database by Using Stored Procedures**

Updating data by using stored procedures developed in the SQL Server database is very similar to the data updating we performed in the last section. With a few modifications to the existing project SQLUpdateDeleteRTOBJECT, we can easily create our new project SQLUpdateRTOBJECTSP to perform the data updating by calling stored procedures developed in the SQL Server database.

To develop our new project in this section, we divide it into three steps:

1. Modify the existing project SQLUpdateDeleteRTOBJECT to create our new project SQLUpdateRTOBJECTSP.
2. Develop the stored procedure in the SQL Server database.
3. Call the stored procedure to perform the data updating, and validate the updated faculty information using the Faculty form window.

Now let's start with the first step.

#### **6.8.2.1 Modify the Existing Project to Create a New Project**

Open Windows Explorer, and create a new folder, **Chapter 6**, if you have not already created it. Then open your Internet browser and locate the folder **DBProjects\**

**Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and copy the project SQLUpdateDeleteRTOBJECT to the new folder **C:\Chapter 6**. Change the name of the project from SQLUpdateDeleteRTOBJECT to SQLUpdateRTOBJECTSP. Double-click SQLUpdateRTOBJECTSP.vbproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to the Project|SQLUpdateRTOBJECTSP Properties menu to open the project's property window. Change the Assembly name from SQLUpdateDeleteRTOBJECT to SQLUpdateRTOBJECTSP and the Root namespace from SQLUpdateDeleteRTOBJECT to SQLUpdateRTOBJECTSP.
- Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to SQLUpdateRTOBJECTSP. Click OK to close this dialog box.

Go to File|Save All to save these modifications. Now we are ready to modify our code for our new project SQLUpdateRTOBJECTSP.

The code modifications include the following:

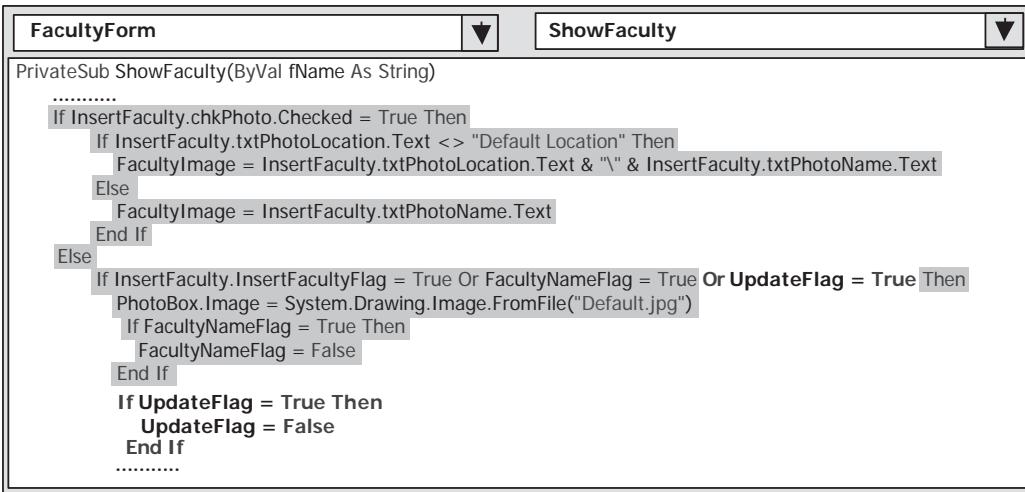
1. Add one more form-level variable, UpdateFlag, to the Faculty form. This flag is used to indicate whether updating has occurred, and it will be used later by the subroutine ShowFaculty() to display a default faculty photo for the updated faculty information.
2. Add one more Or condition to the If block in the subroutine ShowFaculty() to detect whether the UpdateFlag has been set. If yes, which means that data updating has occurred, we need to display a default faculty photo for that updated faculty member.
3. Change the query string for the Update button event procedure in the Faculty form to allow the procedure to call the stored procedure to perform the data updating.

Open the code window of the Faculty form, and add one more form-level variable, UpdateFlag, using the statement Private UpdateFlag As Boolean.

Open the user-defined subroutine ShowFaculty(), and add one more Or condition to this procedure. Your finished modifications should match the code shown in Figure 6.31. The code developed in the previous sections is highlighted with a gray background, and the newly added code is indicated in bold.

Modification step 3 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 this modification, 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 have assumed that we know this information, and we can put those pieces of information into our procedure in advance. This assumed information includes the following:

1. The name of the stored procedure – assume it is dbo.UpdateFacultySP.
2. The names of the input parameters – assume that the names of those input parameters are identical to the column names in the database.
3. The name of the input dynamic parameter – assume it is @facultyName.



```

FacultyForm
ShowFaculty

PrivateSub ShowFaculty(ByVal fName As String)
    .....
    If InsertFaculty.chkPhoto.Checked = True Then
        If InsertFaculty.txtPhotoLocation.Text <> "Default Location" Then
            FacultyImage = InsertFaculty.txtPhotoLocation.Text & "\" & InsertFaculty.txtPhotoName.Text
        Else
            FacultyImage = InsertFaculty.txtPhotoName.Text
        End If
    Else
        If InsertFaculty.InsertFacultyFlag = True Or FacultyNameFlag = True Or UpdateFlag = True Then
            PhotoBox.Image = System.Drawing.Image.FromFile("Default.jpg")
            If FacultyNameFlag = True Then
                FacultyNameFlag = False
            End If
        End If
        If UpdateFlag = True Then
            UpdateFlag = False
        End If
    .....

```

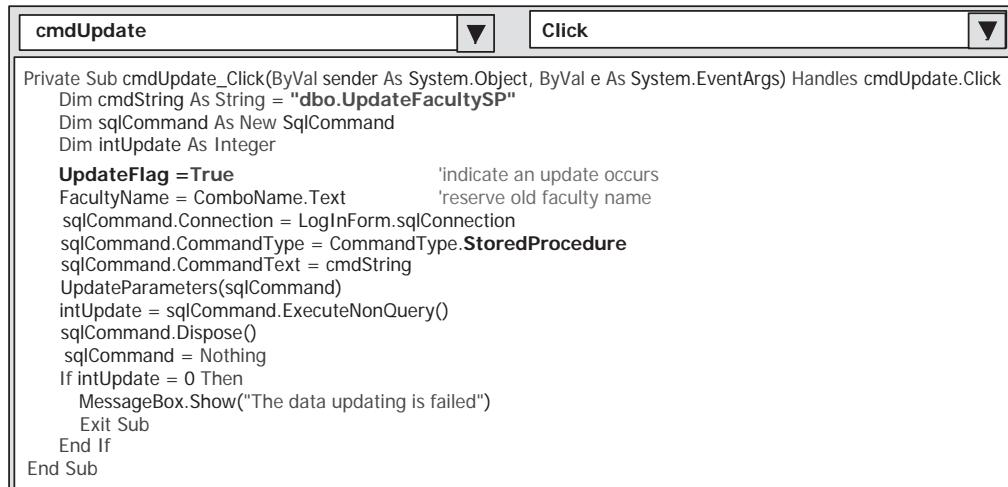
Figure 6.31. Modifications to the subroutine ShowFaculty.

On the basis of these assumptions, we can first modify our coding 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 when we develop it later.

Open the Update button's Click event procedure and modify its coding. Your finished modifications to this procedure should match the code shown in Figure 6.32. The modified parts have been highlighted in bold.

Let's see how this piece of modified code works.

- The content of the query string now is equal to the name of the stored procedure.



```

cmdUpdate
Click

Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click
    Dim cmdString As String = "dbo.UpdateFacultySP"
    Dim sqlCommand As New SqlCommand
    Dim intUpdate As Integer
    UpdateFlag = True                                'indicate an update occurs
    FacultyName = ComboName.Text                      'reserve old faculty name
    sqlCommand.Connection = LogInForm.sqlConnection
    sqlCommand.CommandType = CommandType.StoredProcedure
    sqlCommand.CommandText = cmdString
    UpdateParameters(sqlCommand)
    intUpdate = sqlCommand.ExecuteNonQuery()
    sqlCommand.Dispose()
    sqlCommand = Nothing
    If intUpdate = 0 Then
        MessageBox.Show("The data updating is failed")
        Exit Sub
    End If
End Sub

```

Figure 6.32. Modified coding for the Update button event procedure.

```

Private Sub UpdateParameters(ByRef cmd As SqlCommand)
    cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text
    cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text
    cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text
    cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text
    cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text
    cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text
    cmd.Parameters.Add("@facultyName", SqlDbType.Char).Value = ComboName.Text
End Sub

```

Figure 6.33. Modified coding for the subroutine UpdateParameters.

- B. The form-level variable UpdateFlag is set to indicate that data updating has occurred. This flag will be used by the subroutine ShowFaculty() to display a default faculty photo for the updated faculty record when the data validation is performed later.
- C. The CommandType property of the Command object is set to StoredProcedure to tell the project that a stored procedure should be called to perform the data updating job.

The modifications to the coding of the user-defined subroutine UpdateParameters() are shown in Figure 6.33.

Two modifications are performed for this subroutine. First, the parameter faculty\_id is removed from this subroutine since we do not need to modify this column when we perform the data updating. Second, the name of the dynamic parameter is changed from @Param1 to @facultyName since we must keep all names of the input parameters to the stored procedure identical to those parameters used in our Visual Basic.NET project.

Now we have finished all coding modifications in the Visual Basic.NET environment. Let's start to create our stored procedure in the SQL Server database. There are two ways you can create the stored procedure: one way is to create it in SQL Server Management Studio Express, and the other way is to create it in Server Explorer in the Visual Studio environment. Since we are working on a Visual Basic.NET project, we will use the second way to create our stored procedure.

### **6.8.2.2 Develop the Stored Procedure in the SQL Server Database**

Open Server Explorer in the Visual Studio environment, and click the small plus icon before our sample database CSE\_DEPT.mdf to expand it. Then right-click 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 to the name of the stored procedure we used in our coding in the last section. Then add the code shown in Figure 6.34 into this stored procedure as the body of our new stored procedure.

Refer to Section 2.10.2 in Chapter 2 for the data types of those input parameters. These data types should be identical to those of the associated columns defined in the Faculty table.

The screenshot shows the Microsoft Visual Studio IDE with the title bar "SQLUpdateRTOBJECTSP - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has various icons for file operations like Open, Save, Print, and Find. Below the toolbar is a tab bar with "dbo.StoredPr...SE\_DEPT.MDF\*" (highlighted), "Faculty Form.vb\*", and "Login Form.vb". The main code editor window contains the following T-SQL code:

```

CREATE PROCEDURE dbo.UpdateFacultySP
(
    @name text,
    @office text,
    @phone text,
    @college text,
    @title text,
    @email text,
    @facultyName text
)
AS
    UPDATE Faculty SET name=@name, office=@office, phone=@phone, college=@college,
    title=@title, email=@email
    WHERE (name LIKE @facultyName)
    RETURN

```

The code editor shows syntax highlighting for keywords and identifiers. The status bar at the bottom indicates "Ready", "Ln 16", "Col 1", "Ch 1", and "INS".

Figure 6.34. The created stored procedure.

Go to the File|Save StoredProcedure1 menu to save our stored procedure.

To test our stored procedure, right-click our newly created stored procedure dbo.UpdateFacultySP, which is located in the Stored Procedure folder, and select the Execute item from the pop-up menu to open the Run Stored Procedure dialog box. Enter the following updated information into each field in the Value column of this dialog box:

■ Frank Tailor	Name Value
■ MTC-228	Office Value
■ 750-378-1220	Phone Value
■ University of Miami	College Value
■ Associate Professor	Title Value
■ ftaylor@college.edu	Email Value
■ Ying Bai	FacultyName Value

Your finished information dialog box should match the one that is shown in Figure 6.35.

Click OK to run this stored procedure.

You can open the Faculty table to check whether this execution is successful. Go to the Server Explorer window, right-click the Faculty table, and select Show Table Data to open the Faculty table. You will find that our updated record is there, which is shown as a highlighted row in Figure 6.36.

Our stored procedure is successful.

In order to keep our database neat, we will replace this updated faculty record with the original data. To do this, enter the following information into the updated row:

■ Ying Bai	name column
■ MTC-211	office column
■ 750-378-1148	phone column

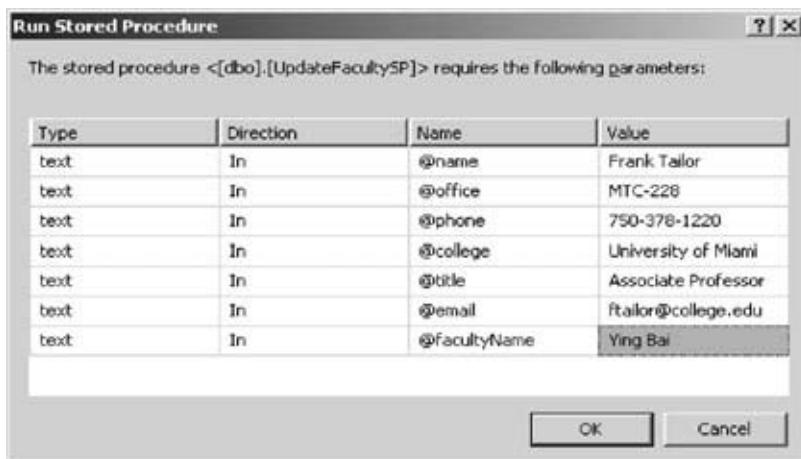


Figure 6.35. The finished information dialog box.

- Florida Atlantic University college column
- Assistant Professor title column
- ybai@college.edu email column

Save and close the database, and let's call this stored procedure from our Visual Basic.NET project to test this data updating functionality.

### 6.8.2.3 Call the Stored Procedure to Perform the Data Updating and Validate the Updated Information

Start the project by clicking the Start Debugging button, enter a suitable username and password in 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 in the combo box control, and click the Select button to display the information for the selected faculty member.

The screenshot shows the Microsoft Visual Studio interface with the title bar "SQLUpdateRTOBJECTSP - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Query Designer, Tools, Window, Community, Help. The toolbar has various icons for file operations. The main area shows the "Faculty: Query... \CSE\_DEPT.MDF" table with the following data:

	faculty_id	name	office	phone	college	title	email
A88979	Allen Jones	MTC-119	750-330-3355	University of Flo...	Professor	asteff@college...	
A99875	All Mhamed	MTC-235	750-330-3387	University of Man...	Associate Profes...	amhomed@college...	
B66750	Alice Brown	MTC-257	750-330-6650	University of Flo...	Assistant Profes...	abrown@college...	
B78880	Frank Tailor	MTC-228	750-378-1220	University of Mi...	Associate Profes...	ftailor@college...	
B86590	Sohish Bihala	MTC-214	750-378-1061	University of No...	Associate Profes...	sbihala@college...	
D77777	Dini Keniry	MTC-119	750-330-3355	University of Chi...	Professor	dkeniry@college...	
F55879	Frank Tom	MTC-335	750-378-9999	University of Flo...	Professor	ftom@college.edu	

At the bottom, there are navigation buttons (Back, Forward, etc.) and a status bar showing "Ready".

Figure 6.36. The updated Faculty table.

To update this faculty information, enter the following information into the associated text boxes:

- |                       |                  |
|-----------------------|------------------|
| ■ Frank Tailor        | Name text box    |
| ■ Associate Professor | Title text box   |
| ■ MTC-228             | Office text box  |
| ■ 750-378-1222        | Phone text box   |
| ■ University of Miami | College text box |
| ■ ftailor@college.edu | Email text box   |

Click the Update button to call the stored procedure to update this faculty information in the Faculty table in the database.

To confirm this updating, first click the drop-down arrow of the ComboName combo box control, select any other faculty name from the box, and click the Select button to display the information related to that selected faculty member. Then reopen the ComboName combo box control, and select the updated faculty name, Frank Tailor, from the box. Click the Select button to retrieve that updated faculty information from the database and display it in this form. Immediately you will find that the updated faculty information is returned and displayed, as shown in Figure 6.37.

Our data updating action using a stored procedure in the SQL Server database is very successful.

In order to keep our database neat, you can open the sample database and the Faculty table to recover the original faculty information by entering the following data into the associated columns:

- |                               |                |
|-------------------------------|----------------|
| ■ Ying Bai                    | name column    |
| ■ MTC-211                     | office column  |
| ■ 750-378-1148                | phone column   |
| ■ Florida Atlantic University | college column |
| ■ Assistant Professor         | title column   |
| ■ ybai@college.edu            | email column   |



Figure 6.37. Confirmation of the data updating.

The completed project SQLUpdateRTOBJECT can be found in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

### 6.8.3 Update Data for an Oracle Database by Using Stored Procedures

Updating data using stored procedures developed in the Oracle database is very similar to the data updating and deleting we performed in Section 6.7. With a few modifications to the existing project OracleUpdateDeleteRTOBJECT, we can easily create a new project, OracleUpdateRTOBJECTSP, to perform the data updating by calling a stored procedure developed in the Oracle database.

To develop our new project in this section, we divide it into three steps:

1. Modify the existing project OracleUpdateDeleteRTOBJECT to create our new project, OracleUpdateRTOBJECTSP.
2. Develop the stored procedure in the Oracle database.
3. Call the stored procedure to perform the data updating, and validate the updated faculty information using the Faculty form window.

Now let's start with the first step.

#### 6.8.3.1 Modify the Existing Project to Create a New Project

Open Windows Explorer, and create a new folder, **Chapter 6**, if you have not already created it. Then open your Internet browser and locate the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and copy the project OracleUpdateDeleteRTOBJECT to the new folder **C:\Chapter 6**. Change the name of the project from OracleUpdateDeleteRTOBJECT to OracleUpdateRTOBJECTSP. Double-click OracleUpdateRTOBJECTSP.vbproj to open this project.

On the opened project, perform the following modifications to get our desired project:

- Go to the Project|OracleUpdateRTOBJECTSP Properties menu to open the project's property window. Change the Assembly name from OracleUpdateDeleteRTOBJECT to OracleUpdateRTOBJECTSP and the Root namespace from OracleUpdateDeleteRTOBJECT to OracleUpdateRTOBJECTSP.
- Click the Assembly Information button to open the Assembly Information dialog box, and change Title and Product to OracleUpdateRTOBJECTSP. Click OK to close this dialog box.

Go to File|Save All to save these modifications. Now we are ready to modify the code for our new project, OracleUpdateRTOBJECTSP.

The code modifications include the following:

1. Add one more form-level variable, UpdateFlag, to the Faculty form. This flag is used to indicate whether updating has occurred, and it will be used later by the subroutine ShowFaculty() to display a default faculty photo for the updated faculty information.
2. Add one more Or condition to the If block in the subroutine ShowFaculty() to detect whether the UpdateFlag has been set. If yes, which means that data

```

FacultyForm
ShowFaculty

Private Sub ShowFaculty(ByVal fName As String)
    .....
    If InsertFaculty.chkPhoto.Checked = True Then
        If InsertFaculty.txtPhotoLocation.Text <> "Default Location" Then
            FacultyImage = InsertFaculty.txtPhotoLocation.Text & "\" & InsertFaculty.txtPhotoName.Text
        Else
            FacultyImage = InsertFaculty.txtPhotoName.Text
        End If
    Else
        If InsertFaculty.InsertFacultyFlag = True Or FacultyNameFlag = True Or UpdateFlag = True Then
            PhotoBox.Image = System.Drawing.Image.FromFile("Default.jpg")
            If FacultyNameFlag = True Then
                FacultyNameFlag = False
            End If
        If UpdateFlag = True Then
            UpdateFlag = False
        End If
    End If
    .....

```

**Figure 6.38.** Modifications to the subroutine ShowFaculty.

updating has occurred, we need to display a default faculty photo for that updated faculty member.

3. Change the query string for the Update button event procedure in the Faculty form to allow the procedure to call the stored procedure to perform the data updating.

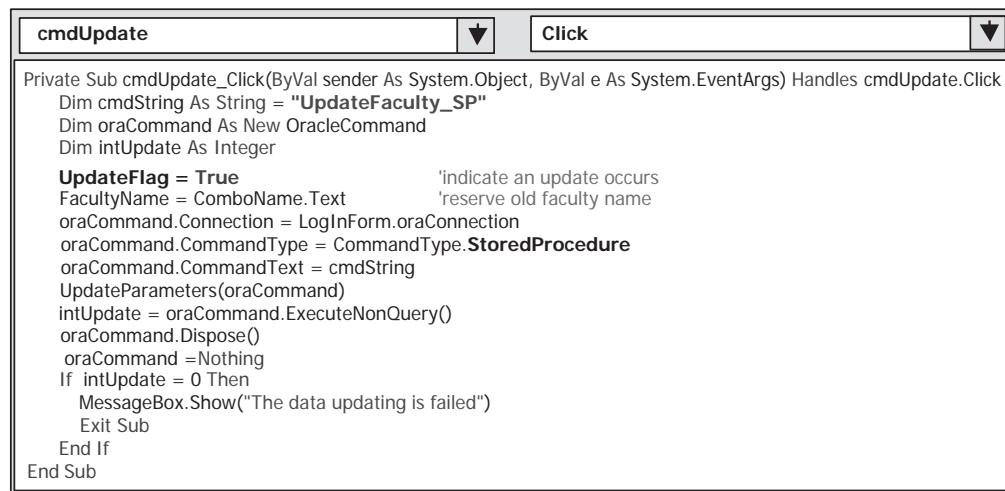
Open the code window of the Faculty form, and add one more form-level variable, UpdateFlag, using the statement Private UpdateFlag As Boolean.

Open the user-defined subroutine ShowFaculty(), and add one more Or condition to this procedure. Your finished modifications should match the code shown in Figure 6.38. The code developed in the previous sections is highlighted with a gray background, and the newly added code is indicated in bold.

Modification step 3 should be performed after the stored procedure has been created in the Oracle database, since we need some information from the created stored procedure to execute this modification, 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 have assumed that we know this information, and we can put those pieces of information into our procedure in advance. This assumed information includes the following:

1. The name of the stored procedure – assume it is UpdateFaculty\_SP.
2. The names of the input parameters – assume that the names of those input parameters are identical to the column names in the database.
3. The name of the input dynamic parameter – assume it is @facultyName.

On the basis of these assumptions, we can first modify our coding 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 when we develop it later.



The screenshot shows the Visual Studio IDE with the code editor open. The title bar says "cmdUpdate" and the tab bar says "Click". The code is as follows:

```

Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click
    Dim cmdString As String = "UpdateFaculty_SP"
    Dim oraCommand As New OracleCommand
    Dim intUpdate As Integer

B   UpdateFlag = True           'indicate an update occurs
    FacultyName = ComboName.Text      'reserve old faculty name
    oraCommand.Connection = LogInForm.oraConnection
    oraCommand.CommandType = CommandType.StoredProcedure
    oraCommand.CommandText = cmdString
    UpdateParameters(oraCommand)
    intUpdate = oraCommand.ExecuteNonQuery()
    oraCommand.Dispose()
    oraCommand = Nothing
    If intUpdate = 0 Then
        MessageBox.Show("The data updating is failed")
        Exit Sub
    End If
End Sub

```

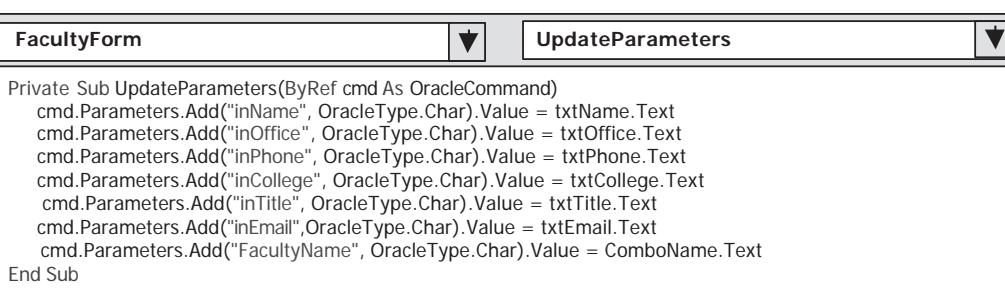
Figure 6.39. Modified coding for the Update button event procedure.

Open the Update button's Click event procedure and modify its coding. Your finished modifications to this procedure should match the code shown in Figure 6.39. The modified parts have been highlighted in bold.

Let's see how this piece of modified code works.

- The content of the query string now is equal to the name of the stored procedure.
- The form-level variable `UpdateFlag` is set to indicate that data updating has occurred. This flag will be used by the subroutine `ShowFaculty()` to display a default faculty photo for the updated faculty record when the data validation is performed later.
- The  `CommandType` property of the `Command` object is set to `StoredProcedure` to tell the project that a stored procedure should be called to perform the data updating job.

The modifications to the coding of the user-defined subroutine `UpdateParameters()` are shown in Figure 6.40.



The screenshot shows the Visual Studio IDE with the code editor open. The title bar says "FacultyForm" and the tab bar says "UpdateParameters". The code is as follows:

```

Private Sub UpdateParameters(ByRef cmd As OracleCommand)
    cmd.Parameters.Add("inName", OracleType.Char).Value = txtName.Text
    cmd.Parameters.Add("inOffice", OracleType.Char).Value = txtOffice.Text
    cmd.Parameters.Add("inPhone", OracleType.Char).Value = txtPhone.Text
    cmd.Parameters.Add("inCollege", OracleType.Char).Value = txtCollege.Text
    cmd.Parameters.Add("inTitle", OracleType.Char).Value = txtTitle.Text
    cmd.Parameters.Add("inEmail", OracleType.Char).Value = txtEmail.Text
    cmd.Parameters.Add("FacultyName", OracleType.Char).Value = ComboName.Text
End Sub

```

Figure 6.40. Modified coding for the subroutine `UpdateParameters`.

Two modifications are performed for this subroutine. First, the parameter faculty\_id is removed from this subroutine since we do not need to modify this column when we perform the data updating. Second, the name of the dynamic parameter has been changed from Param1 to FacultyName since we must keep all names of the input parameters to the stored procedure identical to those parameters we used in our coding in the Visual Basic.NET project. The reason we changed the input parameters' names by adding the prefix "in" before each of them is that PL/SQL is a case-insensitive language. In order to distinguish between the column names of the Faculty table and the input parameters' names, we must add this prefix.

Now we have finished all coding modifications in the Visual Basic.NET environment. Let's start to create our stored procedure in the Oracle database. Refer to Section 4.19.7 in Chapter 4 for a detailed discussion about stored procedures and packages in the Oracle database. In this section, since we want to perform the data updating functionality and we do not need the query to return any data from the database, the stored procedure is enough for our applications. There are many ways you can create a stored procedure in an Oracle database. One way is to create it using the Object Browser page in Oracle Database 10g XE, and the other way is to create it using the SQL Command page. Since we used Object Browser to create our data tables in Chapter 2, we will use the second way to create our stored procedure here.

### **6.8.3.2 Develop the Stored Procedure in the Oracle Database**

Open the Oracle Database 10g XE home page by going to Start|All Programs| Oracle Database 10g Express Edition|Go To Database Home Page items. This time we want to use the SQL Command page to create our stored procedure. The reason is because we can run and test our stored procedure directly in the Oracle Database 10g XE environment as soon as the stored procedure is done. That is very convenient for us because we will not need to wait to test it by calling the finished stored procedure later from the Visual Basic.NET project.

To open the SQL Command page, click the SQL icon and select the SQL Commands|Enter Command item.

Enter the code shown in Figure 6.41 into this page as the body of our stored procedure.

Now highlight all the code, and click the Run button to create our stored procedure. Immediately you will find a message is displayed in the bottom pane in the Results tab to indicate that the stored procedure has been created, which is shown below:

---

**Procedure created.**

**0.19 seconds**

---

To call this stored procedure to test it, type the code shown in Figure 6.42 under the code of the stored procedure. Then highlight that code, and click the Run button to run the stored procedure.

If the stored procedure is correctly created and executed, the running result is displayed in the bottom pane under the Results tab.

User: SYSTEM  
Home > SQL > SQL Commands

Autocommit Display 10

```
create or replace PROCEDURE UpdateFaculty_SP
(inName IN VARCHAR2,
inOffice IN VARCHAR2,
inPhone IN CHAR,
inCollege IN VARCHAR2,
inTitle IN VARCHAR2,
inEmail IN VARCHAR2,
FacultyName IN VARCHAR2) AS
begin
  UPDATE Faculty
  SET name = inName, office = inOffice, phone = inPhone, college = inCollege,
      title = inTitle, email = inEmail
  WHERE name = FacultyName;
end;
```

Figure 6.41. The body of our stored procedure.

User: SYSTEM  
Home > SQL > SQL Commands

Autocommit Display 10

```
create or replace PROCEDURE UpdateFaculty_SP
(inName IN VARCHAR2,
inOffice IN VARCHAR2,
inPhone IN CHAR,
inCollege IN VARCHAR2,
inTitle IN VARCHAR2,
inEmail IN VARCHAR2,
FacultyName IN VARCHAR2) AS
begin
  UPDATE Faculty
  SET name = inName, office = inOffice, phone = inPhone, college = inCollege,
      title = inTitle, email = inEmail
  WHERE name = FacultyName;
end;

-- To call the stored procedure, you can use the following block coding
begin
  UpdateFaculty_SP('Frank Tailor', 'HIC-777', '750-330-3377',
                    'University of Miami', 'Professor', 'ftailor@college.edu',
                    'Ying Bai');
end;
```

Figure 6.42. Code to run the stored procedure.

	FACULTY_ID	NAME	OFFICE	PHONE	COLLEGE	TITLE	EMAIL
	A52990	Black Anderson	MTC-218	750-378-9987	Virginia Tech	Professor	bblackerson@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
	<b>B78880</b>	Frank Taylor	MTC-777	750-330-3377	University of Miami	Professor	ftaylor@college.edu
	B66590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor	sbhalla@college.edu
	H69918	Jeff Henry	MTC-326	750-330-6650	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 Kno	MTC-324	750-378-1230	East Florida University	Professor	jkno@college.edu

Figure 6.43. The updated Faculty table.

---

**Statement processed.**

**0.13 seconds**

---

Now let's open our Faculty table to confirm that the selected row has been updated after the stored procedure UpdateFaculty\_SP is executed. Click the Home button that is located at the upper right corner of the page to return to the Home page. Click the drop-down arrow on the Object Browser icon, and select the item Browse|Tables to open the Tables page.

Click the Faculty table from the table list, and then click the Data tab to open the Faculty table, which is shown in Figure 6.43.

You will find that the fourth row, which is indicated by the arrow, has been updated. This confirms that our stored procedure works fine.

Before we close the Oracle Database 10g XE window, it is highly recommended that the original Faculty table be recovered. To do that, click the Edit icon before the fourth row, and then click the Delete button to remove this row. Then click the Insert Row button, and enter the following information into the associated box to recover the original fourth row:

- |                               |                |
|-------------------------------|----------------|
| ■ B78880                      | Faculty ID box |
| ■ Ying Bai                    | Name box       |
| ■ MTC-211                     | Office box     |
| ■ 750-378-1148                | Phone box      |
| ■ Florida Atlantic University | College box    |
| ■ Assistant Professor         | Title box      |
| ■ ybai@college.edu            | Email box      |

Click the Create button to insert this row into the Faculty table.

Now close Oracle Database 10g XE. Next, we will call this stored procedure from the Visual Basic.NET project to perform the data updating functionality.

### 6.8.3.3 Call the Stored Procedure to Perform the Data Updating and Validation

Since we finished the modifications to our new project in Section 6.8.3.1, now let's run the project to test the data updating functionality by calling the stored procedure we developed in the last section.

Click the Start Debugging button to run our project, enter a suitable username and password in 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 combo box control, and click the Select button to display the information for the selected faculty member.

To update this faculty information, enter the following information into the associated text boxes:

- |                       |                  |
|-----------------------|------------------|
| ■ Frank Tailor        | Name text box    |
| ■ Associate Professor | Title text box   |
| ■ MTC-228             | Office text box  |
| ■ 750-378-1222        | Phone text box   |
| ■ University of Miami | College text box |
| ■ ftailor@college.edu | Email text box   |

Click the Update button to call the stored procedure to update this faculty information in the Faculty table in the database.

To confirm this updating, first click the drop-down arrow of the ComboName combo box control, select any other faculty name from the box, and click the Select button to display the information related to that selected faculty. Then reopen the ComboName combo box control, and select the updated faculty name, Frank Tailor, from the box. Click the Select button to retrieve that updated faculty information from the database and display it in this form. Immediately you will find that the updated faculty information is returned and displayed, as shown in Figure 6.44.

To keep our database neat, we recommend that you recover the original information for the updated faculty member. To do that, enter the following original information into the associated text boxes:

- |                               |                |
|-------------------------------|----------------|
| ■ Ying Bai                    | name column    |
| ■ MTC-211                     | office column  |
| ■ 750-378-1148                | phone column   |
| ■ Florida Atlantic University | college column |
| ■ Assistant Professor         | title column   |
| ■ ybai@college.edu            | email column   |

Click the Update button to recover this information.

Click the Back and Exit buttons to close our project. The completed project OracleUpdateRTOBJECTSP is located in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).



Figure 6.44. Confirmation of the data updating.

#### 6.8.4 Delete Data from the Oracle Database by Using Stored Procedures

In this section we will discuss how to delete an existing record from the Oracle database using a stored procedure. As mentioned in the previous sections, to delete data from related tables, we must first delete data from the child table and then remove data from the parent table. We will use our Faculty table as an example to show how to delete an existing record from related tables.

In our sample database, there are two child tables related to our Faculty table, the LogIn table and the Course table, and one child table related to the Course table, the StudentCourse table. Two child tables are connected with the Faculty table by faculty\_id, which is the primary key in the Faculty table but a foreign key in the two child tables. The connection between the Course and StudentCourse tables is course\_id, which is the primary key in the Course table but a foreign key in the StudentCourse table. To delete a faculty member from the parent table, Faculty, we must first delete the records related to that faculty\_id from the child tables, LogIn and Course, then delete the records related to the course\_id from the child table StudentCourse, and finally delete that faculty member from the Faculty table. Basically, this deleting can be divided into the following four steps:

1. Delete all records that use course\_id as the foreign key from the child table StudentCourse.
2. Delete all records that use faculty\_id as the foreign key from the Course table. In our sample database, there are four to six records related to each faculty\_id in the Course table since each faculty member teaches four to six courses.
3. Delete all records that use faculty\_id as the foreign key from the LogIn table. In our sample database, only one row is related to each faculty\_id in the LogIn table.
4. Delete the faculty member from the parent, or Faculty, table.

These four steps can be mapped to three deleting queries. Of steps 2, 3, and 4, each is equivalent to a deleting query. The deleting query equivalent to step 1 can

be performed by the Oracle database engine since the On Delete Cascade option was set up when we created the foreign key between the Course and StudentCourse tables when we built our sample database in Chapter 2. Refer to Section 7.8.2.4 in Chapter 7 to get more detailed information about this issue. In the following section, we will combine three deleting queries that are equivalent to steps 2, 3, and 4 into one stored procedure to perform the data deletion functionality.

We will use the sample project OracleUpdateRTOBJECTSP we developed in the previous section to illustrate this data deletion for related tables. We only need to modify the coding for the Delete button's Click event procedure in the Faculty form to perform the data deletion. We will divide this discussion into two sections. First, we need to create stored procedures in the Oracle database to contain the three deleting queries, and then we can modify the coding in the Delete button event procedure to call the stored procedure to delete an existing record from our Faculty table.

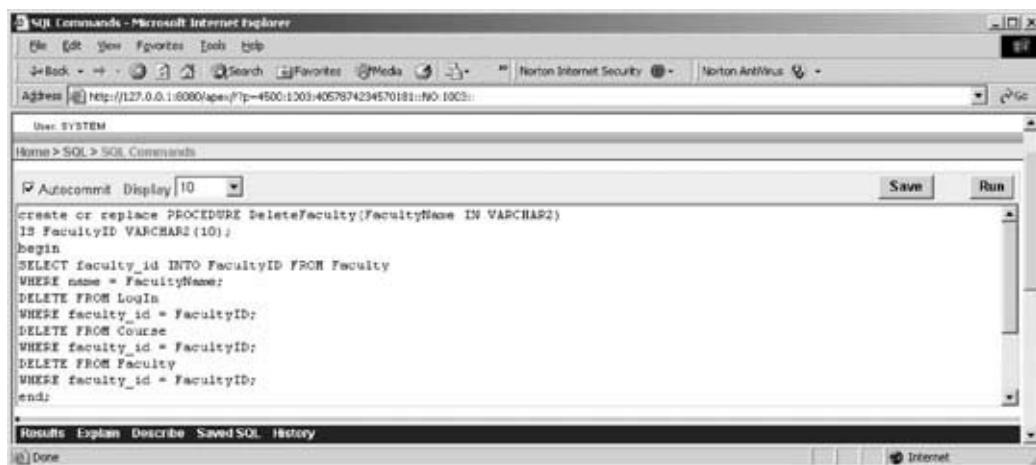
#### 6.8.4.1 Create the Stored Procedure in the Oracle Database

Open the Oracle Database 10g XE home page by going to Start|All Programs| Oracle Database 10g Express Edition|Go To Database Home Page items. This time we still want to use the SQL Command page to create our stored procedure. The reason is because we can run and test our stored procedure directly in the Oracle Database 10g XE environment as soon as the stored procedure is done. That is very convenient for us because we will not need to wait to test it by calling the finished stored procedure later from the Visual Basic.NET project.

To open the SQL Command page, click the SQL icon and select the SQL Commands|Enter Command item.

Enter the code shown in Figure 6.45 into this page as the body of our stored procedure.

The name of this stored procedure is DeleteFaculty, and it has only one input parameter, FacultyName, which is the faculty name defined in the Faculty table. Since we need to use faculty\_id as a criterion to delete data from both the child and the parent tables, first we need to perform a SELECT INTO query to pick



A screenshot of a Microsoft Internet Explorer window titled "SQL Commands - Microsoft Internet Explorer". The address bar shows the URL: "Http://127.0.0.1:60804/app/rtp=4500:1203:4057874234570181::NO:10GB". The user is logged in as "SYSTEM". The main content area displays the following SQL code:

```
create or replace PROCEDURE DeleteFaculty(FacultyName IN VARCHAR2)
IS FacultyID VARCHAR2(10);
begin
SELECT faculty_id INTO FacultyID FROM Faculty
WHERE name = FacultyName;
DELETE FROM LogIn
WHERE faculty_id = FacultyID;
DELETE FROM Course
WHERE faculty_id = FacultyID;
DELETE FROM Faculty
WHERE faculty_id = FacultyID;
end;
```

Figure 6.45. The body of the stored procedure.

up faculty\_id from the Faculty table and assign it to the local variable FacultyID. Then three DELETE commands are executed to delete the data in the three related tables.

Select this piece of coding by highlighting it, and then click the Run button to create our stored procedure.

To confirm our stored procedure, we can directly call and run it in this SQL Command window. Type the following code under the stored procedure:

---

```
begin
DeleteFaculty("Ying Bai");
end;
```

---

Highlight these three lines of code, and then click the Run button to run the stored procedure.

If this calling is successful, the following information will be displayed in the Results window:

---

```
Statement processed.
0.75 seconds
```

---

Now let's verify this data deletion by opening the three tables to check for those deleted records. Click the Home icon, click the drop-down arrow of the Object Browser, and select Browse|Tables to open the table list.

First, check the LogIn table by selecting it from the list, and then click the Data tab to open this table. You will find that the faculty member with the username ybai has been deleted from this table.

Next, open the Course table to see whether all courses taught by that deleted faculty member have been removed. Select the Course table from the list to open it, and you will find that all courses taught by the faculty member named Ying Bai, represented by the associated faculty\_id B78880, have been deleted from this table.

Then open the StudentCourse table, and you will find that the five courses taught by the deleted faculty member have also been deleted from this table.

Finally, open the Faculty table, and you will find that the faculty member Ying Bai has been deleted.

Our stored procedure is successful!

Before we can close Oracle database 10g XE, we highly recommend that you recover all data that has been deleted, by using the Insert Row button for each table. To recover the LogIn table, refer to Table 6.3 to add the deleted records into that table. For the Faculty, Course, and StudentCourse tables, refer to Tables 6.4, 6.5, and 6.6 to add those deleted records into those tables.

An important issue when you recover the deleted data by inserting it into the tables is the order in which you insert it into the associated tables. Since faculty\_id is the primary key in the Faculty table (parent table), you must first recover the data in the Faculty table. Then you can insert data into the LogIn and the Course tables

**Table 6.3.** The Data to Be Inserted into the LogIn Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	

**Table 6.4.** The Data to Be Inserted into the Faculty Table

faculty_id	name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Assistant Professor	ybai@college.edu

**Table 6.5.** The Data to Be Inserted 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 6.6.** The Data to Be Added to the StudentCourse Table

s_course_id	student_id	course_id	credit	major
1005	J77896	CSC-234A	3	CSIS
1009	A78835	CSE-434	3	CE
1014	A78835	CSE-438	3	CE
1016	A97850	CSC-132B	3	ISE
1017	A97850	CSC-234A	3	ISE

```

cmdDelete_Click
Click

Private Sub cmdDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click
    Dim cmdString As String = "DeleteFaculty"
    Dim vbButton As MessageBoxButtons = MessageBoxButtons.YesNo
    Dim oraCommand As New OracleCommand
    Dim Answer As DialogResult
    Dim intDelete As Integer
    Answer = MessageBox.Show("You sure you want to delete this record?", "Delete", vbButton)
    If Answer = System.Windows.Forms.DialogResult.Yes Then
        oraCommand.Connection = LoginForm.oraConnection
        oraCommand.CommandType = CommandType.StoredProcedure
        oraCommand.CommandText = cmdString
        oraCommand.Parameters.Add("FacultyName", OracleType.Char).Value = ComboName.Text
        intDelete = oraCommand.ExecuteNonQuery()
        oraCommand.Dispose()
        oraCommand = Nothing
    If intDelete = 0 Then
        MessageBox.Show("The data Deleting is failed")
        Exit Sub
    End If
    For intDelete = 0 To 6           'Initialize the object array
        FacultyTextBox(intDelete).Text = String.Empty
    Next intDelete
    End If
End Sub

```

**A**

**B**

**C**

Figure 6.46. Modified code to call the stored procedure.

in any order. For the Course and StudentCourse tables, first recover the Course table, and then recover the StudentCourse table.

Now close Oracle Database 10g XE. We need to develop the code to call that stored procedure from the Visual Basic.NET project to perform the data deletion function.

#### 6.8.4.2 Develop the Code to Call the Stored Procedure to Delete Records

The main coding job we need to perform is in the Delete button's Click event procedure in the Faculty form window, and we want to use this form as our GUI to perform the data deletion and validation functions. Open the project OracleUpdate-RTObjectSP we developed in the last section, open the form window of the Faculty form, and then double-click the Delete button to open its event procedure. Modify the coding in this event procedure, which is shown in Figure 6.46. The modified code is indicated in bold.

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

- A. The content of the query string is set to the name of the stored procedure. This name must be identical to the name of the stored procedure we created in the Oracle SQL Command page. This query string is used by the project as it runs, to locate the stored procedure we developed before.
- B. The CommandType property of the Command object must be set to the StoredProcedure to tell the Visual Basic.NET project that a stored procedure will be called to perform the associated functions as the project runs.
- C. The input parameter to our stored procedure is the FacultyName, and this parameter must be identical to the parameter's name that is assigned to the Command object's Parameters collection object.

Now let's run the project to test the data deletion function to delete existing data from our sample database. Click the Start Debugging button to start our project, enter a suitable username and password in the LogIn form, and select the Faculty Information item from the Selection form to open the Faculty form window. Keep the default faculty name in the ComboName combo box control unchanged, and click the Select button to retrieve the information on the selected faculty member from the database and display it in this form.

Now click the Delete button, and click Yes to confirm the deletion of this record from the Faculty table. The stored procedure DeleteFaculty() is called, and three DELETE commands are executed to delete all records related to the selected faculty member from the child tables, LogIn and Course, and to delete the selected faculty member from the parent table, Faculty.

To confirm this data deletion, keep the default faculty name selected from the ComboName combo box control, and click the Select button again to try to retrieve that deleted faculty information. The message "No matched faculty found" is displayed to indicate that the faculty member we tried to query has been deleted from the Faculty table.

Our project that calls a stored procedure to delete existing data from our sample database is successful.

It is highly recommended that you recover the deleted information for that selected faculty member for the four tables, LogIn, Faculty, StudentCourse and Course, in order to keep our sample database neat and complete. Refer to Tables 6.3–6.6 in the last section to recover the deleted records. One important point to note when you recover the deleted records is the order of the data recovery. The parent table should always be recovered first, and the child tables can be recovered in any order after the parent table is recovered.

In a similar way, you can create other stored procedures in the Oracle database to delete other records from our sample database. The completed project OracleUpdateRTOBJECTSP that calls a stored procedure to delete existing data from our sample database is located in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

## 6.9 CHAPTER SUMMARY

Data updating and deleting queries were discussed in this chapter with three popular databases: Access, SQL Server, and Oracle.

Three popular data updating and deleting methods were discussed and analyzed with seven real project examples:

1. Using TableAdapter DBDirect methods, such as TableAdapter.Update() and TableAdapter.Delete(), to update and delete data directly from the databases
2. Using the TableAdapter.Update() method to update and execute the associated TableAdapter's properties, such as UpdateCommand or DeleteCommand, to update changes made to the table in the DataSet to the table in the database
3. Using the runtime object method to develop and execute the Command object's method ExecuteNonQuery() to update or delete data from the database directly

Both methods 1 and 2 need to use Visual Basic.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 methods 1 and 2 is that method 1 can be used to directly access a database to perform 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 is performed on the associated tables in the DataSet, and then the updated or deleted data is updated in the tables in the database by executing the TableAdapter.Update() method.

This chapter was divided into two parts. Part I provided discussions of data updating and deleting using methods 1 and 2, or, in other words, using the TableAdapter.Update() and TableAdapter.Delete() methods developed with Visual Basic.NET design tools and wizards. Part II presented data updating and deleting using the runtime object method to develop command objects to execute the ExecuteNonQuery() method dynamically.

Seven real sample projects were provided in this chapter to help readers understand and design professional data-driven applications to update or delete data from three types of databases: Access, SQL Server, and Oracle. Stored procedures were discussed in the last section of each part to help readers perform data updating or deleting more efficiently and conveniently.

## 6.10 HOMEWORK

### I. True/False Selections

- \_\_\_\_\_ 1. Three popular data updating methods are TableAdapter DB-Direct, TableAdapter.Update(), and ExecuteNonQuery() of the Command class.
- \_\_\_\_\_ 2. Unlike the Fill() method, a valid database connection must be set before data can be updated in a database.
- \_\_\_\_\_ 3. We can directly update data or delete records from a 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 from the order of the data columns in the database.
- \_\_\_\_\_ 5. To update data in an Oracle database using stored procedures, an Oracle package must be developed to include the stored procedures.
- \_\_\_\_\_ 6. We can directly delete records from a database using TableAdapter DBDirect methods such as TableAdapter.Delete().
- \_\_\_\_\_ 7. When performing data updating, the same data can be updated in the database multiple times.
- \_\_\_\_\_ 8. To delete data from a database using the TableAdapter.Update() method, the data should be first deleted from the table in the DataSet, and then the Update() method should be executed to update that deletion to the table in the database.
- \_\_\_\_\_ 9. To update data in an SQL Server database using stored procedures, we can create and test the new stored procedure in the Server Explorer window.

- \_\_\_\_\_ 10. To call stored procedures to update data in a database, the parameters' names must be identical to the names of the input parameters defined in the stored procedures.

## II. Multiple Choices

1. To update data in a database using the TableAdapter.Update() method, we need to use \_\_\_\_\_ to build \_\_\_\_\_.
  - a. The data source, Query Builder
  - b. TableAdapter Query Configuration Wizard, an Update query
  - c. The runtime object, an Insert query
  - d. Server Explorer, the data source
2. To delete data from a database using the TableAdapter.Update() method, we need to first delete data from the \_\_\_\_\_ and then update that data in the database.
  - a. Data table
  - b. Data table in the database
  - c. DataSet
  - d. Data table in the DataSet
3. To delete data from a database using the TableAdapter.Update() method, we can delete \_\_\_\_\_.
  - a. One data row only
  - b. Multiple data rows
  - c. The whole data table
  - d. All of the above
4. Because ADO.NET provides a disconnected mode to the database, to update or delete a record from 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 an Oracle database.
  - a. =:
  - b. LIKE
  - c. =
  - d. @
6. To confirm a stored procedure built in the Object Browser page in an Oracle database, we can \_\_\_\_\_ the stored procedure to make sure it works.

- a. Build
  - b. Test
  - c. Debug
  - d. Compile
7. To confirm a stored procedure built in the Server Explorer window for an SQL Server database, we 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 type of the parameters in the SET list should be \_\_\_\_\_.
- a. OleDbType
  - b. SqlDbType
  - c. OracleDbType
  - d. OracleType
9. To update data using stored procedures, the CommandType property of the Command object must be equal to \_\_\_\_\_.
- a. CommandType.InsertCommand
  - b. CommandType.StoredProcedure
  - c. CommandType.Text
  - d. CommandType.Insert
10. To update data using stored procedures, the CommandText property of the Command object must be equal to \_\_\_\_\_.
- a. The content of the CommandType.InsertCommand
  - b. The content of the CommandType.Text
  - c. The name of the Insert command
  - d. The name of the stored procedure

### III. Exercises

1. The following is a stored procedure developed in an SQL Server database. Please develop a piece of code in Visual Basic.NET to call this stored procedure to update a record in the database.

---

```
CREATE OR REPLACE PROCEDURE dbo.UpdateStudent
( @Name IN VARCHAR(20),
  @Major IN text,
  @SchoolYear IN int,
  @Credits IN float,
```

```

@Email IN text
@StudentName IN VARCHAR(20))
AS
UPDATE Student SET name=@Name, major=@Major,
schoolYear=@SchoolYear, credits=@Credits,
email=@Email
WHERE (name=@StudentName)
RETURN

```

---

2. The following is a piece of code developed in Visual Basic.NET used to call a stored procedure in an Oracle database to update a record in the database. Please create the associated stored procedure in the Oracle database using PL/SQL.
- 

```

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()

```

---

3. Use the tools and wizards provided by Visual Basic.NET and ADO.NET to perform data updating for the Student form in the AccessUpdateDeleteWizard project (the project file is located in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354)).
4. Use runtime objects to complete the update data query for the Student form by using the project SQLUpdateDeleteRTOBJECT (the project file is located in the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354)).

5. Use a stored procedure to complete the data updating query for the Student form of the Student table by using the project OracleUpdateRTO-  
bjectSP (the project file is located in the folder **DBProjects\Chapter 6** at  
[www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354)).
6. Use a stored procedure to complete the data deleting query for the Student form of the Student table by using the project OracleUpdateRTO-  
bjectSP (the project file is located in the folder **DBProjects\Chapter 6** at  
[www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354)). It is highly recommended that you  
recover the deleted records after they are deleted.

**Hint:** Delete the related records from the LogIn, Course, and Student-  
Course tables, and then delete the record from the Student table.

## Accessing Data in ASP.NET

We provided a very detailed discussion of database programming with Visual Basic.NET using Windows-based applications in the previous chapters. Starting from this chapter, we will concentrate on database programming with Visual Basic.NET using Web-based applications. To develop a Web-based application and allow users to access a database through the Internet, you need to understand an important component: Active Server Page.NET or ASP.NET.

Essentially, ASP.NET allows you to write software to access databases through a Web browser rather than a separate program installed on computers. With the help of ASP.NET, you can easily create and develop an ASP.NET Web application and run it on the server as a server-side project. The user can then send requests to the server to download any Web page and 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).

After finishing this chapter, you will be able to

- Understand the structure and components of ASP.NET Web applications
- Understand the structure and components of the .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 a stored procedure to perform 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 of ASP.NET. But the prerequisite to understanding ASP.NET is the .NET Framework since ASP.NET is a part of the .NET Framework,

or in other words, the .NET Framework is a foundation of ASP.NET. So, first we need to have a detailed discussion of the .NET Framework.

## 7.1 WHAT IS THE .NET FRAMEWORK?

The .NET Framework is a model that provides a foundation to develop and execute different applications in an integrated environment such as Visual Studio.NET. In other words, the .NET Framework can be considered 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#.

The .NET Framework consists of the following components:

- *The Common Language Runtime (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 creation of Web and Windows forms.

The main goal of the .NET Framework is to overcome limitations on Web applications since different clients may provide different browsers. To solve these limitations, the .NET Framework provides a common language called Microsoft Intermediate Language (MSIL) that is language independent and platform independent. All programs developed in any .NET-based language can be converted into MSIL. MSIL can be recognized by the Common Language Runtime, which can compile and execute the MSIL code by using the Just-In-Time compiler.

You can access the .NET Framework by using the class libraries provided by the .NET Framework, and you can implement the .NET Framework by using the tools such as Visual Studio.NET provided by the .NET Framework, too. The 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 7.1.

The .NET Framework supports following three types of user interfaces:

- Windows forms that run on Windows 32 client computers. All the projects we developed in the previous chapters use this kind of user interface.
- Web forms that run on server computers through ASP.NET and the Hypertext Transfer Protocol (HTTP).
- The command console.

The advantages of using the .NET Framework to develop Windows-based and Web-based applications include but are not limited to the following:

- The .NET Framework is based on Web standards and practices, and it fully supports Internet technologies, including HTML, HTTP, XML, Simple Object Access Protocol (SOAP), XML Path Language (XPath), and other Web standards.

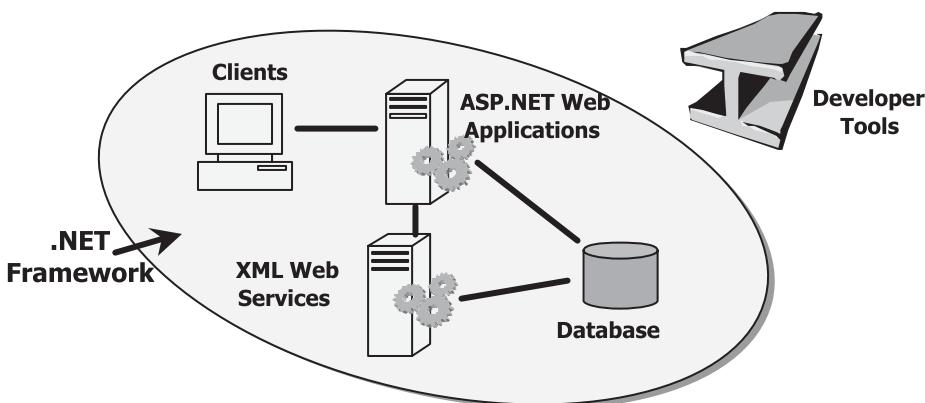


Figure 7.1. A .NET Framework model.

- 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 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 us have a closer look at ASP.NET.

## 7.2 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 a Web page that users view in a browser, and an ASP.NET Web application can contain one or more Web forms. A Web form is a dynamic page that can access server resources.

The complete structure of an ASP.NET Web application is shown in Figure 7.2.

Unlike a traditional Web page that can run scripts on the client side, an ASP.NET Web form can also run server-side code 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 of 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, an ASP.NET Web application can be developed in any .NET-based language.

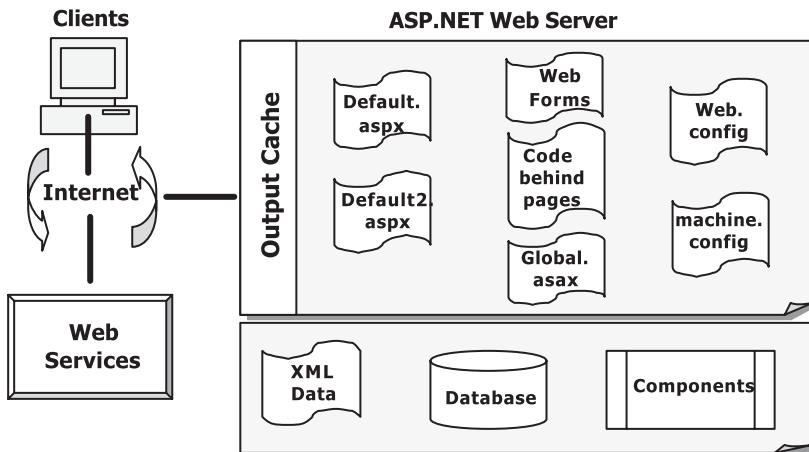


Figure 7.2. The structure of an ASP.NET Web application.

The ASP.NET technology also supports XML Web services. XML Web services are distributed applications that use XML for transferring information between clients, applications, and other XML Web services.

The main parts of an ASP.NET Web application include the following:

- *Web forms or Default.aspx pages*: These provide the user interface for the Web application, and they are very similar to the Windows forms in the Windows-based application. Web form files are indicated with an extension of .aspx.
- *Code-behind pages*: These are related to the Web forms and contain the server-side code for the Web form. The code-behind page is very similar to the code window for the Windows forms in the Windows-based applications discussed in the previous chapters. Most event procedures or handlers associated with controls on the Web forms are located in the code-behind page. 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 designated by the extension .asmx.vb.
- *Configuration files*: These 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*: This is also known as the ASP.NET application file, and it 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*: This allows the Web application to transfer data to and from database sources. Generally, it is not recommended to allow users to access

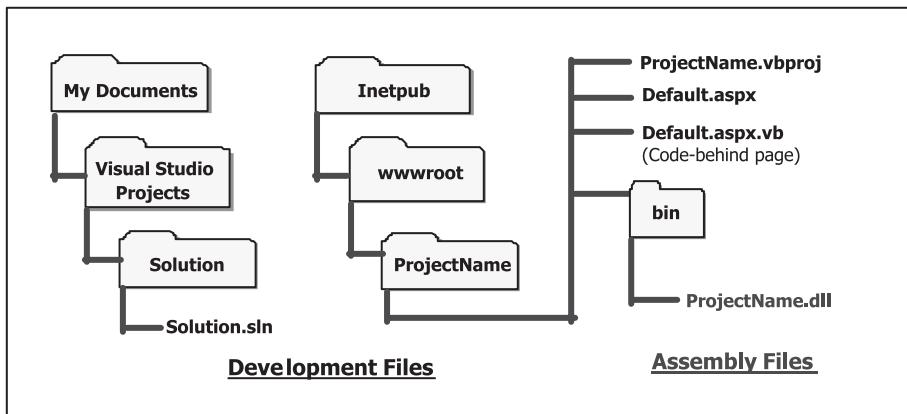


Figure 7.3. ASP.NET Web application file structure.

the database from the server directly because of security issues; instead, in most industrial and commercial applications, the database can be accessed through the application layer to strengthen the security of the database.

- *Caching:* This allows the Web application to return Web forms and data more quickly after the first request.

### 7.2.1 ASP.NET Web Application File Structure

When you create an ASP.NET Web application, Visual Studio.NET creates two folders to hold the files that relate to the application. When the project is compiled, a third folder is created to store the terminal .dll file. In other words, the final or terminal file of an ASP.NET Web application is a dynamic linked library file (.dll).

Figure 7.3 shows a typical file structure of an ASP.NET Web application.

The folders listed on the left side in Figure 7.3 are very familiar to us since they are created by Windows-based applications. But the folders created on the right side are new to us, and the functionalities of these folders are as follows:

- The Inetpub folder contains another folder named wwwroot and 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 are converted to a .dll file and stored in the server's memory.

### 7.2.2 ASP.NET Execution Model

When you finish 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 Web site and will be executed by the Web server as a request is received from the client at the runtime.
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 course code.
3. Based on the interpreting result, ASP.NET will direct the request to the associated assembly file (.dll) if the code has been compiled into the .dll files. Otherwise, ASP.NET invokes the compiler to convert the code into the .dll format.
4. Runtime loads and executes the MSIL code and sends back the required Web page to the client in the HTML file format.

The second time the user requests the same Web page, no compiling process is needed, and ASP.NET can directly call the .dll file and execute the MSIL code to speed up this request.

From this execution sequence, it looks like the execution of a Web application is easy and straightforward, but in practice, a lot of data round trips occur between the client and the server. To make it clear, let us perform a little more analysis and see what happens between the client and the server as a Web application is executed.

### 7.2.3 What Really Happens When a Web Application Is Executed?

The key point is that a Web form is built and runs on a Web server. When the user sends a request from the client browser to request a Web page, the server needs to build that form and send it back to the user's browser in HTML format. Once the Web page is received by the client's browser, the connection between the client and the server is terminated. If the user wants to request any other page or information from the server, additional requests must be submitted.

To make this issue more clear, we can use our LogIn form as an example. The first time the user sends a request to the server asking to start a login process, the server builds the LogIn form and sends it back to the client in HTML format. After that, the connection between the client and the server is closed. After the user receives the LogIn Web page and enters the necessary login 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 login information. If, after receiving and processing the login information, the server finds that the login information is invalid, then it needs to rebuild the LogIn form and resend it to the client with a warning message. So you can see how many round trips occur between the client and the server as a Web application is executed.

A good solution to try to reduce these round trips is to make sure that all information entered from the client side is as correct as possible. In other words, try to

do as much validation as possible on the client side to reduce the burden on the server.

Now we have finished the discussion about the .NET Framework and ASP.NET as well as ASP.NET Web applications. Next, we will develop some actual Web projects using ASP.NET Web forms to illustrate how to access a database through a Web browser to select, display, and manipulate data on Web pages.

### 7.2.4 The Requirements to Test and Run the Web Project

Before we can start to create our real Web project using ASP.NET, we need the following requirements:

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 Internet Information Services (IIS) on your local computer before the .NET Framework was installed on your computer, FrontPage Server Extensions 2000 should be installed on your local computer. This software allows your Web development tools such as Visual Studio.NET to connect to the server to upload or download pages from the server.
2. In this chapter, in order to make our Web project simple and easy, we always use our local computer as a pseudo server. In other words, we always use 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 manipulation.

If you have not installed IIS on your computer, follow the steps below to install this component:

- Click Start, then click Control Panel, and click Add or Remove Programs.
- Click Add/Remove Windows Components. The Windows Components Wizard appears (Figure 7.4).
- Check the check box for IIS from the list to add IIS to your computer. To confirm that this installation contains the installation of FrontPage 2000 Server Extensions, click the item IIS to select it and click the Details button, and you can find that the server has been checked. Although Microsoft has stopped supporting this version of the server and the current version is FrontPage 2002 Server Extensions, you can still use it without problem.
- Click Next to begin installing IIS and FrontPage 2000 Server Extensions to your computer.

The .NET Framework includes two Data Providers for accessing enterprise databases: the .NET Framework Data Provider for OLE DB and the .NET Framework Data Provider for SQL Server. Because there is no significant difference between the Microsoft Access database and the SQL Server database, in this chapter we use only the SQL Server database and the Oracle database as our target databases to illustrate how to select, display, and manipulate data from a database through Web pages.

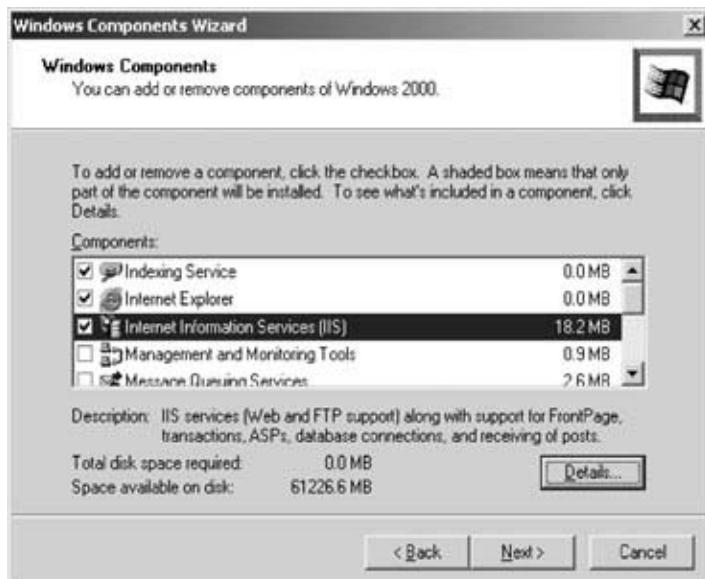


Figure 7.4. The opened Windows Components dialog box.

In this chapter, we will do the following:

1. Develop an ASP.NET Web application to select and display data from a Microsoft SQL Server database.
2. Develop an ASP.NET Web application to select and display data from an Oracle database.
3. Develop an ASP.NET Web application to insert data into a Microsoft SQL Server database.
4. Develop an ASP.NET Web application to insert data into an Oracle database.
5. Develop an ASP.NET Web application to update and delete data in a Microsoft SQL Server database.
6. Develop an ASP.NET Web application to update and delete data in an Oracle database.

Let's start by creating and building our ASP.NET Web application.

### 7.3 DEVELOP AN ASP.NET WEB APPLICATION TO SELECT DATA FROM SQL SERVER DATABASES

Let's create a new ASP.NET Web application project, SQLWebSelect, to illustrate how to access and select data from a database via the Internet. Open Visual Studio.NET and go to File|New Web Site to create a new ASP.NET Web application project.

In the New Web Site dialog box (Figure 7.5), keep the default template ASP.NET Web Site selected, keep the default contents of the Location and Language boxes unchanged, and then enter the project name SQLWebSelect into the box that is next to the Browse button, as shown in Figure 7.5. You can place your

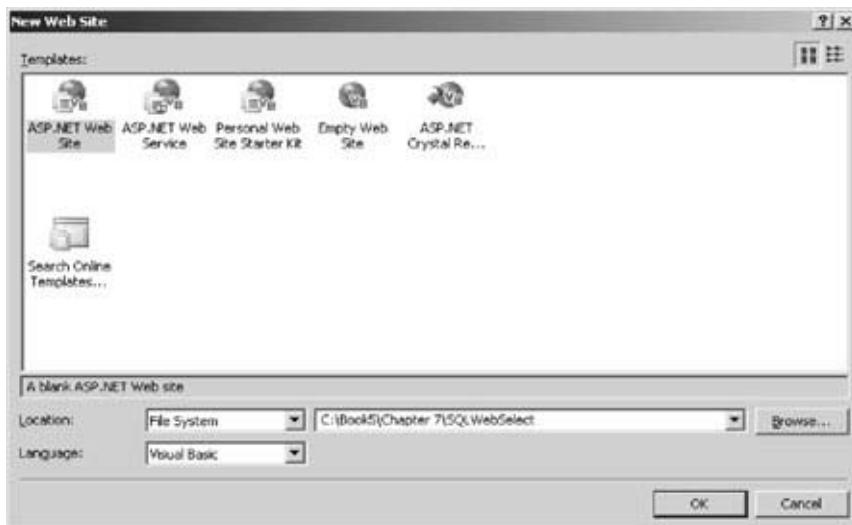


Figure 7.5. The opened New Web Site dialog box.

new project in any folder in your computer. In this case, we place it in the folder **C:\Book5\Chapter 7**. Click the OK button to create this new Web application project.

In the opened new project, the default Web form is named Default.aspx and is located in the Solution Explorer window. This Web form works as a user interface at the server side. Now let us perform some modifications to this form to make it our LogIn form.

### 7.3.1 Create the User Interface – LogIn Form

Right-click the Default.aspx item, select the Rename item from the popup menu, and change the name of this Web form to LogIn.aspx. Then we need to perform the associated modifications to the source file. Open the source file by clicking the Source tab at the bottom of the window.

You will find that the first line of the code is underscored with a blue line, which means that there is an error in this code. Move the cursor to the end of this line and change the value of the Inherits item from Inherits = “\_Default” to Inherits = “LogIn”.

The source file is basically an HTML file that contains the related code for all controls you added into this Web form. The difference between the code-behind page and the source file 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.

The following line of code indicates that this Web form will run at the server:

---

```
<form id="form1" runat="server">
```

---

**Table 7.1.** Controls for the Login Form

Type	ID	Text	TabIndex	BackColor	TextMode	Font
Label	Label1	Welcome to CSE DEPT	0	#E0E0E0		Bold/Large
Label	Label2	UserName	1			Bold/Medium
TextBox	txtUserName		2			
Label	Label3	Pass Word	3			Bold/Medium
TextBox	txtPassWord		4		Password	
Button	cmdLogin	Login	5			Bold/Medium
Button	cmdCancel	Cancel	6			Bold/Medium

Now click the View Designer button from the Solution Explorer window to open and design our Web form window.

Unlike the Windows-based application, by default the user interface in the Web-based application has no background color. You can modify the Web form by adding another style sheet and formatting the form as you like. Also, if you want to make a style such as the header and footer of the form apply to all 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.

Add the controls shown in Table 7.1 into our LogIn user interface or Web page.

One point to note 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 works a unique identifier for each control you add into the Web form.

Another difference from the Windows-based form is that when you add controls into the Web form, first you must locate a position for the control to be added using the space bar and the Enter key on your keyboard and then pick up a control from the Toolbox window and drag it to that location. You cannot pick and drag a control to a random location in this Web form, and this is a significant difference between the Windows-based form and the Web-based form.

Your finished user interface should match the one shown in Figure 7.6.

Before we can add the code into the code-behind page to respond to the controls to perform the login process, first we must run the project to allow the web.config file to recognize the controls we have added into the Web form. Click the Start Debugging button on the toolbar to run our project. Click OK at the prompt to add a Web.config file with debugging enabled. Your running Web page should match the one shown in Figure 7.6. Click the Close button that is located at the upper right corner of the form to close this page.

Now let us develop the code to access the database to perform the login process.

### 7.3.2 Develop the Code to Access and Select Data from the Database

Open the code-behind page by clicking 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



Figure 7.6. The finished LogIn Web form.

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
```

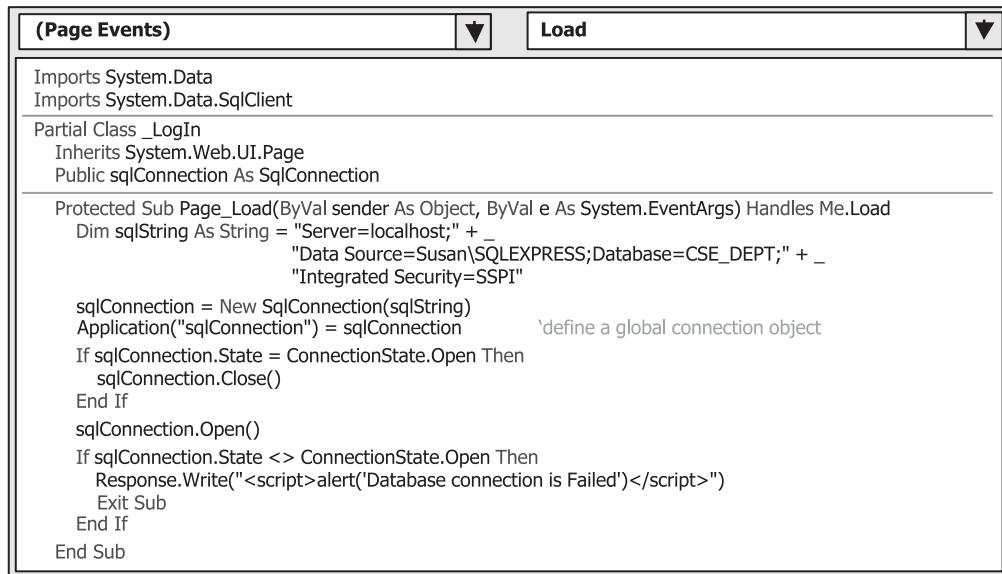
---

This connection object will be used by all Web forms in this project later.

Now we need to create the code 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 from the Method Name combo box to open this event procedure. Enter the code shown in Figure 7.7 into this event procedure.

Let us have a closer look at this piece of code to see how it works.

- A global connection object is declared first, and this object will be used by all Web forms in this project later to connect to the database.
- As we did for the `Form_Load()` event procedure in the Windows-based applications, we need to perform the database connection job in this event procedure. A connection string is created with the database server name, database name, and security mode.
- A new database connection object is created with the connection string as the argument.
- The global connection object `sqlConnection` is added into the `Application state` function, and this object can be used by any pages in this application



The screenshot shows a code editor window titled '(Page Events)' with a 'Load' button at the top right. The code is written in VB.NET and handles the Page\_Load event. It includes imports for System.Data and System.Data.SqlClient, defines a partial class \_LogIn that inherits from System.Web.UI.Page, and declares a SqlConnection named sqlConnection. The code then sets up a connection string (sqlString) for a local SQL Server Express database named Susan, specifies the database as CSE\_DEPT, and uses integrated security. It checks if the connection state is open; if not, it closes the connection and then opens it again. If opening fails, it writes an alert message to the browser.

```

Imports System.Data
Imports System.Data.SqlClient

Partial Class _LogIn
    Inherits System.Web.UI.Page
    Public sqlConnection As SqlConnection

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        Dim sqlString As String = "Server=localhost;" +
            "Data Source=Susan\SQLEXPRESS;Database=CSE_DEPT;" +
            "Integrated Security=SSPI"

        sqlConnection = New SqlConnection(sqlString)
        Application("sqlConnection") = sqlConnection      'define a global connection object

        If sqlConnection.State = ConnectionState.Open Then
            sqlConnection.Close()
        End If
        sqlConnection.Open()

        If sqlConnection.State <> ConnectionState.Open Then
            Response.Write("<script>alert('Database connection is Failed')</script>")
            Exit Sub
        End If
    End Sub

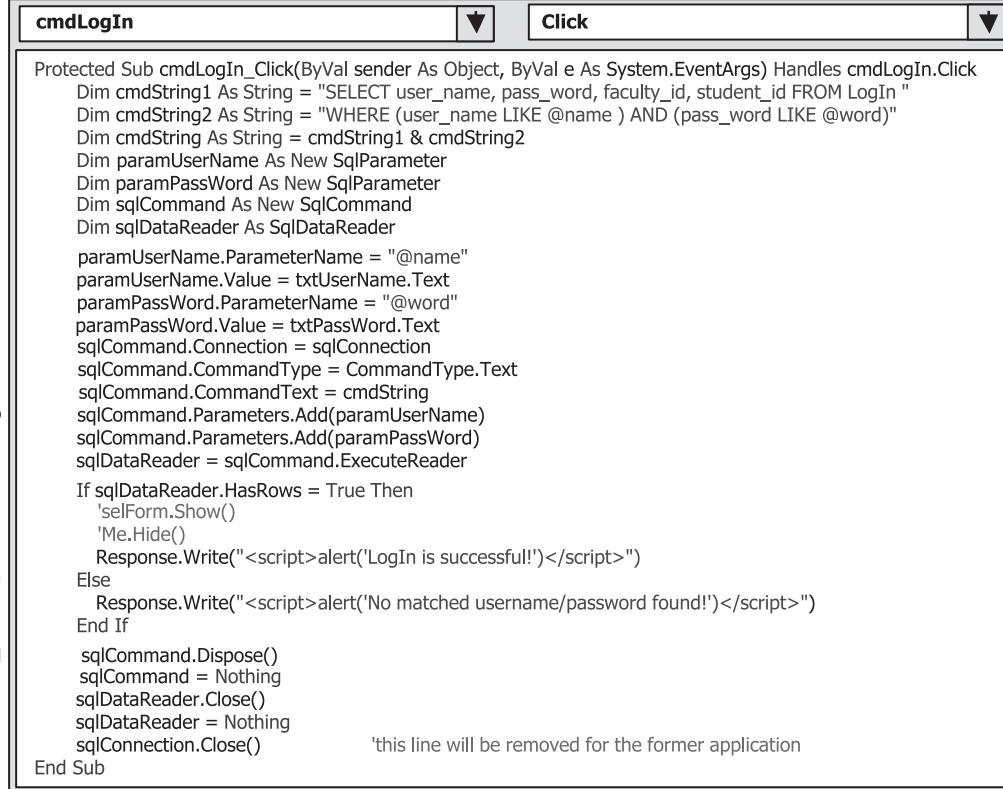
```

**Figure 7.7.** The code for the Page\_Load event procedure.

by accessing this Application state function later. Unlike global variables in the Windows-based applications, one cannot access a global variable from other pages by prefixing the form's name before the global variable declared in that form. In a Web-based application, the Application state function is a good place to store global variables. In an 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 accessing it is faster than storing and retrieving information in a database.

- E. First, we need to check whether this database is connected. If it is, we first need to disconnect this connection by using the `Close()` method.
- F. Then we can call the `Open()` method to set up the database connection.
- G. By checking the database connection state property, we can confirm the connection. 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 a message 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 from the server to the user. 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`.



The screenshot shows the Visual Studio IDE with the code editor open. The title bar says "cmdLogIn" and the tab bar says "Click". The code is a VB.NET event handler for the "cmdLogIn\_Click" event. It uses a SELECT query to find a user in a database table named "LogIn". The query includes WHERE clauses for both the username and password. It then creates SQL parameters for these values, initializes a command object, adds the parameters to it, and executes a reader. If rows are found, it shows a confirmation message. If not, it shows an error message. Finally, it disposes of the command and reader objects.

```

Protected Sub cmdLogIn_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdLogIn.Click
    Dim cmdString1 As String = "SELECT user_name, pass_word, faculty_id, student_id FROM LogIn "
    Dim cmdString2 As String = "WHERE (user_name LIKE @name) AND (pass_word LIKE @word)"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim paramUserName As New SqlParameter
    Dim paramPassWord As New SqlParameter
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader

    paramUserName.ParameterName = "@name"
    paramUserName.Value = txtUserName.Text
    paramPassWord.ParameterName = "@word"
    paramPassWord.Value = txtPassWord.Text
    sqlCommand.Connection = sqlConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add(paramUserName)
    sqlCommand.Parameters.Add(paramPassWord)
    sqlDataReader = sqlCommand.ExecuteReader

    If sqlDataReader.HasRows = True Then
        'selfForm.Show()
        'Me.Hide()
        Response.Write("<script>alert('LogIn is successful!')</script>")
    Else
        Response.Write("<script>alert('No matched username/password found!')</script>")
    End If

    sqlCommand.Dispose()
    sqlCommand = Nothing
    sqlDataReader.Close()
    sqlDataReader = Nothing
    sqlConnection.Close()      'this line will be removed for the former application
End Sub

```

Figure 7.8. The code for the LogIn button's Click event procedure.

Now let us create the code for the LogIn button's Click event procedure. The functionality of this piece of code 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 login information. Currently, since we have not created our next page – the Selection page – we just display a message to confirm the success of the login process. Click the View Design button from the Solution Explorer window and then double-click the LogIn button to open its event procedure. Enter the code shown in Figure 7.8 into this event procedure.

Let us have a closer look at this piece of code to see how it works.

- 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 one if you like.
- Some data objects are created here, such as the Command object, Data-Reader object, and Parameter objects.
- 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, it means that at least one matched username and password has been found and retrieved from the database. A success message is created and sent back from the server to the client to display in the client browser.
  - G. Otherwise, if no matched username or password is 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. Later on in our formal application, the database will be closed by the Selection Web form when the Exit button on that form is clicked by the user as the project runs.

Next, let us create the code for the Cancel button's Click event procedure.

The functionality 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 the Cancel button from the Design View of the LogIn form to open this event procedure, and enter the code shown in Figure 7.9 into it.

The functionality of this piece of code is as follows:

- 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 the Javascript statement window.close() to close the Web application.

At this point, we have finished developing the code for the LogIn Web form. Before we can run the project to test our Web page, we need to add some data validation functionalities in the client side to reduce the burden of the server.

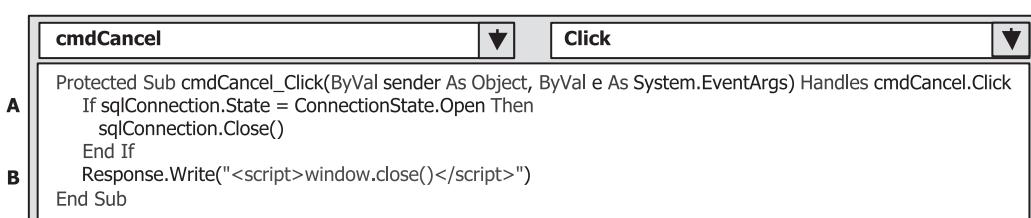


Figure 7.9. The code for the Cancel button event procedure.

**Table 7.2. Validation Controls**

<b>Validation Control</b>	<b>Functionality</b>
RequiredFieldValidator	Validate whether the required field has valid data (not blank).
RangeValidator	Validate whether a value is within a given numeric range. The range is defined by the MaximumValue and MinimumValue properties provided by users.
CompareValidator	Validate whether a value fits a given expression by using different Operator properties such as “equal”, “greater than”, and “less than” and the type of the value, which is set by the Type property.
CustomValidator	Validate a given expression using a script function. This method provides the maximum flexibility in data validation, but one needs to add a function to the Web page and send it to the server to get feedback from it.
RegularExpressionValidator	Validate whether a value fits a given regular expression by using the ValidationExpression property, which should be provided by the user.

### 7.3.3 Validate the Data in the Client Side

As we mentioned in Section 7.2.3, 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 send requests to the server, we need to make sure that the information to be sent to the server is as correct as possible. ASP.NET provides some tools to help us complete this data validation. These tools include the five validation controls that are shown in Table 7.2.

All these controls are located on the Validation tab in the Toolbox window in the Visual Studio.NET environment.

Here 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 click the Validation tab to expand it. Drag the RequiredFieldValidator control from the Toolbox window and place it next to the username text box. Set the following properties for this control in the property window:

- ErrorMessage: UserName is Required
- ControlToValidate: txtUserName

Perform similar dragging and placing operations to locate the second RequiredFieldValidator next to the password text box. Set the following properties for this control in the property window:

- ErrorMessage: PassWord is Required
- ControlToValidate: txtPassWord



**Figure 7.10.** Adding the data validation – RequiredFieldValidator.

Your finished LogIn Web form should match the one shown in Figure 7.10.

Now run our project to test this data validation by clicking the Start Debugging button, without entering any data into two text boxes, and then click 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 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.

Everything has its ups and downs, which is true in this project, too. After the RequiredFieldValidator is added to our Web page, the user cannot close the page by clicking the Cancel button if the username and password text boxes 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 text boxes are filled with valid information. So if you want to close the Web page now, you have to first enter a valid username and password, and then you can close the page by clicking the Cancel button.

### 7.3.4 Create the Second User Interface – Selection Page

Now let us continue to develop our Web application by adding another Web page, the Selection page. As for the projects in the previous chapters, after the login process, the next step is to allow users to select different functionalities from the Selection form to perform the associated database actions.

The Selection page allows users to visit different pages to perform database actions such as selecting, inserting, updating, or deleting data against the database via the different tables. So this Selection page needs to perform the following jobs:

1. Provide and display all available selections to allow users to select them.
2. Open the associated page based on the users' selection.

Now let us build this page. To do that, we need to add a new Web page. Right-click the project item from the Solution Explorer window and select Add New Item from the popup menu. In the opened window, keep the default template Web form

**Table 7.3.** Controls for the Selection Form

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	Make Your Selection:	0	#E0E0E0	Bold/Large
DropDownList	ComboSelection		1		
Button	cmdSelect	Select	2		Bold/Medium
Button	cmdExit	Exit	3		Bold/Medium

selected, enter Selection.aspx into the Name box as the name for this new page, and then click the Add button to add this page into our project.

On the opened Web form, add the controls in Table 7.3 to 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 bar or the Enter key from the keyboard to locate the positions on the page for those controls. Your finished Selection page should match the one shown in Figure 7.11.

Next, let us create the code for this Selection page to allow users to select the different pages to perform the associated data actions.

### 7.3.5 Develop the Code to Open the Other Pages

First, let us run the Selection page to build the Web configuration file. Click the Start Debugging button to run this page, and then click the Close button located at the upper right corner of the page to close it.

Click the View Code button from the Solution Explorer window to open the code page for the Selection Web form. First, 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
```

---



Figure 7.11. The finished Selection page.

The screenshot shows the Visual Studio Class View window. The left pane displays a tree structure of classes and methods. The right pane shows the code for the `Page_Load` event procedure. The code adds three items to a `ComboSelection` dropdown: "Faculty Information", "Course Information", and "Student Information".

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    ComboSelection.Items.Add("Faculty Information")
    ComboSelection.Items.Add("Course Information")
    ComboSelection.Items.Add("Student Information")
End Sub

```

**Figure 7.12.** The code for the Page\_Load event procedure of the Selection page.

Then select 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 code shown in Figure 7.12 into this event procedure to add all selection items into the `ComboSelection` combo box control.

The functionality of this piece of code is straightforward. Three pieces of information are added into the `ComboSelection` combo box 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 code for Click event procedures for two buttons. First, let us do the coding for the Select button. Click the View Designer button from the Solution Explorer window to open the Selection Web form, and then double-click the Select button to open its event procedure. Enter the code shown in Figure 7.13 into this event procedure.

The functionality of this piece of code is easy. Based on the information selected by the user, the related Web page is opened by using the server's `Response` object, or to be exact, by using the `Redirect()` method of the server's `Response` object. All these pages will be created and discussed in the following sections.

Finally, let us create the code for the Exit button's Click event procedure. The functionality of this piece of code is to close the database connection and the Web application. Double-click the Exit button from the Design View of the Selection page to open this event procedure. Enter the code shown in Figure 7.14 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.

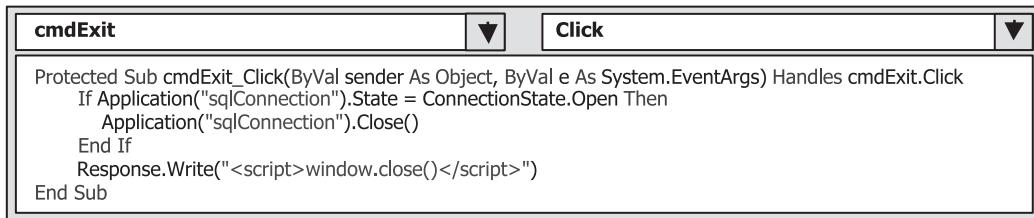
The screenshot shows the Visual Studio Class View window. The left pane displays a tree structure of classes and methods. The right pane shows the code for the `cmdSelect_Click` event procedure. The code uses an `If...ElseIf...Else...End If` structure to redirect the user to different pages based on the selected item in the `ComboSelection` dropdown.

```

Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    If ComboSelection.Text = "Faculty Information" Then
        Response.Redirect("Faculty.aspx")
    ElseIf ComboSelection.Text = "Student Information" Then
        Response.Redirect("Student.aspx")
    ElseIf ComboSelection.Text = "Course Information" Then
        Response.Redirect("Course.aspx")
    End If
End Sub

```

**Figure 7.13.** The code for the Select button event procedure.



**Figure 7.14.** The code for the Exit button event procedure.

Now we have finished the coding for the Selection page. Before we can run the project to test this page, first we need to make some modifications to the code 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 when the LogIn button is clicked as the project runs.

Open the LogIn page and the LogIn button's Click event procedure, and locate the following code inside the If block:

---

```
Response.Write("<script>alert('LogIn is successful!')</script>")
```

---

Replace that code with the following:

---

```
Response.Redirect("Selection.aspx")
```

---

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 address of the Selection page. Since the Selection page is located in the same application as the LogIn page, a direct page name is used.

Now let us 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 LogIn.aspx from the Solution Explorer window and select Set As Start Page from the popup menu. Click the Start Debugging button to run our project.

Enter a suitable username and password, such as ybai and reback, in the username and password boxes, and click the LogIn button. The Selection page is displayed if this login process is successful, as shown in Figure 7.15.

Click the Close button located at the upper right corner of the page to close the application since we have not yet developed any pages to select.

So far our Web application is good.

Now let us begin to develop our next page, the Faculty page.

### 7.3.6 Create the Third User Interface – Faculty Page

Right-click our project folder from the Solution Explorer window and select Add New Item from the popup menu. In the opened dialog box, keep the default



**Figure 7.15.** The running status of the second page – Selection page.

template Web form selected, enter Faculty.aspx into the Name box as the name for this new page, and then click the Add button to add this new page into our project.

On the opened Web form, add the controls shown in Table 7.4 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 bar or the Enter key 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.

One important point to note is the position of the Image control on the page form. When you pick up an Image control from the Toolbox window and drag it into the page window, you must set the Style property for that Image control first to allow it to be located at the desired position on the form without affecting the other controls we have already set up on the form. To do that, right-click on the Image control and select the Style item from the popup menu to open the Style Builder dialog box. In the opened dialog box, select the Layout item from the left column and click the drop-down arrow on the Allow floating objects combo box control, click the item on either side of that combo box control to select it, then click the OK button to close this dialog box. In this way, we set the style of this Image to a floating mode to allow it to be located in any position on our Web page.

Now you can enlarge this Image and place it in any location on this page by dragging it to the desired position.

Your finished Faculty page should match the one shown in Figure 7.16.

Although we have added five buttons into this Faculty page, in this section we only take care of the Select button and the Back button since we want to discuss how to retrieve data from the database based on the query command entered by the user and display the retrieved result in this Faculty page. The other buttons will be used in the following sections.

Now let us begin to develop the code for the Faculty page.

**Table 7.4.** Controls for the Faculty Form

Type	ID	Text	TabIndex	BackColor	Font
Label	Label1	CSE_DEPT Faculty Page	0	#E0E0E0	Bold/Large
Image	PhotoBox		22		
Label	Label2	Faculty Name	1		Bold/Medium
DropDownList	ComboName		2		
Label	Label3	Faculty ID	3		Bold/Medium
TextBox	txtID		4		
Label	Label4	Name	5		Bold/Medium
TextBox	txtName		6		
Label	Label5	Title	7		Bold/Medium
TextBox	txtTitle		8		
Label	Label6	Office	9		Bold/Medium
TextBox	txtOffice		10		
Label	Label7	Phone	11		Bold/Medium
TextBox	txtPhone		12		
Label	Label8	College	13		Bold/Medium
TextBox	txtCollege		14		
Label	Label9	Email	15		Bold/Medium
TextBox	txtEmail		16		
Button	cmdSelect	Select	17		Bold/Medium
Button	cmdInsert	Insert	18		Bold/Medium
Button	cmdUpdate	Update	19		Bold/Medium
Button	cmdDelete	Delete	20		Bold/Medium
Button	cmdBack	Back	21		Bold/Medium

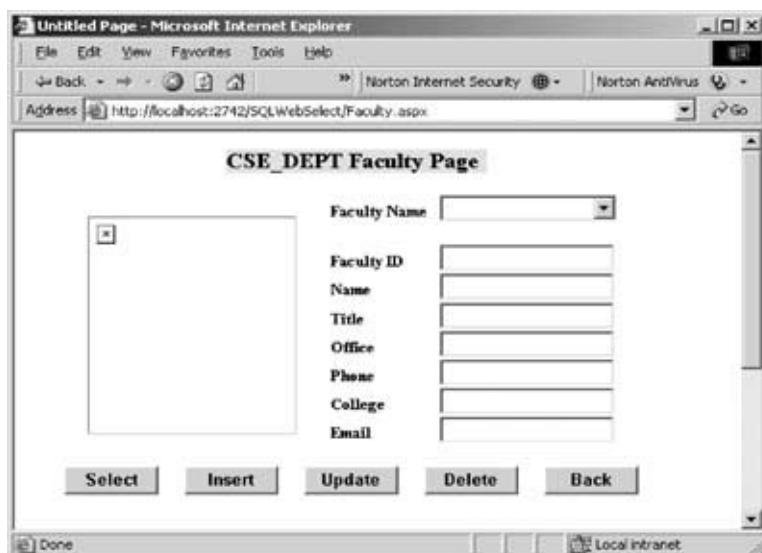


Figure 7.16. The finished Faculty page.

### 7.3.7 Develop the Code to Select the Desired Faculty Information

First, let us run the project to build the configuration file web.config to configure all controls we just added into the Faculty page. Click the Start Debugging button to run the project, and enter a suitable username and password to open the Selection page. Select the Faculty Information item from this page to open the Faculty page. Click the Close button located at the upper right corner of this page to close the project.

Open the code page of the Faculty form and, as we did before, first add two Imports commands to the top of this code page:

---

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

---

The code for this page can be divided into three parts: code for the Page\_Load() event procedure, code for the Select button's Click event procedure, and code for other procedures. First, let us take care of the code for the Page\_Load() event procedure.

#### 7.3.7.1 Develop the Code for the Page\_Load Event Procedure

In the opened code page, open the Page\_Load() event procedure by selecting Page Events from the Class Name combo box and Load from the Method Name combo box. Enter the code shown in Figure 7.17 into this event procedure.

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

(Page Events)		▼	Load	▼
	Imports System.Data Imports System.Data.SqlClient			
A	Partial Class Faculty Inherits System.Web.UI.Page Private FacultyTextBox(6) As TextBox		'Faculty table has 7 columns, we used all of them	
B	Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load			
	If Application("sqlConnection").State <> ConnectionState.Open Then			
	Application("sqlConnection").Open()			
	End If			
C	If Not IsPostBack Then			
	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")			
	End If			
	End Sub			

Figure 7.17. The code for the Page\_Load event procedure.

- A. A form-level text box array is created first since we need this array to hold six pieces of faculty information and display them in six text boxes 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, 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). The Web server will post back a refreshed Faculty page to the client when it receives this request (`IsPostBack = True`). When this happens, the `Page_Load` event procedure will be activated and the eight faculty names will be attached to the end of the `ComboName` combo box control again. To avoid this duplication, we need to check the `IsPostBack` property of the page and add the eight faculty members into the combo box control only one time, when the project starts (`IsPostBack = False`). Refer to Section 7.3.8.1 for more detailed discussion about the `AutoPostBack` property.

Next, we need to develop the code for the Select button's Click event procedure to perform the data query actions against the database.

### 7.3.7.2 Develop the Code for the Select Button Event Procedure

The functionality of this code is to make a query to the database to retrieve faculty information for the faculty member selected by the user from the `ComboName` combo box control, and display the retrieved information in six text box controls on the Faculty page.

Open the Select button's Click event procedure by double-clicking this button from the Design View of the Faculty form, and enter the code shown in Figure 7.18 into this event procedure.

Let us take a look at this piece of code to see how it works.

- A. The query string that contains a `SELECT` statement is declared here since we need to use this 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()` is called to display the selected faculty photo in the `Image` control on the Faculty page. We will develop this subroutine later.

```

cmdSelect Click
Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString As String = "SELECT faculty_id, name, office, phone, college, title, email FROM Faculty " & _
        "WHERE name LIKE @facultyName"
    Dim paramFacultyName As New SqlParameter
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader
    paramFacultyName.ParameterName = "@facultyName"
    paramFacultyName.Value = ComboName.Text
    sqlCommand.Connection = Application("sqlConnection")
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add(paramFacultyName)
    Call ShowFaculty(ComboName.Text)
    sqlDataReader = sqlCommand.ExecuteReader
    If sqlDataReader.HasRows = True Then
        Call FillFacultyReader(sqlDataReader)
    Else
        Response.Write("<script>alert('No matched faculty found!')</script>")
    End If
    sqlDataReader.Close()
    sqlDataReader = Nothing
    sqlCommand.Dispose()
    sqlCommand = Nothing
End Sub

```

Figure 7.18. The code for the Select button event procedure.

- 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 the six text boxes 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 is 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 have finished the coding for the Select button's Click event procedure.

### 7.3.7.3 Develop the Code for Other Procedures

Next, let us take care of the coding for other procedures in this Faculty page. This includes coding for the following procedures:

1. The user-defined subroutine procedure ShowFaculty()
2. The user-defined subroutine procedure FillFacultyReader()
3. The user-defined subroutine procedure MapFacultyTable()
4. The back button's Click event procedure

The third subroutine, MapFacultyTable(), is used and called by the second subroutine, FillFacultyReader(), in our project. Now let us discuss the coding for these subroutines one by one.

First, let's look at the code for the subroutine ShowFaculty(). The functionality 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 default location for the photo file is the current ASP.NET Web application folder. In our case, it is C:\....\SQLWebSelect. You must place all faculty photo 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 name of the photo file.

In this project, to make it simple, we use the default folder to store our faculty photo files.

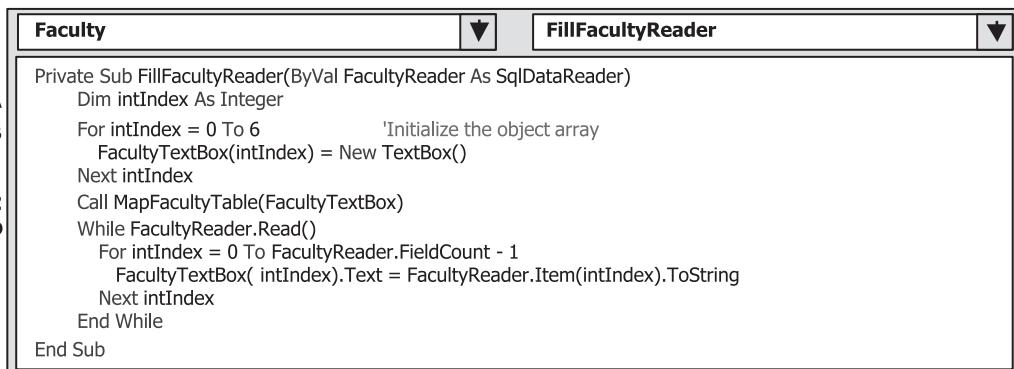
Open the code page of the Faculty page and create the subroutine shown in Figure 7.19.

Let's have a look at the code for this subroutine to see how it works.

- A. A local string variable, FacultyImage, is created, and it is used to hold the name of the matched faculty photo file.
- B. 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.

```
Faculty ShowFaculty
Private Sub ShowFaculty(ByVal fName As String)
    Dim FacultyImage As String
    Select Case fName
        Case "Ying Bai"
            FacultyImage = "Bai.jpg"
        Case "Satish Bhalla"
            FacultyImage = "Satish.jpg"
        Case "Black Anderson"
            FacultyImage = "Anderson.jpg"
        Case "Steve Johnson"
            FacultyImage = "Johnson.jpg"
        Case "Jenney King"
            FacultyImage = "King.jpg"
        Case "Alice Brown"
            FacultyImage = "Brown.jpg"
        Case "Debby Angles"
            FacultyImage = "Angles.jpg"
        Case "Jeff Henry"
            FacultyImage = "Henry.jpg"
        Case Else
            FacultyImage = ""
            Response.Write("<script>alert('No matched faculty image found!')</script>")
            Exit Sub
    End Select
    PhotoBox.ImageUrl = FacultyImage
End Sub
```

Figure 7.19. The code for the subroutine ShowFaculty.



```

Faculty FillFacultyReader
Private Sub FillFacultyReader(ByVal FacultyReader As SqlDataReader)
    Dim intIndex As Integer
    For intIndex = 0 To 6      'Initialize the object array
        FacultyTextBox(intIndex) = New TextBox()
    Next intIndex
    Call MapFacultyTable(FacultyTextBox)
    While FacultyReader.Read()
        For intIndex = 0 To FacultyReader.FieldCount - 1
            FacultyTextBox( intIndex).Text = FacultyReader.Item(intIndex).ToString()
        Next intIndex
    End While
End Sub

```

**Figure 7.20.** The code for the subroutine FillFacultyReader.

- C. If not, this means that no matched faculty photo file exists. A warning message is displayed in the client using the Write() method of the server Response object, and the procedure is exited.
- D. The name of the matched faculty photo file is assigned to the ImageUrl property of the Image control to display that photo.

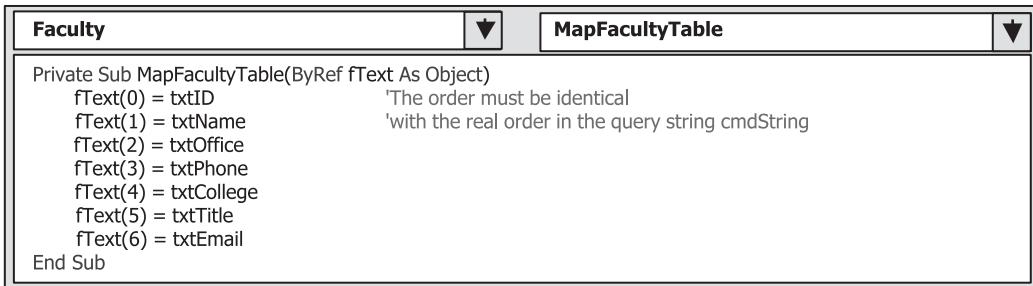
One significant difference in displaying an image in a Windows-based application and a Web-based application is that the ImageUrl property, which belongs to the control System.Web.UI.WebControls.Image, is utilized to access the matched faculty photo file and only the name of an image file is needed to display the associated image in a Web-based application. In a 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 code shown in Figure 7.20 to create this subroutine inside the Faculty class.

The functionality 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 text box on the Faculty page to display it.

Let's see how this piece of code works.

- A. A loop counter intIndex is declared.
- B. Seven instances of the object array, or text box 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 text box 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, which means that valid data is read out from the DataReader. This method will return a false if no valid data can be read out from the DataReader, which means that all data has been read out. In this application, in fact, this While loop is executed only one time since we have only one row (one record) to read out from the DataReader.



The screenshot shows the Visual Studio code editor with the following code:

```

Faculty MapFacultyTable
Private Sub MapFacultyTable(ByRef fText As Object)
    fText(0) = txtID
    fText(1) = txtName
    fText(2) = txtOffice
    fText(3) = txtPhone
    fText(4) = txtCollege
    fText(5) = txtTitle
    fText(6) = txtEmail
End Sub

```

The code is enclosed in a code block with a title bar labeled "Faculty" and "MapFacultyTable". There are two downward-pointing arrows at the top right of the code block.

**Figure 7.21.** The code for the MapFacultyTable subroutine.

- E. A For . . . Next loop is utilized to pick up each piece of data read out from the DataReader object and assign each of them to the associated text box control on the Faculty page window. The Item property with the index is used here to identify each piece of data from the DataReader.

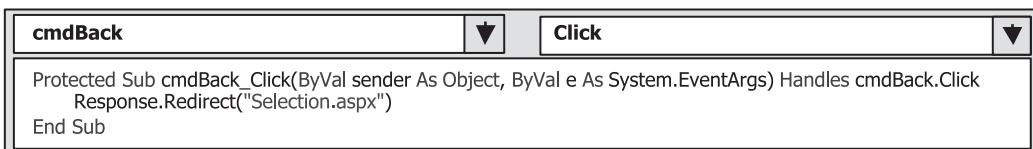
Now let us develop the code for the subroutine MapFacultyTable(). The functionality of this subroutine, as we mentioned, is to set up a correct mapping relationship between the seven text boxes in the text box array on the Faculty page and the seven data columns in the query string since the order in which the text boxes are displayed in the Faculty page may not be identical to the order of the data columns in the query string cmdString that 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 code shown in Figure 7.21 to create this subroutine inside the Faculty class.

The order of the seven text boxes 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 text box controls on the Faculty page window have a correct one-to-one relation with the queried columns in the query string cmdString.

Finally, let us take care of the code for the Back button's Click event procedure. The functionality of this piece of code is to return to the Selection page when this button is clicked. Double-click the Back button from the Faculty page window to open this event procedure and enter the code shown in Figure 7.22 into this event procedure.

This code 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. That is, the server resends the Selection page to the client when this button is clicked and a request is sent to the server.



The screenshot shows the Visual Studio code editor with the following code:

```

cmdBack Click
Protected Sub cmdBack_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdBack.Click
    Response.Redirect("Selection.aspx")
End Sub

```

The code is enclosed in a code block with a title bar labeled "cmdBack" and "Click". There are two downward-pointing arrows at the top right of the code block.

**Figure 7.22.** The code for the Back button event procedure.



Figure 7.23. The running status of the Faculty page.

Now we have finished all code development for the Faculty page. It is the time to run the project to test our pages. But before you run the project, make sure that you have stored all faculty photo files in the default location. Now click the Start Debugging button to run the project. Enter a suitable username and password, such as jhenry and test, in the LogIn page, and select Faculty Information from the Selection page to open the Faculty page. Select one faculty member from the combo box control, such as Ying Bai, and then click the Select button to retrieve the selected faculty member's information from the database. All information related to the selected faculty member is retrieved and displayed in this Faculty page, as shown in Figure 7.23.

Click the Back button to return to the Selection page, and then click the Exit button to terminate our project. So far our Web application is successful.

Next, we need to create our last Web page, the Course page, and add it to our project to select and display all courses taught by a selected faculty member.

### 7.3.8 Create the Fourth User Interface – Course Page

To create a new Web page and add it to our project, go to the Solution Explorer window and right-click our project folder. Select Add New Item from the popup menu to open the Add New Item dialog box. In the opened dialog box, keep the default template Web form selected, enter Course.aspx into the Name box as the name for our new page, and click the Add button to add it to our project.

On the opened Web form, add the controls that are shown in Table 7.5 into this page.

As mentioned before, you cannot place a control in any position on the form that you like. But now we introduce a technology to place a control in the desired position on the page window just as you did for your Windows-based application. The key element is the Position property inside the Style Builder for each control added into the page window. For example, in this Course page, we added two panel

**Table 7.5.** Controls for the Course Form

Type	ID	Text	TabIndex	BackColor	Font	AutoPostBack
Panel	Panel1		16	#COCOFF		
Label	Label1	Faculty Name	0		Bold/Smaller	
DropDownList	ComboName		1			
ListBox	CourseList		17		Bold/Medium	True
Panel	Panel2		18	#COCOFF		
Label	Label2	Course	2		Bold/Smaller	
TextBox	txtCourse		3			
Label	Label3	Schedule	4		Bold/Smaller	
TextBox	txtSchedule		5			
Label	Label4	Classroom	6		Bold/Smaller	
TextBox	txtClassRoom		7			
Label	Label5	Credit	8		Bold/Smaller	
TextBox	txtCredit		9			
Label	Label6	Enrollment	10		Bold/Smaller	
TextBox	txtEnrollment		11			
Button	cmdSelect	Select	12		Bold/Medium	
Button	cmdInsert	Insert	13		Bold/Medium	
Button	cmdUpdate	Update	14		Bold/Medium	
Button	cmdBack	Back	15		Bold/Medium	

controls, one list box control, five text box controls, and four button controls. In order to locate those controls in any position you like on the page, you must set the Position property, which is inside the Style Builder, of each control to Offset from normal flow. In this way, you can locate any control in any location on the page.

For example, to locate the panel in any location on the page, first add the panel to the page. Then follow the steps below:

1. Right-click the panel and select the Style item from the popup menu to open the Style Builder dialog box.
2. In the opened dialog box, click the Position tab from the left column.
3. Select Offset from normal flow from the Position mode list box.
4. Click the OK button to close this dialog box.

Perform a similar operation to define the Position property for each control shown in Table 7.5. Your finished Course page should match the one shown in Figure 7.24. Before we can continue to develop the following code, 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 between the Windows-based and Web-based applications.

### 7.3.8.1 The AutoPostBack Property of the List Box Control

One important property is the AutoPostBack property for the CourseList list box control. Unlike the list box control used in the Windows-based application, a



Figure 7.24. The finished Course Web page.

SelectedIndexChanged event will not be created in the server side if the user clicks and selects an item from the list box. The reason is that 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 when the user clicks and changes the item in the list box, a SelectedIndexChanged event can only be created in the client side but cannot be sent to the server. As you know, the list box is running at the server side when your project runs. So no matter how many times you click and change the items from the list box, no event will be created in the server side. Therefore, it appears that the project cannot respond to your clicking on the list box.

But 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 you click an item to select it from the list box, the AutoPostBack property will set a value to post back to the server to indicate that a user has interacted with the control.

In this section, we only discuss the coding for the Select and the Back buttons' Click event procedures to perform the course data query. The operations for the other buttons, such as Insert or Update, will be discussed in the following sections when we perform the data inserting or updating in the database using the Web pages.

Now let us develop the code for the Select and the Back buttons' click event procedures to pick up the course data from the database using the Course Web page.

### 7.3.9 Develop the Code to Select the Desired Course Information

The functionalities of the Course page are as follows:

1. When the user selects the desired faculty member from the Faculty Name combo box control and clicks the Select button, the IDs of all courses taught

by the selected faculty member should be retrieved from the database and displayed in the CourseList list box control on the Course page.

2. When the user clicks any course\_id from the CourseList list box control, the detailed course information related to the selected course\_id in the list box will be retrieved from the database and displayed in the five text boxes on the Course page form.

Based on the functionalities analyzed above, we need to concentrate on the coding for two event procedures; one is the Select button's click event procedure, and the second is the SelectedIndexChanged event procedure of the CourseList list box control. The first retrieves and displays all course\_id related to courses taught by the faculty member selected in the CourseList list box control, and the second retrieves and displays the detailed course information, such as course, schedule, classroom, credit, and enrollment, related to the course\_id selected from the CourseList.

The coding jobs above can be divided into four parts:

1. Coding for the Course page's 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 CourseList list box control.
4. Coding for other user-defined 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 these two Imports commands at the top of that page:

---

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

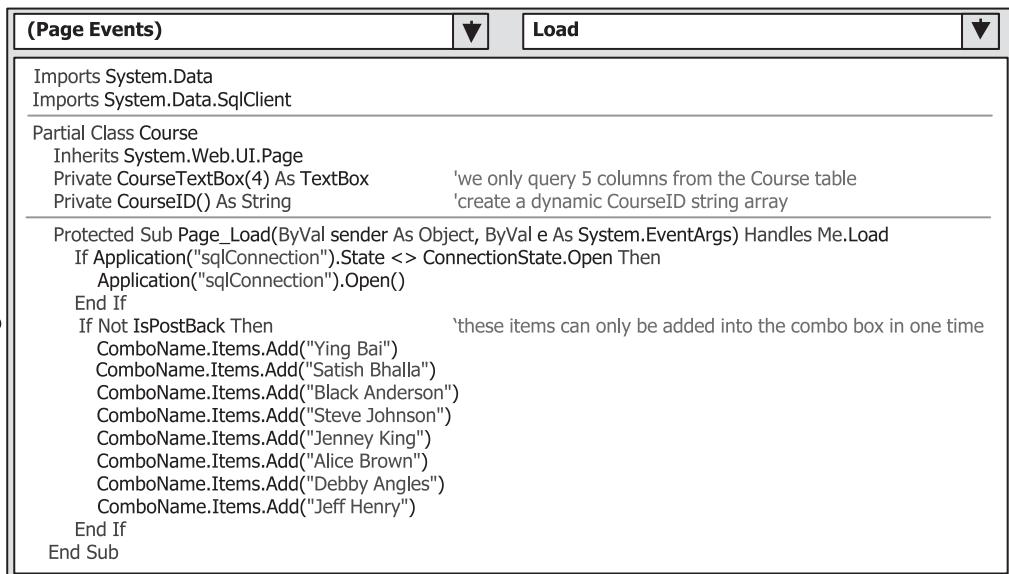
---

### 7.3.9.1 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 in the code window. Enter the code shown in Figure 7.25 into this event procedure.

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

- A. This code fragment is similar to one we used for the Faculty form. Five text box controls are used to display the detailed course information that is related to the faculty member selected from the Faculty Name combo box. The Course table has seven columns, but we only need five 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 4.
- B. Recall that in the first project, SelectWizard, the contents displayed in the CourseList box are the names of all the courses taught by the faculty member selected when the Select button is clicked in the Course form. Also, the



The screenshot shows a code editor window with the title bar '(Page Events)' and a 'Load' button. The code is divided into four sections labeled A, B, C, and D:

```

Imports System.Data
Imports System.Data.SqlClient

Partial Class CourseList
    Inherits System.Web.UI.Page
    Private CourseTextBox(4) As TextBox
    Private CourseID() As String

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        If Application("sqlConnection").State <> ConnectionState.Open Then
            Application("sqlConnection").Open()
        End If
        If Not IsPostBack Then
            'these items can only be added into the combo box in one time
            ComboName.Items.Add("Ying Bai")
            ComboName.Items.Add("Satish Bhalla")
            ComboName.Items.Add("Black Anderson")
            ComboName.Items.Add("Steve Johnson")
            ComboName.Items.Add("Jenney King")
            ComboName.Items.Add("Alice Brown")
            ComboName.Items.Add("Debby Angles")
            ComboName.Items.Add("Jeff Henry")
        End If
    End Sub

```

**Figure 7.25.** The code for the Page\_Load event procedure.

detailed information related to the course name selected in the CourseList box will be displayed in five text boxes when the user clicks one course name. Note that the names of the courses in the Course table are not unique, which means that one course may have two sections with the same name. For example, the courses CSC-132A and CSC-132B have the same course name, Introduction to Programming. An error may occur if one tries to use the course name as a criterion to query all detailed information related to that course since the course name is not unique in the Course table. In order to solve this problem, in this project we use the course\_id, which provides a unique value in the Course table, as a criterion to query the detailed information, such as course name, credit, classroom, schedule, and enrollment for that course. Because of the AutoPostBack property we discussed in Section 7.3.8.1, we need to use a page-level string array, CourseID(), to save each course\_id when it is selected. This CourseID array will be used in the next event procedure, CourseList\_SelectedIndexChanged, which will be triggered when the user clicks the course\_id from the CourseList box. Since the number of courses taught by each faculty member is different and cannot be known at this moment, we declare this string array as a dynamic one – blank parentheses that have no dimension follow the array name CourseID.

- C. Before we can perform a 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 is not connected to the database.
- D. The following code is 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. A potential bug exists for this piece of code. As we mentioned in Section 7.3.8.1, an AutoPostBack property will be set to True whenever the user clicks and selects an item from the list box control, and this property will be sent back to the server to indicate that an action has been taken by the user in this list box. After the server receives 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 will run again when a refreshed Course page is sent back. The result of the execution of this Page\_Load procedure is to attach another copy of all the faculty names to the end of the list that was already added into the ComboName combo box control when the Course page was first displayed. As the number of items clicked in the Course List box increases, the number of copies of the faculty names displayed in the ComboName box will also increase. To avoid this duplication, we only need to add all faculty members in the first time the Course page is displayed, but do nothing if an AutoPostBack property occurs.

The coding for the Back button's click event procedure is similar to that of the Back button 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 accomplish this switching back. Double-click the Back button from the Course page window and enter the following code into this event procedure:

---

```
Response.Redirect("Selection.aspx")
```

---

Now let's do the coding for the Select button's click event procedure.

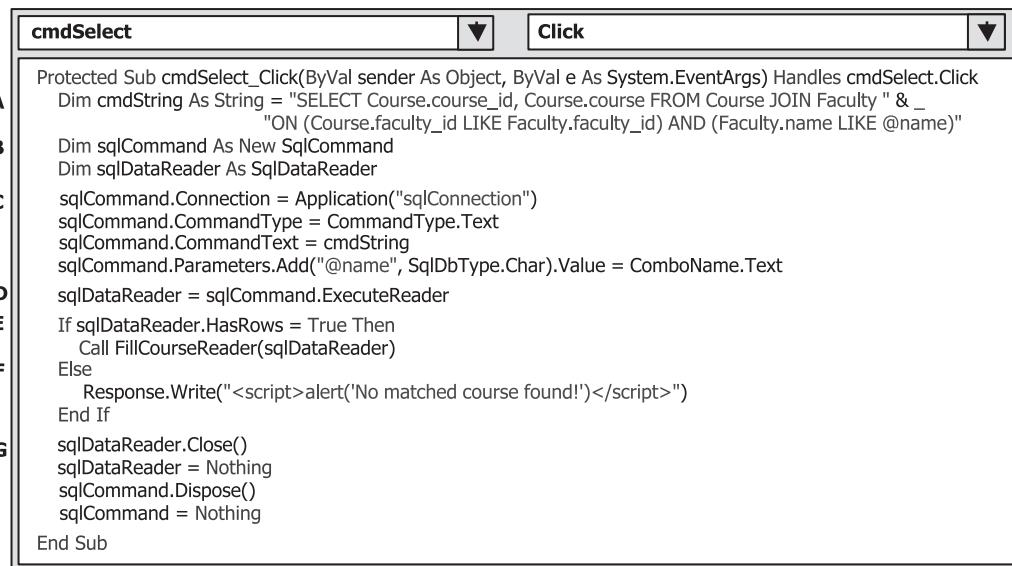
### 7.3.9.2 Coding for the Select Button's Click Event Procedure

As mentioned at the beginning of this section, the functionality of this event procedure is that when the user selects the desired faculty member from the Faculty Name combo box control and clicks the Select button, all course\_id related to courses taught by the selected faculty member should be retrieved from the database and displayed in the CourseList list box control on the Course page.

Double-click the Select button on the Course page window to open this event procedure, and enter the code shown in Figure 7.26 into this event procedure.

Let's have a look at this piece of code 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 one is the course name. The reason for this is that we need to use the course\_id, not the course name, as the identifier to pick up each course's detailed information from the Course table when the user clicks and selects the course\_id from the Course List box. We use the course\_id together with



```

cmdSelect_Click
Click

Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString As String = "SELECT Course.course_id, Course.course FROM Course JOIN Faculty " & _
        "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.name LIKE @name)"
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader
    sqlCommand.Connection = Application("sqlConnection")
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add("@name", SqlDbType.Char).Value = ComboName.Text
    sqlDataReader = sqlCommand.ExecuteReader
    If sqlDataReader.HasRows = True Then
        Call FillCourseReader(sqlDataReader)
    Else
        Response.Write("<script>alert('No matched course found!')</script>")
    End If
    sqlDataReader.Close()
    sqlDataReader = Nothing
    sqlCommand.Dispose()
    sqlCommand = Nothing
End Sub

```

Figure 7.26. The code for the Select button's event procedure.

the course name in this joined table query, and we will use the `course_id` later. The comparator `LIKE` is used to replace the original equal symbol for the criteria in the `ON` clause in the definition of the query string, which is required by the SQL Server database. For a more detailed description of the joined table query, refer to Section 4.18.6 in Chapter 4.

- B. Some SQL data objects, such as the `Command` and `DataReader`, are created here. All these objects should be prefixed by the keyword `sql` to indicate that those components are related to the SQL Server database.
- C. The `sqlCommand` object is initialized with the connection string, command type, command text, and command parameter. The parameter's name must be identical to the dynamic nominal name `@name`, which is defined in the query string and is located after the `LIKE` comparator in the `ON` clause. The parameter's value is the content of the Faculty Name combo box, which should be entered by the user as the project runs.
- D. The `ExecuteReader()` method of the `Command` class is executed to read back all courses taught by the selected faculty member and assign them to the `DataReader` object.
- E. If the `HasRows` property of the `DataReader` is `True`, which means that at least one row of data has been retrieved from the database, the `FillCourseReader()` subroutine is called to fill the `course_id` into the Course List box.
- F. Otherwise, this joined query fails and a warning message is displayed.
- G. Finally, some cleaning jobs are preformed to release objects used for this query.

Now let us take care of the coding for the user-defined procedure `FillCourseReader()`, which is shown in Figure 7.27. Open the code page of the Course Web

Course	▼	FillCourseReader	▼
A		Private Sub FillCourseReader(ByVal CourseReader As SqlDataReader)	
B		Dim strCourse As String	
C		Dim pos As Integer	
D		ReDim CourseID(9)	' redefine the CourseID string array
E		CourseList.Items.Clear()	
		While CourseReader.Read()	
		strCourse = CourseReader.GetSqlString(0)	' the 1st column is course_id
		CourseList.Items.Add(strCourse)	
		CourseID(pos) = Convert.ToString(CourseReader.GetSqlString(0))	
		pos = pos + 1	
		End While	
		Application("CourseID") = CourseID	' make the CourseID as a global array
		End Sub	

Figure 7.27. The code for the subroutine FillCourseReader.

form and enter the code shown in Figure 7.27 to create this procedure inside the Course class.

Let's see how this piece of code works.

- A. A local string variable strCourse is created. This variable can be considered an intermediate variable that is used to temporarily hold the queried data from the Course table. A local integer variable, pos, is created and it works as a loop counter later for the While loop and the index for the CourseID array.
- B. The dynamic string array CourseID is redefined here by a definite dimension number 9. Since the maximum number of courses taught by any faculty member in this department is less than 10, a string array of dimension 10 is defined to store up to 10 course\_id. This number can be modified based on your real project.
- C. Similarly, we need to clean up the Course List box before it can be filled.
- D. A While loop is utilized to retrieve the first column's data (GetString(0)), whose column index is 0 and data value is the course\_id. The queried data is first assigned to the intermediate variable strCourse, and then it is added into the Course List box by using the Add() method. Also, each retrieved first column's data course\_id is stored into the string array CourseID with an appropriate index (pos). The index pos is then updated by 1 for the next loop.
- E. Another potential bug exists in the page-level array variable CourseID. This variable is created in the client side and has nothing to do with the server. We will use this CourseID array when we perform another query to the Course table to get the detailed course information in the next event procedure, the Course List box's SelectedIndexChanged procedure. As mentioned, each time the user clicks and selects an item from the Course List box, an AutoPostBack signal is sent to the server, and the server sends back a refreshed Course page to the client after it receives this signal. The course\_id stored in the CourseID array will be gone after the page is refreshed. So in order to keep those course\_id stored in the CourseID array, we need to use an Application state to store that array as a global array variable.

CourseList	▼	SelectedIndexChanged	▼
------------	---	----------------------	---

```

Protected Sub CourseList_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    CourseList.SelectedIndexChanged
    Dim cmdString As String = "SELECT course, credit, classroom, schedule, enrollment FROM Course " & _
        "WHERE course_id LIKE @courseid"
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader
    Dim index As Integer
    index = CourseList.SelectedIndex
    sqlCommand.Connection = Application("sqlConnection")
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add("@courseid", SqlDbType.Char).Value = Application("CourseID")(index)
    sqlDataReader = sqlCommand.ExecuteReader
    If sqlDataReader.HasRows = True Then
        Call FillCourseReaderTextBox(sqlDataReader)
    Else
        Response.Write("<script>alert('No matched course information found!')</script>")
    End If
    sqlDataReader.Close()
    sqlDataReader = Nothing
    sqlCommand.Dispose()
    sqlCommand = Nothing
End Sub

```

Figure 7.28. The code for the SelectedIndexChanged event procedure.

Now let us develop the code for the SelectedIndexChanged event procedure of the CourseList list box control. The functionality of this event procedure is that when the user clicks any course.id from the CourseList list box control, the detailed course information related to the selected course.id, such as course name, schedule, credit, classroom, and enrollment, will be retrieved from the database and displayed in five text boxes on the Course page form.

### 7.3.9.3 Coding for the SelectedIndexChanged Event Procedure of the List Box Control

Double-click the CourseList list box control on the Course Web form to open this event procedure, and then enter the code shown in Figure 7.28 into this event procedure.

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

- The query string is created with five columns, such as course, credit, classroom, schedule, and enrollment. The query criterion is course.id, which was obtained from the FillCourseReader() subroutine above. 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, which is required for SQL Server database operation.
- Two SQL data objects are created, and these objects are used to perform the data operations between the database and our project. All these objects should be prefixed by the keyword sql since in this project we use an SQL Server Data Provider. A local integer variable, index, is also created here.
- The local variable index is used to hold the index of the item selected from the Course List box when the user selects that item.

- D. The sqlCommand object is initialized with the connection object, command type, command text, and command parameter. The parameter's name must be identical to the dynamic nominal name @courseid, which is defined in the query string after the LIKE comparator in the WHERE clause. The parameter's value is the element's value in the global array Application("CourseID")(index) that we obtained from the FillCourseReader() subroutines above. The index of the CourseID array is the SelectedIndex of the item selected from the Course List box.
- E. The ExecuteReader() method is executed to read back the detailed information for the selected course and assign it to the DataReader object.
- F. If the HasRows property of the DataReader is True, which means that at least one row of data has been retrieved from the database, the FillCourseReaderTextBox() subroutine is called to fill those pieces of information into five text boxes.
- G. Otherwise, this query fails and a warning message is displayed.
- H. Finally, some cleaning jobs are preformed to release objects used for this query.

The coding for other user-defined procedures includes the coding for the user-defined subroutine procedures FillCourseReaderTextBox() and MapCourseTable().

#### 7.3.9.4 Coding for Other User-Defined Procedures

First, let us develop the code for the procedure FillCourseReaderTextBox(). On the opened code page of the Course Web form, enter the code shown in Figure 7.29 inside the Course class to create this user-defined subroutine procedure.

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

- A. A loop counter, intIndex, is first created and is used for the loop of the creation of the text box object array and the loop of retrieving data from the DataReader later.
- B. The first loop is used to create the text box object array and perform the initialization for those objects.

Course		FillCourseReaderTextBox
A	Private Sub FillCourseReaderTextBox(ByVal CourseReader As SqlDataReader)	
	Dim intIndex As Integer	
B	For intIndex = 0 To 4	'Initialize the object array
	CourseTextBox(intIndex) = New TextBox	
	Next intIndex	
C	Call MapCourseTable(CourseTextBox)	
D	While CourseReader.Read()	
	For intIndex = 0 To CourseReader.FieldCount - 1	
	CourseTextBox(intIndex).Text = CourseReader.Item(intIndex).ToString	
	Next intIndex	
	End While	
	End Sub	

Figure 7.29. The code for the subroutine FillCourseReaderTextBox.

Course	▼	MapCourseTable	▼
<pre>Private Sub MapCourseTable(ByRef fCourse As Object)     fCourse(0) = txtCourse           'The order must be identical with the column order in the query     fCourse(1) = tx tCredit          'string – cmdString in CourseList_SelectedIndexChanged procedure     fCourse(2) = txtClassRoom     fCourse(3) = txtSchedule     fCourse(4) = txtEnrollment End Sub</pre>			

**Figure 7.30.** The code for the subroutine MapCourseTable.

- C. The subroutine MapCourseTable() is executed to set up a one-to-one relationship between each text box control on the Course page and each queried column in the query string. This step is necessary since the distribution order of the five text box controls on the Course page may be different from the column order in the query string.
- D. A While and a For . . . Next loop are used to pick up all five pieces of course-related information from the DataReader one by one. The Read() method is used as the While loop condition. A returned True means that valid data is read out from the DataReader, and a returned False means that no valid data is 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 a FieldCount of 1 as the termination condition since the index of the first data field is 0, not 1, in the DataReader object. Each piece of read-out data is converted to a string and assigned to the associated text box control in the text box object array.

The code for the subroutine MapCourseTable() is shown in Figure 7.30.

The functionality of this piece of code is straightforward. The order of the text boxes on the right-hand side of the equal operator is the column order of the cmdString query string. By assigning each column of required data to each of its partner text boxes in the text box object array in this order, a one-to-one relationship between each column of queried data and the associated text box control on the Course page is built, and the data retrieved from the DataReader can be mapped to the associated text box control and displayed there.

At this point, we have finished all the coding for the Course Web form. Now let us run the project to test the functionality of this form. Click the Start Debugging button to run the project. Enter a suitable username and password, such as jhenry and test, in the LogIn page, and select the Course Information item from the Selection page to open the Course page. On the opened page, select a faculty member from the combo box control and then click the Select button to retrieve all courses taught by that faculty member and display them in the Course List box. Your running result should match the one shown in Figure 7.31.

Click any course\_id from the Course List box to select it, and immediately the detailed information related to that selected course\_id is displayed in five text boxes, as shown in Figure 7.32.

Click the Back button to return to the Selection page, where you can click any other item from the Selection page to perform the associated information query or click the Exit button to terminate the application.

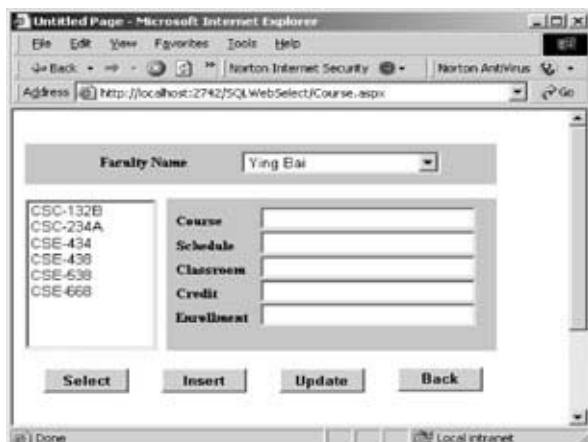


Figure 7.31. The running status of the Course page.

Our Web application is successful.

The completed Web application project used for data query in the SQL Server database, SQLWebSelect, is located in the folder **DBProjects\Chapter 7**, which is available at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

In the next section, we will discuss how to access the Oracle database to perform a data query.

## 7.4 DEVELOP AN ASP.NET WEB APPLICATION TO SELECT DATA FROM ORACLE DATABASES

Because of the similarity between SQL Server and Oracle database coding, we will emphasize the main differences between the coding for SQL Server and Oracle data actions. Also, in order to save time and space, we will modify the existing Web application project SQLWebSelect that was developed in the last section to make it our new project, OracleWebSelect, in this section.

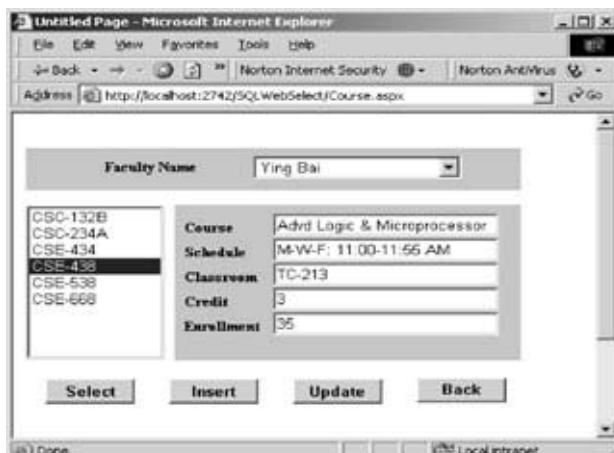


Figure 7.32. The detailed course information.

The main coding differences in these two database operations are in the following:

1. Connection string in the LogIn page
2. LogIn query string in the LogIn page
3. Query string in the Faculty page
4. Query strings in the Course page, which include the query string in the Select button's click event procedure and the query string in the SelectedIndexChanged event procedure of the CourseList box control
5. Imports commands and data objects used in the Selection page
6. Prefix for each data object and class used for the Oracle database operations
7. Data type of the passed arguments of procedures for Oracle database operations

Now let's begin to modify the project SQLWebSelect based on the seven differences listed above to make it our new project OracleWebSelect. Open the Windows Explorer and create a new folder, such as **Chapter 7**, if you have not created it. Copy the project SQLWebSelect from the folder **DBProjects\Chapter 7**, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and paste it in the folder **Chapter 7**. Rename the project OracleWebSelect.

Open Visual Studio.NET, go to the File|Open Web Site menu item and browse to our folder **Chapter 7**, select our new project OracleWebSelect, and then click the Open button to open it.

Let's first modify the code of the connection string in the LogIn page.

#### **7.4.1 Modify the Connection String in the LogIn Page**

Open the code page of the LogIn Web form by clicking it from the Solution Explorer window and then clicking the View Code button. The first thing we need to do is to add the Oracle data client reference to our project. To do that, right-click our project icon on the Solution Explorer window and select the Add Reference item from the popup menu to open the Add Reference dialog box. Browse down the list until you find the item System.Data.OracleClient. Click this item to select it, and click the OK button to add this reference to our project. Now we need to change the two Imports commands to set the namespace that contains the data components for the Oracle Data Provider. Modify the two Imports commands to the following:

---

```
Imports System.Data
Imports System.Data.OracleClient
```

---

Now open the Page\_Load event procedure by selecting the Page Events item from the Class Name combo box and the Load item from the Method Name combo box. Perform the following modifications to this event procedure:

- A. Change the prefix for the global connection object from sqlConnection to oraConnection since we need to use the Oracle data components in this section.

The screenshot shows the Visual Studio code editor with the title bar '(Page Events)' and a 'Load' button. The code is as follows:

```

Imports System.Data
Imports System.Data.OracleClient
Partial Class _LogIn
    Inherits System.Web.UI.Page
    Public oraConnection As OracleConnection
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    Dim oraString As String = "Data Source=XE;" + _
        "User ID=system;" + "Password=reback"
    oraConnection = New OracleConnection(oraString)
    Application("oraConnection") = oraConnection      'define a global connection object
    If oraConnection.State = ConnectionState.Open Then
        oraConnection.Close()
    End If
    oraConnection.Open()
    If oraConnection.State <> ConnectionState.Open Then
        Response.Write("<script>alert('Database connection is Failed')</script>")
        Exit Sub
    End If
End Sub

```

Annotations A through E are placed next to specific lines of code:

- A**: Points to the line 'Public oraConnection As OracleConnection'.
- B**: Points to the line 'Dim oraString As String = "Data Source=XE;" + \_ "User ID=system;" + "Password=reback"'.
- C**: Points to the line 'oraConnection = New OracleConnection(oraString)'.
- D**: Points to the line 'Application("oraConnection") = oraConnection'.
- E**: Points to the line 'If oraConnection.State = ConnectionState.Open Then oraConnection.Close()'.

Figure 7.33. The modified connection object and connection string.

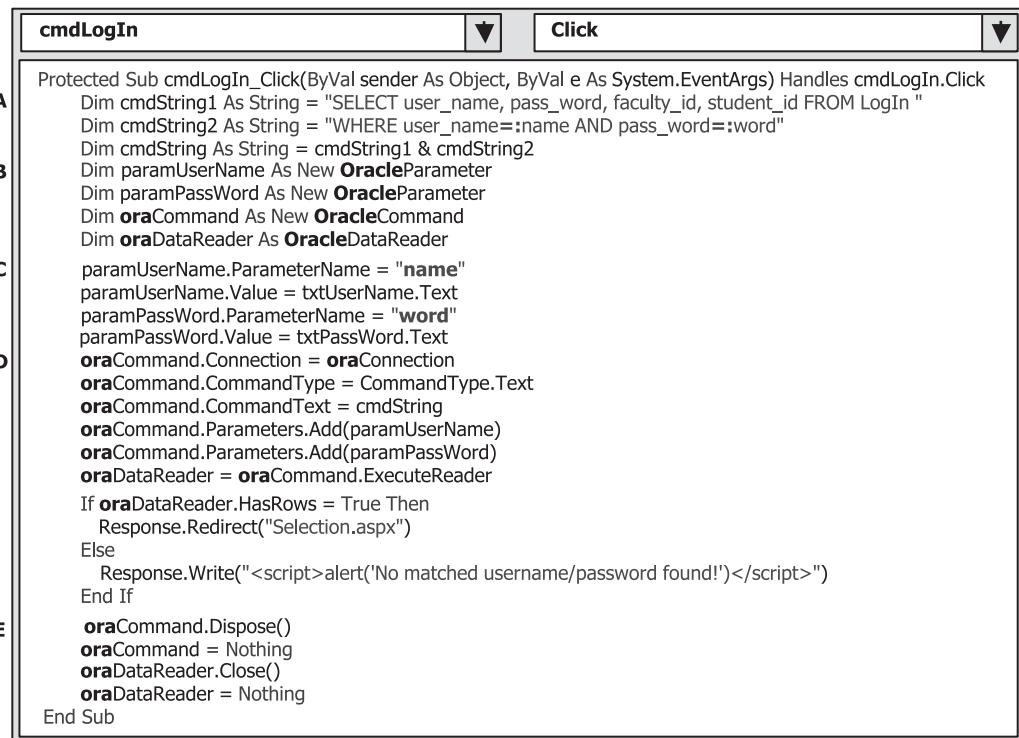
- Change the connection string to contain the user ID and password related to the Oracle database.
- Create a new instance of the Oracle connection class with the Oracle connection string oraString as the argument. Also change the prefix for all Oracle data objects and classes from Sql to Oracle and from sql to ora, respectively.
- Change the prefix for the global connection object stored in the Application state from sql to ora.
- 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 that shown in Figure 7.33. All modified parts have been highlighted in bold.

## 7.4.2 Modify the Query String in the LogIn Page

Now open the LogIn button's click event procedure and perform the following modifications to this event procedure.

- Change the query string from the SQL Server database format to the Oracle database format. The Oracle database assignment operator := is used to replace the SQL Server database assignment operator LIKE @.
- Change the prefix for all data components and classes from sql to ora and from Sql to Oracle, respectively.
- Change the nominal names for the dynamic parameters from @name to name and from @word to word, respectively.
- Change the prefix for all data components and classes from sql to ora and from Sql to Oracle, respectively.
- Change the prefix for all data components from sql to ora.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "cmdLogIn" and the tab bar says "Click". The code is written in VB.NET and handles the Click event of the cmdLogIn button. It uses Oracle objects like OracleParameter, OracleCommand, and OracleDataReader. The code is organized into five sections labeled A through E.

```

Protected Sub cmdLogIn_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdLogIn.Click
    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"
    Dim cmdString As String = cmdString1 & cmdString2
    Dim paramUserName As New OracleParameter
    Dim paramPassWord As New OracleParameter
    Dim oraCommand As New OracleCommand
    Dim oraDataReader As OracleDataReader
    paramUserName.ParameterName = "name"
    paramUserName.Value = txtUserName.Text
    paramPassWord.ParameterName = "word"
    paramPassWord.Value = txtPassWord.Text
    oraCommand.Connection = oraConnection
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add(paramUserName)
    oraCommand.Parameters.Add(paramPassWord)
    oraDataReader = oraCommand.ExecuteReader
    If oraDataReader.HasRows = True Then
        Response.Redirect("Selection.aspx")
    Else
        Response.Write("<script>alert('No matched username/password found!')</script>")
    End If
    oraCommand.Dispose()
    oraCommand = Nothing
    oraDataReader.Close()
    oraDataReader = Nothing
End Sub

```

Figure 7.34. The modifications to the codes in the LogIn button event procedure.

Your finished event procedure should match the one shown in Figure 7.34. All modified parts have been highlighted in bold.

Another modification to this page is the modification to the codes in the Cancel button's click event procedure. This modification is simple: just change the prefix of all data objects from sql to ora.

Go to the File|Save All menu item to save these modifications.

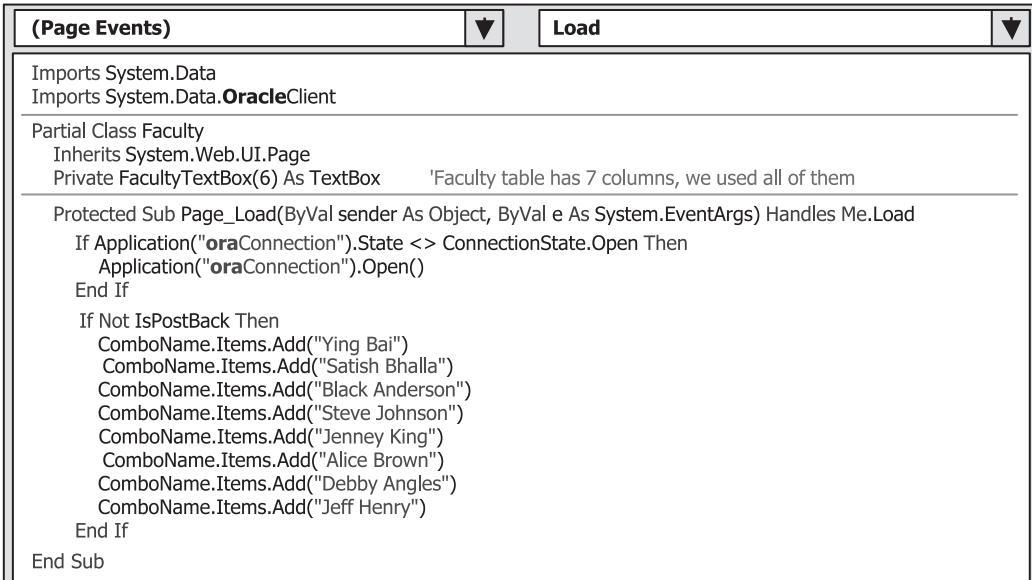
Now let's continue our modifications to the next page, the Faculty page.

### 7.4.3 Modify the Query String in the Faculty Page

The modifications to this page include the following:

1. Modifications to the Imports commands on the top of this page.
2. Modifications to the global connection object stored in the Application state in the Page\_Load event procedure.
3. Modifications to the code in the Select button's click event procedure.
4. Modifications to the data type of the passed argument in the user-defined subroutine FillFacultyReader().

Let's first modify the Imports commands. Replace the two Imports commands that are located at the top of this page with the following two commands:



The screenshot shows a code editor window with the title bar '(Page Events)' and a 'Load' button. The code is written in VB.NET:

```

Imports System.Data
Imports System.Data.OracleClient

Partial Class Faculty
    Inherits System.Web.UI.Page
    Private FacultyTextBox(6) As TextBox      'Faculty table has 7 columns, we used all of them

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        If Application("oraConnection").State <> ConnectionState.Open Then
            Application("oraConnection").Open()
        End If
        If Not IsPostBack Then
            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")
        End If
    End Sub

```

Figure 7.35. The modified Page\_Load event procedure.

---

```

Imports System.Data
Imports System.Data.OracleClient

```

---

Open the Page\_Load event procedure and change the connection object stored in the Application state from sqlConnection to oraConnection. Your finished event procedure should match the one shown in Figure 7.35. The modified parts have been highlighted in bold.

Now open the Select button's click event procedure and perform the following modifications:

- A. Change the query string by replacing the SQL Server database assignment operator LIKE @ with the Oracle database operator =: in the WHERE clause.
- B. Change the prefix for all data objects and classes from sql to ora and from Sql to Oracle, respectively.
- C. Modify the 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 sqlConnection to oraConnection.
- E. Change the prefix for all data objects and classes from sql to ora and from Sql to Oracle.

Your finished event procedure should match the one shown in Figure 7.36. All modified parts have been highlighted in bold.

```

cmdSelect Click
Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString As String = "SELECT faculty_id, name, office, phone, college, title, email FROM Faculty " & _
        "WHERE name=:facultyName"
    Dim paramFacultyName As New OracleParameter
    Dim oraCommand As New OracleCommand
    Dim oraDataReader As OracleDataReader
    paramFacultyName.ParameterName = "facultyName"
    paramFacultyName.Value = ComboName.Text
    oraCommand.Connection = Application("oraConnection")
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add(paramFacultyName)
    Call ShowFaculty(ComboName.Text)
    oraDataReader = oraCommand.ExecuteReader
    If oraDataReader.HasRows = True Then
        Call FillFacultyReader(oraDataReader)
    Else
        Response.Write("<script>alert('No matched faculty found!')</script>")
    End If
    oraDataReader.Close()
    oraDataReader = Nothing
    oraCommand.Dispose()
    oraCommand = Nothing
End Sub

```

Figure 7.36. The modifications to the Select button event procedure.

The modification to the data type of the passed argument in the user-defined subroutine FillFacultyReader() is simple: just change the data type of that passed argument from SqlDataReader to OracleDataReader.

#### 7.4.4 Modify the Query String in the Course Page

The modifications to this page include the following:

1. Modifications to the Imports commands on the top of this page.
2. Modifications to the global connection object stored in the Application state in the Page\_Load event procedure.
3. Modifications to the code in the Select button's click event procedure.
4. Modifications to the code in the SelectedIndexChanged event procedure of the CourseList list box control.
5. Modifications to the data type of the passed argument in the user-defined subroutines FillCourseReader() and FillCourseReaderTextBox().

Let's first modify the Imports commands. Replace the two Imports commands that are located at the top of this page with the following two commands:

---

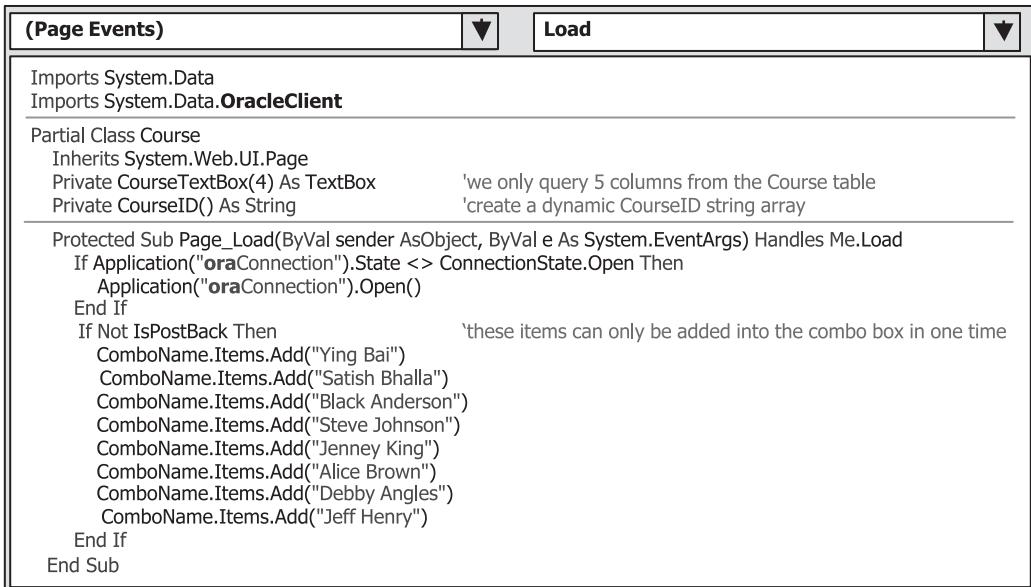
```

Imports System.Data
Imports System.Data.OracleClient

```

---

Open the Page\_Load event procedure and change the connection object stored in the Application state from sqlConnection to oraConnection. Your finished event



The screenshot shows the Visual Studio code editor with the title bar '(Page Events)' and a 'Load' button. The code is as follows:

```

Imports System.Data
Imports System.Data.OracleClient

Partial Class Course
    Inherits System.Web.UI.Page
    Private CourseTextBox(4) As TextBox
    Private CourseID() As String

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        If Application("oraConnection").State <> ConnectionState.Open Then
            Application("oraConnection").Open()
        End If
        If Not IsPostBack Then
            ComboName.Items.Add("Ying Bai") 'these items can only be added into the combo box in one time
            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")
        End If
    End Sub

```

Figure 7.37. The modified Page\_Load event procedure.

procedure should match the one shown in Figure 7.37. The modified parts have been highlighted in bold.

Now open the Select button's click event procedure and perform the following modifications:

- The query string applied for the joined table must be modified since the syntax of the query string used for the SQL Server database is the ANSI 92 standard. This standard is up-to-date, but it cannot be recognized by the Oracle database since the Oracle database still uses an old standard called ANSI 89 standard, as shown in Figure 7.38. In order to match the requirement of the Oracle database, the query string must be modified. Refer to Section 4.18.6 in Chapter 4 to get more detailed information about these two standards.
- Change the prefix for all data objects and classes from sql to ora and from Sql to Oracle, respectively.
- Modify the global connection object stored in the Application state from sqlConnection to oraConnection.
- Change the prefix for all data objects and classes from sql to ora and from Sql to Oracle.
- Modify the nominal name of the dynamic parameter by removing the @ symbol before the parameter **name**. Also change the data type for this dynamic parameter from SqlDbType to OracleType.

Your modified event procedure should match the one shown in Figure 7.38. All modified parts have been highlighted in bold.

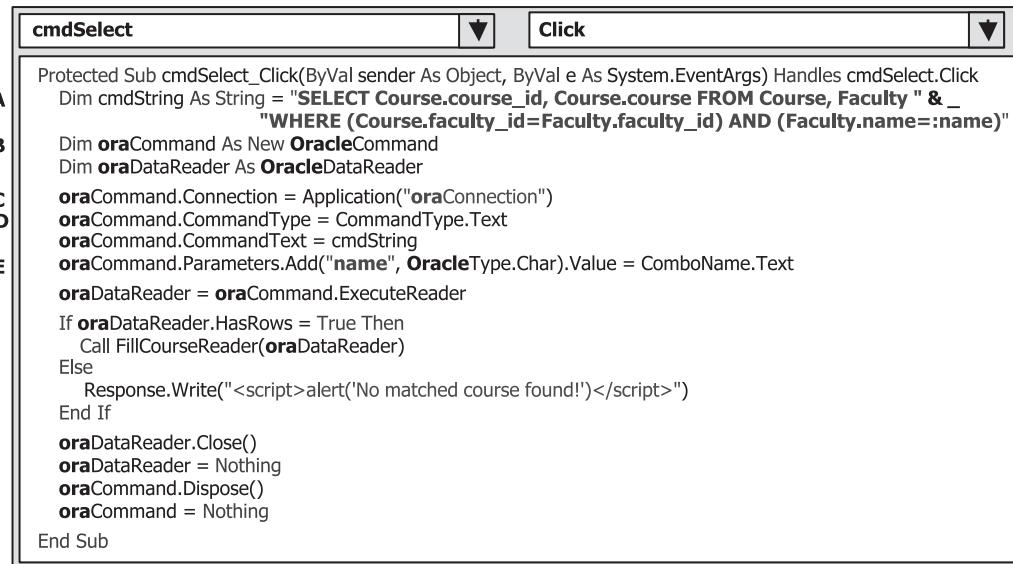


Figure 7.38. The modified Select button's event procedure.

Next, open the SelectedIndexChanged event procedure of the CourseList list box control, and perform the following modifications:

- A. Modify the assignment operator for the dynamic parameter courseid in the query string by replacing LIKE @ with the Oracle assignment operator =:.
- B. Change the prefix for all data objects and classes from sql to ora and from Sql to Oracle, respectively.
- C. Modify the global connection object stored in the Application state from sqlConnection to 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 OracleType.
- E. Change the prefix for all data objects and classes from sql to ora and from Sql to Oracle.

Your modified SelectedIndexChanged event procedure should match the one shown in Figure 7.39. All modified parts have been highlighted in bold.

Modifications to the data type of the passed argument in the user-defined subroutines FillCourseReader() and FillCourseReaderTextBox() are simple: just change the data type of that passed argument from SqlDataReader to OracleDataReader. Also change the name of the method from GetSqlString() to GetOracleString() in the subroutine FillCourseReader().

The last modification is to change the Imports commands and data objects used in the Selection page. Modify the Imports commands that are located at the top of this page to the following:

	CourseList	SelectedIndexChanged	
--	------------	----------------------	--

```

Protected Sub CourseList_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
    CourseList.SelectedIndexChanged
    Dim cmdString As String = "SELECT course, credit, classroom, schedule, enrollment FROM Course " & _
        "WHERE course_id =: courseid"
    Dim oraCommand As New OracleCommand
    Dim oraDataReader As OracleDataReader
    Dim index As Integer
    index = CourseList.SelectedIndex
    oraCommand.Connection = Application("oraConnection")
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add("courseid", OracleType.Char).Value = Application("CourseID")(index)
    oraDataReader = oraCommand.ExecuteReader
    If oraDataReader.HasRows = True Then
        Call FillCourseReaderTextBox(oraDataReader)
    Else
        Response.Write("<script>alert('No matched course information found!')</script>")
    End If
    oraDataReader.Close()
    oraDataReader = Nothing
    oraCommand.Dispose()
    oraCommand = Nothing
End Sub

```

Figure 7.39. The modified SelectedIndexChanged event procedure.

---

```

Imports System.Data
Imports System.Data.OracleClient

```

---

Modify the global connection object stored in the Application state from SqlConnection to oraConnection in the Exit button's click event procedure.

At this point, we have finished all modifications to the project. Before we run the project to test the functionalities of our code, note the following two points:

1. Make sure that all faculty photo files have been stored in the default folder, the one 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 our project icon in the Solution Explorer window and select the Start Options item from the popup menu to open the Property page. On the opened page window, select the Specific page radio button and click the ellipsis button next to the Specific page box to open the Select Page to Start dialog box. In the opened dialog box, click LogIn.aspx from the list, and click OK to select it as our start page. Finally, click the OK button on the Property page to finish this setup.

Now you can click the Start Debugging button to run the project to confirm the functionalities of our code.

The completed Web application project OracleWebSelect can be found in the folder **DBProjects\Chapter 7**, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

## 7.5 DEVELOP AN ASP.NET WEB APPLICATION TO INSERT DATA INTO SQL SERVER DATABASES

In this section, we discuss how to insert a new record into the SQL Server database from the Web page. To do that, we need to create a new Web page, the Insert page, and add it into our new project. To save time and space, we can modify an existing project, SQLWebSelect, which we developed in the previous section, and make it into our new Web application project named SQLWebInsert.

Open the Windows Explorer and create a new folder, **Chapter 7**, if you have not already done. Then copy the project SQLWebSelect from the folder **DBProjects**, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and paste it into our new folder **Chapter 7**. Rename this project SQLWebInsert.

As we mentioned, to add a new record into the database, we need a user interface. So we need to create and add a new Web page called Insert.aspx into our new project to perform the data insertion functionality.

### 7.5.1 Create a New Web Page – Insert.aspx

Right-click our new project icon in the Solution Explorer window and select Add New Item from the popup menu to open the Add New Item dialog box. Keep the default template Web form selected, and enter Insert.aspx into the Name box as the name for this new page. Click the Add button to add this new page into our new project.

Add the controls shown in Table 7.6 into this Web page.

**Table 7.6. Controls for the Insert Form**

Type	ID	Text	TabIndex	BackColor	Font
Panel	Panel1		19	#COCOFF	
Label	Label1	Faculty Photo	0		Bold/Smaller
TextBox	txtPhoto		1		
Panel	Panel2		20	#COCOFF	
Label	Label2	Faculty ID	2		Bold/Smaller
TextBox	txtID		3		
Panel	Panel3		21	#COCOFF	
Image	PhotoBox		4		
Label	Label3	Name	5		Bold/Smaller
TextBox	txtName		6		
Label	Label4	Title	7		Bold/Smaller
TextBox	txtTitle		8		
Label	Label5	Office	9		Bold/Smaller
TextBox	txtOffice		10		
Label	Label6	Phone	11		Bold/Smaller
TextBox	txtPhone		12		
Label	Label7	College	13		Bold/Smaller
TextBox	txtCollege		14		
Label	Label8	Email	15		Bold/Smaller
TextBox	txtEmail		16		
Button	cmdInsert	Insert	17		Bold/Medium
Button	cmdBack	Back	18		Bold/Medium

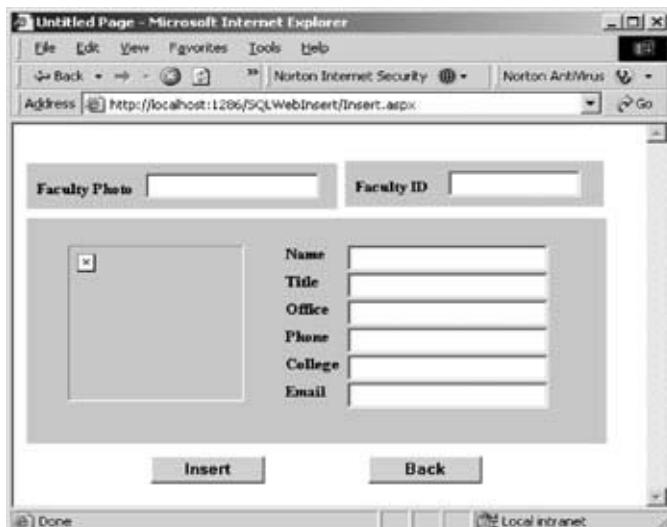


Figure 7.40. The Insert Web page.

The key points to note when adding these controls to the page are as follows: for the three panels, set the Style|Position property to Offset from normal flow, and set the Style|Position property for all Label and TextBox controls to Absolutely position. You can move, copy, and paste those Label and TextBox controls one by one on the page form to save time.

Your finished Insert page should match the one shown in Figure 7.40.

### 7.5.2 Develop the Code to Perform the Data Insertion Functionality

The functionalities of this Insert page are as follows:

1. When the project runs, you need to open the Insert page by clicking the Insert button from the Faculty page.
2. To insert a new faculty record into the database, you need to enter seven pieces of new information into seven text boxes on the Insert page. The information includes the faculty\_id, faculty\_name, title, office, phone, college, and email.
3. The Faculty Photo text box is optional, which means that you can either enter a new faculty photo name with this new record or leave it blank. If you leave it blank, a default photo will be displayed when this new record is validated later.
4. After all the information has been filled into the text boxes, you can click the Insert button to insert this new record into the Faculty table in the database via the Web page.
5. The Back button is used to return to the Faculty page to perform the data validation to confirm this data insertion.

Now let's create the code for this Insert page. The coding can be divided into three parts: the coding for the Page\_Load and Back button's click event procedures,

The screenshot shows the Visual Studio code editor with the title bar '(Page Events)' and the method name 'Load' selected. The code is as follows:

```

Imports System.Data
Imports System.Data.SqlClient

Partial Class Insert
    Inherits System.Web.UI.Page
    Public FacultyName As String      'used for validation later by Faculty page.

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        If Application("sqlConnection").State <> ConnectionState.Open Then
            Application("sqlConnection").Open()
        End If
    End Sub

```

**Figure 7.41.** The code for the Page\_Load event procedure.

the coding for the Insert button's click event procedure, and the coding for other procedures.

### 7.5.2.1 Develop the Code for the Page\_Load and Back Button Event Procedures

Open the code page window of the Insert Web form, and enter the following two Imports commands at the top of this page:

---

```

Imports System.Data
Imports System.Data.SqlClient

```

---

Open the Page\_Load event procedure by selecting Page Events from the Class Name combo box and selecting the Load item from the Method Name combo box. Enter the code shown in Figure 7.41 into this event procedure.

The functionality of this piece of coding is as follows:

- A. A global string variable, FacultyName, is created first since this variable will work as a storage unit to save the newly inserted faculty name. This variable will be added into the ComboName combo box control on the Faculty page as a new faculty name that works as a criterion to validate this data insertion later.
- B. The global connection object sqlConnection stored in the Application state will be checked to make sure that a valid database connection exists before this data insertion. Redo this connection if no valid connection exists.

The coding for the Back button's click event procedure is simple. When this button is clicked, the current page will be returned to the Faculty page to perform the data validation. Open this event procedure by double-clicking the Back button from the Insert Web form, and enter the following code into this event procedure:

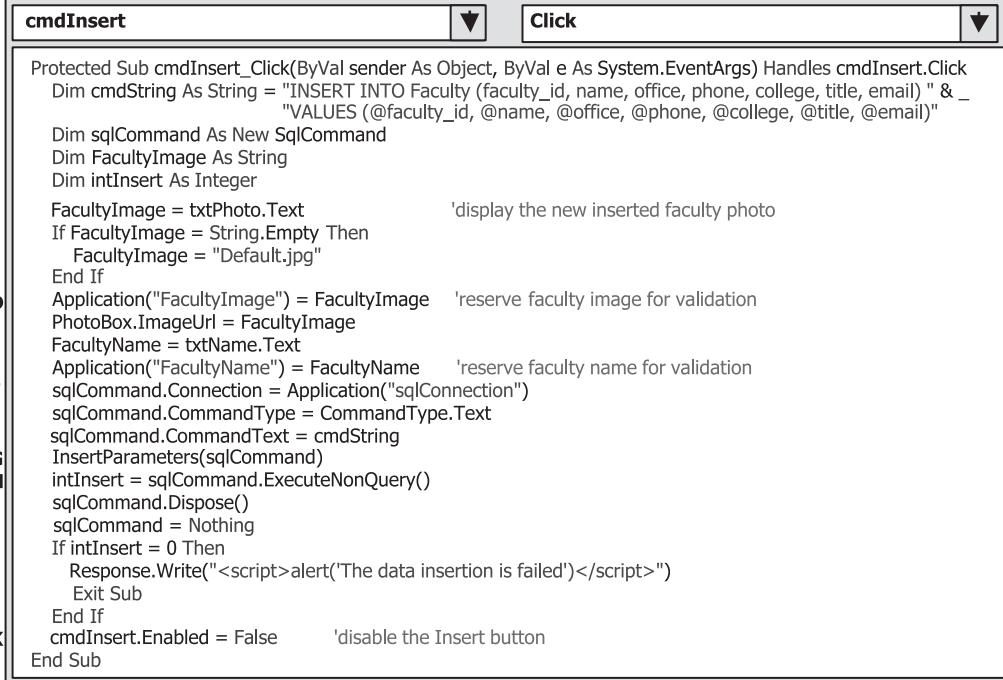
---

```

Response.Redirect("Faculty.aspx")

```

---



The screenshot shows the Visual Studio code editor with the title bar "cmdInsert" and the tab "Click". The code is a VB.NET event handler for the "cmdInsert\_Click" event. The code uses Application state to store the faculty image and name, and it handles the execution of an SQL INSERT query. It also checks if a photo was uploaded and sets the FacultyImage variable accordingly. If no photo was uploaded, it uses a default image. After executing the query, it checks if the insertion was successful. If not, it writes an error message to the response. Finally, it disables the insert button.

```

Protected Sub cmdInsert_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim cmdString As String = "INSERT INTO Faculty (faculty_id, name, office, phone, college, title, email) " & _
        "VALUES (@faculty_id, @name, @office, @phone, @college, @title, @email)"
    Dim sqlCommand As New SqlCommand
    Dim FacultyImage As String
    Dim intInsert As Integer
    FacultyImage = txtPhoto.Text
    If FacultyImage = String.Empty Then
        FacultyImage = "Default.jpg"
    End If
    Application("FacultyImage") = FacultyImage 'reserve faculty image for validation
    PhotoBox.ImageUrl = FacultyImage
    FacultyName = txtName.Text
    Application("FacultyName") = FacultyName 'reserve faculty name for validation
    sqlCommand.Connection = Application("sqlConnection")
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    InsertParameters(sqlCommand)
    intInsert = sqlCommand.ExecuteNonQuery()
    sqlCommand.Dispose()
    sqlCommand = Nothing
    If intInsert = 0 Then
        Response.Write("<script>alert('The data insertion is failed')</script>")
        Exit Sub
    End If
    cmdInsert.Enabled = False 'disable the Insert button
End Sub

```

Figure 7.42. The code for the Insert button's click event procedure.

The `Redirect()` method of the server `Response` object is used to redirect the current page to go to the Faculty page.

### 7.5.2.2 Develop the Code for the Insert Button's Click Event Procedure

Open the Insert button's click event procedure by double-clicking the Insert button from the Insert Web form, and enter the code shown in Figure 7.42 into this event procedure.

Let us have a closer look at this piece of code to see how it works.

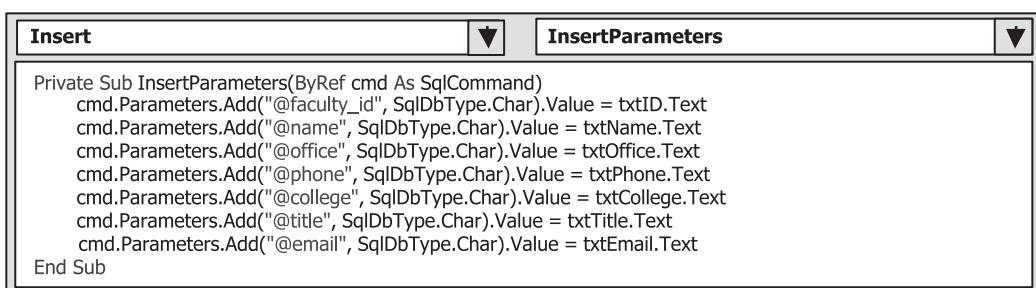
- The insert query string is declared first, and it contains seven pieces of information related to seven columns in the Faculty table in the database.
- The data components and local variables used in the procedure are declared here, too. The `FacultyImage` string variable is used to hold the faculty photo name, and the local integer variable `intInsert` is used to hold the returned data from the execution of the data insertion command.
- The faculty photo name is reserved to the `FacultyImage` string variable. If this variable contains an empty string, which means that the user did not enter a photo name in the `txtPhoto` box, a default faculty image will be assigned and displayed as the project runs. Otherwise, the selected faculty member's photo will be displayed based on the photo name entered by the user.
- The name of the faculty photo file, which is entered by the user, is reserved to the `FacultyImage` string variable. Also, this faculty photo file is stored into the `Application` state as a global variable since we need to use it later as we perform the data validation in the Faculty page. The reason we use the `Application` state to store this global variable is that each time the client sends a

request to the server, the server posts back a refreshed page to the client after the server receives that request. The values of all global variables created in the client side are lost when this refreshed page is sent back, so in order to keep those values, we must use this Application state to reserve them.

- E. The faculty name entered by the user is reserved to the FacultyName string variable. Also, this faculty name is stored into the Application state as a global variable since we need to use it later as we perform the data validation in the Faculty page. The reason we use the Application state to store this global variable is the same as the reason we discussed in step **D** for the FacultyImage.
- F. 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.
- G. The user-defined subroutine InsertParameters() is executed to assign all seven input parameters to the Parameters collection of the command object.
- H. The ExecuteNonQuery() method of the command object is called to run the insert query to perform this data insertion.
- I. A cleaning job is performed to release all objects used in the procedure.
- J. The ExecuteNonQuery() method will return a data to indicate whether this data insertion is successful or not. The value of this returned data equals the number of rows that have been successfully inserted into the Faculty table in the database. If a zero is returned, which means that no row has been inserted into the database, a warning message is displayed to indicate this situation, and the procedure is exited. Otherwise, the data insertion is successful.
- K. The Insert button is disabled after the current record is inserted into the database. This is to avoid multiple insertions of the same record into the database. The Insert button will be enabled again when the content of the Faculty ID text box is changed, which means that a new, different faculty record will be inserted.

The detailed code for the user-defined subroutine InsertParameters() is shown in Figure 7.43.

This code is straightforward and easy to understand. Each piece of new faculty information is assigned to the associated input parameter by using the Add() method of the Parameters collection of the command object.



The screenshot shows a Microsoft Visual Studio code editor window. The title bar says "Insert" on the left and "InsertParameters" on the right. The code itself is a Visual Basic subroutine:

```

Private Sub InsertParameters(ByRef cmd As SqlCommand)
    cmd.Parameters.Add("@faculty_id", SqlDbType.Char).Value = txtID.Text
    cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text
    cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text
    cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text
    cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text
    cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text
    cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text
End Sub

```

Figure 7.43. The code for the subroutine InsertParameters.

The coding for other procedures includes the coding for the Insert button's click event procedure in the Faculty page. The functionality of this piece of code is to direct the project from the Faculty page to the Insert page when the user clicks this Insert button on the Faculty page.

#### 7.5.2.3 Develop the Code for Other Procedures

Open the Insert button's click event procedure in the Faculty page by double-clicking the Insert button from the Faculty Web form window, and enter the following code into this event procedure to direct to the Insert page:

---

```
Response.Redirect("Insert.aspx")
```

---

Now we can run the project to test the data insertion functionality via the Web site. But before we can start the project, make sure that a default faculty photo file named Default.jpg and a faculty photo file named Mhamed.jpg have been stored in the default folder in which our project is located since we need to use those photo files to run our project. Also, make sure that the start page is the LogIn page by setting up the start page using the Start Options menu item.

Click the Start Debugging button to run the project. Enter a suitable username and password, such as jhenry and test, in the LogIn page, and select the Faculty Information item from the Selection page to open the Faculty page. Click the Insert button to open the Insert page, and enter the following data as the information for a new faculty member:

■ Mhamed.jpg	Faculty Photo text box
■ M56789	Faculty ID text box
■ Ali Mhamed	Faculty Name text box
■ Professor	Title text box
■ MTC-353	Office text box
■ 750-378-3355	Phone text box
■ University of Main	College text box
■ amhamed@college.edu	Email text box

Your finished new faculty information page should match the one shown in Figure 7.44.

Click the Insert button to insert this new record into the database. The Insert button is immediately disabled and the associated faculty image is displayed in the PhotoBox, as shown in Figure 7.44.

Click the Back button to return to the Faculty page. Next we need to perform the data validation to confirm that our data insertion is successful.

#### 7.5.3 Validate the Data Insertion

This data validation contains two parts. The first part is to confirm that all seven text boxes in the Insert page are not empty; in other words, all required information related to a new faculty record has been entered in these text boxes. The Faculty



Figure 7.44. The running status of the Insert page.

Photo text box is optional and can be empty. The second part is to validate the newly inserted record by retrieving it and displaying it in the Faculty page.

First, we need to add the RequiredFieldValidator to all seven text boxes to validate the data for those seven pieces of faculty information.

Open the Design View of the Insert Web form, go to the Toolbox window, and click the Validation tab to expand it. Drag the RequiredFieldValidator control from the Toolbox window and place it next to the Faculty ID text box. Set the following properties for this control in the property window:

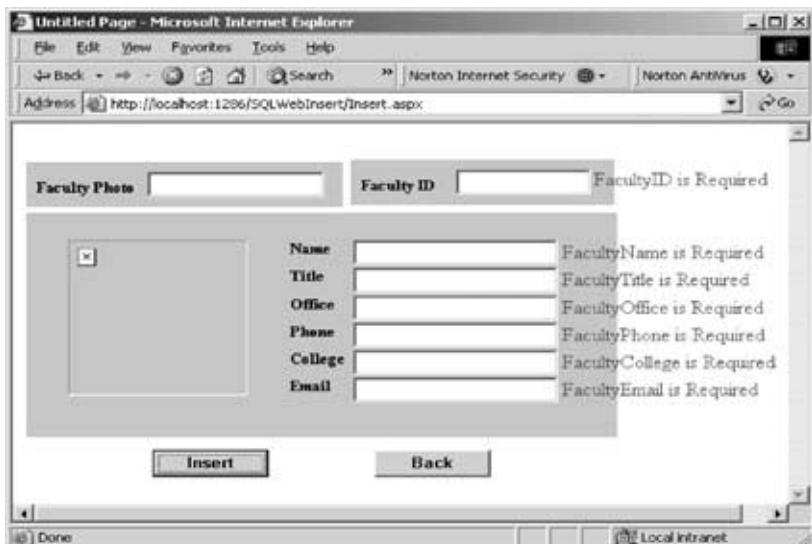
- ErrorMessage: FacultyID is required
- ControlToValidate: txtID

Perform similar dragging and placing operations to place all other six RequiredFieldValidators next to the text boxes, and set the associated properties that are shown in Table 7.7 for these controls in the property window.

One point to note is that when you drag the RequiredFieldValidator to the Insert form, make sure that you set the Style|Position property for each

**Table 7.7. Validating Controls in the Insert Page**

Control	ErrorMessage	ControlToValidate
Name TextBox	FacultyName is Required	txtName
Title TextBox	FacultyTitle is Required	txtTitle
Office TextBox	FacultyOffice is Required	txtOffice
Phone TextBox	FacultyPhone is Required	txtPhone
College TextBox	FacultyCollege is Required	txtCollege
Email TextBox	FacultyEmail is Required	txtEmail



**Figure 7.45.** The data validation in the Insert page.

RequiredFieldValidator to Absolutely position, and then you can place it in any location on the form that you like.

Your finished Insert Web form should match the one shown in Figure 7.45.

After you add these RequiredFieldValidator controls to the Insert page, a warning message will be displayed when you click the Insert button if any of the text boxes is empty as the project runs.

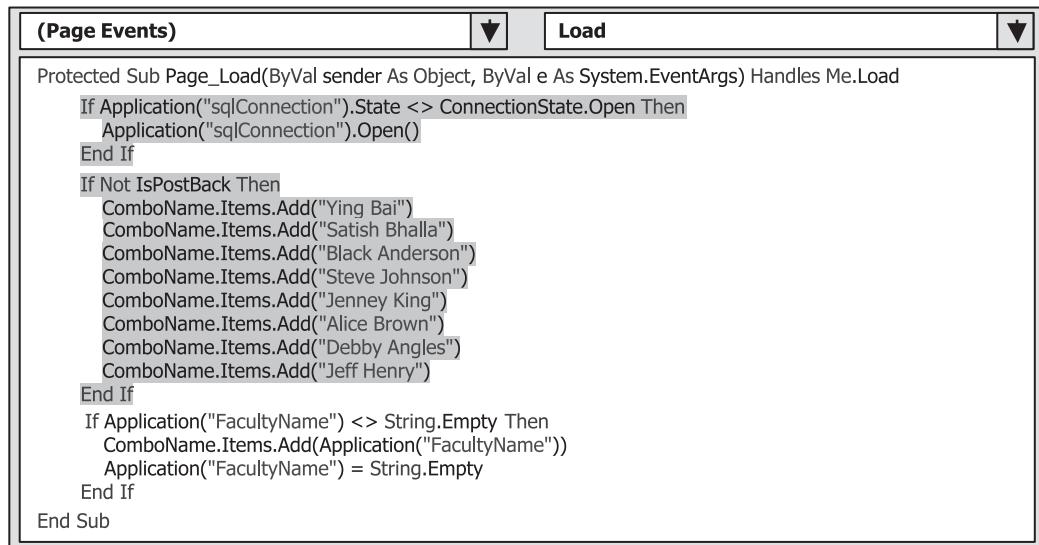
To validate the newly inserted data from the Faculty page, we need to make some modifications to the code in the Faculty page and add some code to allow us to retrieve the new inserted record from the database and display it in this page. The following procedures need to be modified:

1. Page\_Load event procedure
2. ShowFaculty() subroutine procedure

Let us first take care of the, Page\_Load event procedure.

After a new faculty record is inserted into the database from the Insert page and returned to the Faculty page, the newly inserted faculty name should be added into the ComboName combo box control to allow the user to select it to retrieve the newly inserted information for the selected faculty member. To do this, we need to add the code shown in Figure 7.46 into the Page\_Load event procedure of the Faculty page.

The code we developed in the previous section is highlighted with a gray background. The functionality of this newly added piece of code is that each time the server posts back a refreshed Faculty page to the client, we need to inspect whether a new faculty record has been inserted into the database by checking the global variable FacultyName, which is stored in the Application state. If this global variable is empty, which means that no data insertion has occurred, we do nothing.



The screenshot shows the Visual Studio IDE with the code editor open. The title bar says '(Page Events)'. Below it, there are two dropdown menus: one labeled 'Load' and another with a downward arrow. The code itself is as follows:

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    If Application("sqlConnection").State <> ConnectionState.Open Then
        Application("sqlConnection").Open()
    End If
    If Not IsPostBack Then
        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")
    End If
    If Application("FacultyName") <> String.Empty Then
        ComboName.Items.Add(Application("FacultyName"))
        Application("FacultyName") = String.Empty
    End If
End Sub

```

**Figure 7.46.** The modified code for the Page\_Load event procedure.

But if this variable contains a valid faculty name, which means that a data insertion has occurred, we need to add this newly inserted faculty name into the ComboName combo box control to allow users to select this new faculty name from the control to perform the associated data actions in the database. After this new faculty name is added into the combo box control, we need to reset this global variable to avoid multiple additions of the same faculty name into the ComboName control.

During the data insertion process, the user may want to include a faculty photo with the data insertion by entering the name of the faculty photo file into the Faculty Photo text box. But another possibility is that the user may not want to insert a faculty photo with that data insertion. In that case, the content of the Faculty Photo text box should be empty. Recall that when we developed the code for the Insert button's click event procedure in the Insert page (refer to Section 7.5.2.2), we used the global variable FacultyImage that is stored in the Application state to store the name of the newly inserted faculty photo file. Now when we validate that data insertion, we need to confirm whether the user inserted a faculty photo or not by checking that global variable FacultyImage.

Open the ShowFaculty() subroutine and add the code shown in Figure 7.47 into this procedure.

The code we developed in the previous section is highlighted with a gray background. The functionality of the newly added code is that a default faculty photo file, Default.jpg will be assigned to the FacultyImage variable if the global variable FacultyImage is empty, which means that the user does not want to add a new faculty photo and the Faculty Photo text box in the Insert page is blank. Otherwise, the faculty photo file stored in the Application state will be assigned to the FacultyImage variable that will be displayed later in the PhotoBox image control in the Faculty page.

```

Faculty ShowFaculty
Private Sub ShowFaculty(ByVal fName As String)
    Dim FacultyImage As String
    Select Case fName
        Case "Ying Bai"
            FacultyImage = "Bai.jpg"
        Case "Satish Bhalla"
            FacultyImage = "Satish.jpg"
        Case "Black Anderson"
            FacultyImage = "Anderson.jpg"
        Case "Steve Johnson"
            FacultyImage = "Johnson.jpg"
        Case "Jenney King"
            FacultyImage = "King.jpg"
        Case "Alice Brown"
            FacultyImage = "Brown.jpg"
        Case "Debby Angles"
            FacultyImage = "Angles.jpg"
        Case "Jeff Henry"
            FacultyImage = "Henry.jpg"
    End Select
    PhotoBox.ImageUrl = FacultyImage
End Sub

```

Figure 7.47. The modifications to the code of ShowFaculty subroutine.

Comparing this with the original coding we did for this part, it can be found that the warning message that would be displayed if no matched faculty image can be found from this ShowFaculty subroutine, has been removed. Right now the default faculty image will be displayed if the global variable FacultyImage is empty or no matched faculty image can be found. The reason we made this modification to this part is to make our project more professional and neat. Of course, if you want to keep that warning message to indicate the situation in which no matched faculty image can be found, you can add another global variable to monitor whether the Insert page has been executed or not. If the Insert page has not been executed and no matched faculty image can be found, the warning message should be displayed. Otherwise, the default faculty image should be displayed.

Now we have completed all modifications to the code of our Faculty page, and we can run the project to test our data insertion functionality via the Web site. Recall that we inserted a new faculty record with the faculty name Ali Mhamed into the Faculty table in the last section. In order to validate this insertion, we need to run the project and insert this new record again. To avoid a duplicate insertion, we first need to open our sample database to delete that newly inserted record from the Faculty table. Open SQL Server 2005|SQL Server Management Studio Express, and then open the Faculty table and delete that newly inserted record.

Click the Start Debugging button to run our project. Enter a suitable username and password in the LogIn page; then select the Faculty Information item from the

Selection page to open the Faculty page. Click the Insert button to open the Insert page, and enter the following data as the information for a new faculty member:

- |                       |                        |
|-----------------------|------------------------|
| ■ Mhamed.jpg          | Faculty Photo text box |
| ■ M56789              | Faculty ID text box    |
| ■ Ali Mhamed          | Faculty Name text box  |
| ■ Professor           | Title text box         |
| ■ MTC-353             | Office text box        |
| ■ 750-378-3355        | Phone text box         |
| ■ University of Main  | College text box       |
| ■ amhamed@college.edu | Email text box         |

Click the Insert button to insert this new record into the database. Then click the Back button to return to the Faculty page to perform the data validation.

Go to the ComboName combo box control and you will see that the newly inserted faculty name Ali Mhamed is already there. Click it to select this faculty member, and then click 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, as shown in Figure 7.48.

Our data insertion is successful. Click the Back button and then the Exit button to close our project. The completed Web application project SQLWebInsert is located in the folder **DBProjects\Chapter 7**, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

In the next section, we will discuss how to perform the data insertion to the Oracle database via the Web site.

## 7.6 DEVELOP AN ASP.NET WEB APPLICATION TO INSERT DATA INTO ORACLE DATABASES

Because of the coding similarity between the SQL Server and Oracle databases, we emphasize only the important differences between the code for these two databases.



Figure 7.48. The data validation process.

To save time and space, we need to modify an existing project OracleWebSelect by adding some code to this project. The code we need to add can be copied from another existing project, SQLWebInsert, with some modifications.

The main coding differences between these two database operations are as follows:

1. The code for the new Insert Web page
2. The added code to the Page\_Load event procedure of the Faculty page
3. The added code to the ShowFaculty() subroutine in the Faculty page
4. The added code to the Insert button's click event procedure in the Faculty page

We divide these added coding jobs into two sections; the first section covers the coding for the new Insert page, and the second section contains the other three steps.

Now let us begin to modify the project OracleWebSelect based on the four differences listed above to make it our new project OracleWebInsert. Open the Windows Explorer and create a new folder, such as **Chapter 7**, if you have not already created it. Copy the project OracleWebSelect from the folder **DBProjects\Chapter 7**, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and paste it in the folder **Chapter 7**. Rename the project OracleWebInsert. Also open the project SQLWebInsert we developed in the last section from the same folder since we need to copy some items and code from that project and paste them into our new project.

Open Visual Studio.NET, go to the File|Open Web Site menu item, browse to our folder **Chapter 7**, select our new project OracleWebInsert, and then click the Open button to open it. First, let us create our new Insert page by adding this page to our new project from the project SQLWebInsert.

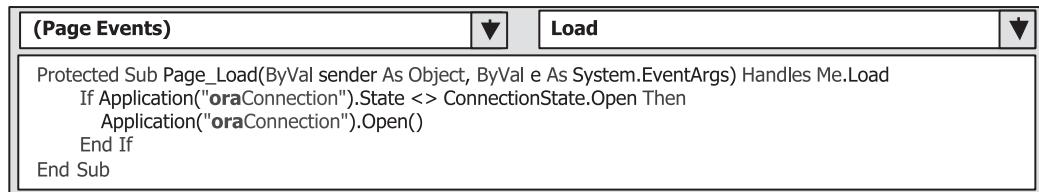
### 7.6.1 Create the Insert Web Page and Develop the Code

We can add a new Insert page to our project by adding the existing Insert page located in the project SQLWebInsert that we developed in the last section. In this way, we can save time and energy. To do that, right-click our new project icon from the Solution Explorer window, and select Add Existing Item from the popup menu. In the opened dialog box, browse to the folder **Chapter 7\SQLWebInsert** and click the Insert.aspx item, and then click the Add button to add this Insert page into our project.

Now let us modify the code of the Insert.aspx page to make it suitable for Oracle database operations. These modifications include the following:

1. Modifications to two Imports commands.
2. Modifications to the global connection object located in the Page\_Load event procedure.
3. Modifications to the code in the Insert button's click event procedure.
4. Modifications to the code in the user-defined subroutine InsertParameters().

Similarly, these modifications can be divided into two steps. Modifications 1 and 2 can be covered by step 1, and modifications 3 and 4 can be included in step 2.



The screenshot shows the 'Page Events' section of the Visual Studio code editor. The title bar says '(Page Events)'. Below it is a dropdown arrow pointing down and a 'Load' button. The code in the editor is:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    If Application("oraConnection").State <> ConnectionState.Open Then
        Application("oraConnection").Open()
    End If
End Sub
```

Figure 7.49. The modified Page\_Load event procedure.

### 7.6.1.1 Modifications to Imports Commands and Page\_Load Event Procedure

Open the code page of the Insert Web form window and replace the Imports commands with the following two Imports commands:

---

```
Imports System.Data
Imports System.Data.OracleClient
```

---

Next, open the Page\_Load event procedure and add the code shown in Figure 7.49 into this event procedure. The modified parts are highlighted in bold.

### 7.6.1.2 Modifications to the Code of Subroutines and Procedures

First, let us modify the code of the Insert button's click event procedure. Open this event procedure and perform the modifications shown in Figure 7.50 to the code in this event procedure.

Let's have a closer look at these modifications to see how they work for this event procedure.

- 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. Change the prefix for all data objects and classes from sql to ora and from Sql to Oracle, respectively.
- C. The faculty photo name is reserved to the FacultyImage string variable. If this variable contains an empty string, which means that the user did not enter any photo name in the txtPhoto box, a default faculty image will be assigned and displayed as the project runs. Otherwise, the selected faculty photo will be displayed based on the photo name entered by the user.
- D. The name of the faculty photo file, which is entered by the user, is reserved to the FacultyImage string variable. Also, this faculty photo file is stored into the Application state as a global variable since we need to use it later as we perform the data validation in the Faculty page. The reason we use the Application state to store this global variable is that each time the client sends a request to the server, the server posts back a refreshed page to the client after the server receives that request. The values of all global variables created in the client side is lost after this refreshed page is sent back, so in

```

Protected Sub cmdInsert_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim cmdString As String = "INSERT INTO Faculty(faculty_id, name, office, phone, college, title, email)" & _
        "VALUES (:faculty_id, :name, :office, :phone, :college, :title, :email)"
    Dim oraCommand As New OracleCommand
    Dim FacultyImage As String
    Dim intInsert As Integer
    FacultyImage = txtPhoto.Text 'display the new inserted faculty photo
    If FacultyImage = String.Empty Then
        FacultyImage = "Default.jpg"
    End If
    Application("FacultyImage") = FacultyImage 'reserve faculty image for validation
    PhotoBox.ImageUrl = FacultyImage
    FacultyName = txtName.Text
    Application("FacultyName") = FacultyName 'reserve faculty name for validation
    oraCommand.Connection = Application("oraConnection")
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    InsertParameters(oraCommand)
    intInsert = oraCommand.ExecuteNonQuery()
    oraCommand.Dispose()
    oraCommand = Nothing
    If intInsert = 0 Then
        Response.Write("<script>alert('The data insertion is failed')</script>")
        Exit Sub
    End If
    cmdInsert.Enabled = False 'disable the Insert button
End Sub

```

Figure 7.50. The modifications to the Insert button's click event procedure.

order to keep those values, we must use this Application state to reserve them.

- E. The same thing we discussed in step **D** happens with the global variable FacultyName.
- F. Change prefixes for all data objects used in this procedure from sql to ora.

All modified parts are highlighted in bold.

Next, let us modify the subroutine InsertParameters(). Open this subroutine and make the modifications shown in Figure 7.51 to this procedure.

- A. Change the data type of the passed command argument from SqlCommand to OracleCommand.

```

Private Sub InsertParameters(ByRef cmd As OracleCommand)
    cmd.Parameters.Add("faculty_id", OracleType.Char).Value = txtID.Text
    cmd.Parameters.Add("name", OracleType.Char).Value = txtName.Text
    cmd.Parameters.Add("office", OracleType.Char).Value = txtOffice.Text
    cmd.Parameters.Add("phone", OracleType.Char).Value = txtPhone.Text
    cmd.Parameters.Add("college", OracleType.Char).Value = txtCollege.Text
    cmd.Parameters.Add("title", OracleType.Char).Value = txtTitle.Text
    cmd.Parameters.Add("email", OracleType.Char).Value = txtEmail.Text
End Sub

```

Figure 7.51. Modifications to the code in the InsertParameters subroutine.

- B. Remove the @ symbol before each input parameter. Also change the data type of all input parameter from SqlDbType to OracleType.

All modifications are highlighted in bold.

### 7.6.2 Modify the Code for the Faculty Page

Three modifications need to be performed for this page.

1. First, we need to attach a piece of code to the end of the Page\_Load event procedure to add the newly inserted faculty name into the ComboName combo box control; in this way, it allows users to select the newly inserted faculty name from this control to validate the new record insertion.
2. Second, we need to modify and add another piece of code in the ShowFaculty() subroutine to allow the newly inserted faculty photo to be displayed as the newly inserted data is validated.
3. Add one line of code to the Insert button's click event procedure to open the Insert page when this button is clicked as the project runs.

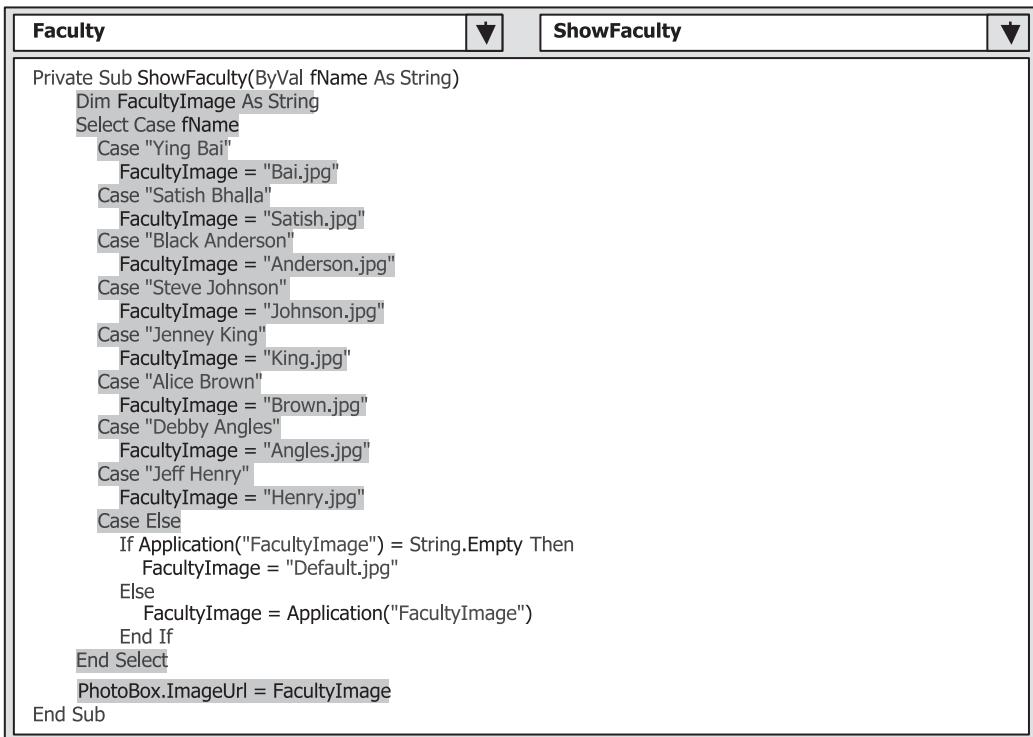
Now let us perform the first modification. Open the Page\_Load event procedure of the Faculty page. Add the code shown in Figure 7.52 into this event procedure. The code we developed in the previous section is highlighted with a gray background.

Let's take a closer look at this piece of newly added code to see how it works.

- A. Each time the server posts back a refreshed Faculty page to the client, we need to inspect whether a new faculty record has been inserted into the database by checking the global variable FacultyName, which is stored in the Application state. If this global variable is empty, which means that no data insertion has occurred, we need to do nothing. But if this variable contains a valid faculty name, which means that a data insertion has occurred, we

```
(Page Events) Load
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    If Application("oraConnection").State <> ConnectionState.Open Then
        Application("oraConnection").Open()
    End If
    If Not IsPostBack Then
        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")
    End If
    A If Application("FacultyName") <> String.Empty Then
        ComboName.Items.Add(Application("FacultyName"))
        Application("FacultyName") = String.Empty
    End If
End Sub
```

Figure 7.52. The modified Page\_Load event procedure.



The screenshot shows the Visual Studio code editor with the title bar "Faculty" and a button "ShowFaculty". The code itself is a Visual Basic subroutine:

```

Private Sub ShowFaculty(ByVal fName As String)
    Dim FacultyImage As String
    Select Case fName
        Case "Ying Bai"
            FacultyImage = "Bai.jpg"
        Case "Satish Bhalla"
            FacultyImage = "Satish.jpg"
        Case "Black Anderson"
            FacultyImage = "Anderson.jpg"
        Case "Steve Johnson"
            FacultyImage = "Johnson.jpg"
        Case "Jenney King"
            FacultyImage = "King.jpg"
        Case "Alice Brown"
            FacultyImage = "Brown.jpg"
        Case "Debby Angles"
            FacultyImage = "Angles.jpg"
        Case "Jeff Henry"
            FacultyImage = "Henry.jpg"
        Case Else
            If Application("FacultyImage") = String.Empty Then
                FacultyImage = "Default.jpg"
            Else
                FacultyImage = Application("FacultyImage")
            End If
    End Select
    PhotoBox.ImageUrl = FacultyImage
End Sub

```

Figure 7.53. The modified subroutine ShowFaculty.

need to add this newly inserted faculty name into the ComboName combo box control to allow users to select this new faculty name from that control to perform the data validation. Finally, we need to reset this global variable to avoid multiple additions of the same faculty name into the ComboName control.

Next, open the ShowFaculty() subroutine and perform the modifications shown in Figure 7.53 to this subroutine.

The code we developed in the previous section is highlighted with a gray background. The functionality of the newly added code is that a default faculty photo file, Default.jpg, will be assigned to the FacultyImage variable if the global variable FacultyImage is empty, which means that the user does not want to add a new faculty photo with that data insertion and the Faculty Photo text box in the Insert page is blank. Otherwise the faculty photo file stored in the Application state will be assigned to the FacultyImage variable that will be displayed later in the PhotoBox image control in the Faculty page.

Finally, open the Insert button's Click event procedure and enter the following code into this event procedure:

---

```
Response.Redirect("Insert.aspx")
```

---

This code will direct the Web application from the current page to the Insert page.

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 three jobs have been done:

1. Make sure that the default faculty photo file Default.jpg has been saved to the default folder in which our Web application project is located. In our application, it is **C:\Chapter 7\OracleWebInsert**.
2. Make sure that the startup page is LogIn. To confirm this, right-click the project icon from the Solution Explorer window, and select the Start Options item from the popup menu. In the opened dialog box, make sure that the Specific page radio button is selected and the page LogIn.aspx is in that box. Click the OK button to close this dialog box.
3. Make sure that a faculty member named Ali Mhamed is not in the Faculty table in our sample database because we will use this faculty name as an example to insert it into our sample database. To confirm that, open the Faculty table from our sample database; delete this record if it is in there. The reason to do this is that the database does not allow us to insert the same record more than one time, so we must delete that record before we can insert the same data into the database. If you want to insert faculty data other than that for Ali Mhamed, you do not need to take this step.

Now click the Start Debugging button to run the project. Enter a suitable user-name and password in the LogIn page, and select Faculty Information from the Selection page to open the Faculty page. Click the Insert button to open the Insert page, and enter the following data as the information for a new faculty member:

■ Mhamed.jpg	Faculty Photo text box
■ M56789	Faculty ID text box
■ Ali Mhamed	Faculty Name text box
■ Professor	Title text box
■ MTC-353	Office text box
■ 750-378-3355	Phone text box
■ University of Main	College text box
■ amhamed@college.edu	Email text box

Click the Insert button to insert this new record into the database. Then click the Back button to return to the Faculty page to perform the data validation.

Go to the ComboName combo box control, and you will see that the newly inserted faculty name Ali Mhamed is already there. Click it to select this faculty member, and then click 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, as shown in Figure 7.54.

Our data insertion to the Oracle database is successful. Click the Back button and then Exit button to close our project. The completed Web application project named OracleWebInsert can be found in the folder **DBProjects\Chapter 7**, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).



Figure 7.54. The data validation process.

In the next section, we will discuss how to perform data updation and deletion in the SQL Server database via the Web site.

## 7.7 DEVELOP WEB APPLICATIONS TO UPDATE AND DELETE DATA IN SQL SERVER DATABASES

To update or delete data in relational databases is a challenging topic. We provided a very detailed discussion and analysis of this topic in Section 6.1.1. Refer to that section to get a more detailed discussion of these data actions. Here we want to emphasize some important points related to the data updating and deleting.

1. When updating or deleting data in related tables in a DataSet, it is important to update or delete data in the proper sequence in order to reduce the chance of violating referential integrity constraints. The order of command execution will also follow the indices of the DataRowCollection in the DataSet. To prevent data integrity errors from being raised, the best practice is to update or delete data in the following sequence:
  - A. Child table: delete records.
  - B. Parent table: insert, update, and delete records.
  - C. Child table: insert and update records.
2. To update existing data in the database, generally it is unnecessary to update the primary key for that record. It is much better to insert a new record with a new primary key into the database than to update the primary key for an existing record because of the complicated table operations listed above. In practice, it is very rare to update a primary key for an existing record in the database in real applications. So in this section, we concentrate on updating an 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 the records related to the data to be deleted in the Faculty table from the child tables such as the LogIn and Course tables, and then delete the record from the Faculty table. The reason for this deleting sequence is that the faculty\_id is a foreign key in the LogIn and 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.

Keeping these three points in mind, now let us begin our project.

We need to modify our existing project SQLWebInsert and make it into our new project, SQLWebUpdateDelete. To do that, open the Windows Explorer and create a new folder, **Chapter 7**, if you have not already done so. Then copy the project SQLWebInsert from the folder **DBProjects\Chapter 7**, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and paste it into our new folder **Chapter 7**. Rename this project SQLWebUpdateDelete.

### 7.7.1 Application User Interfaces

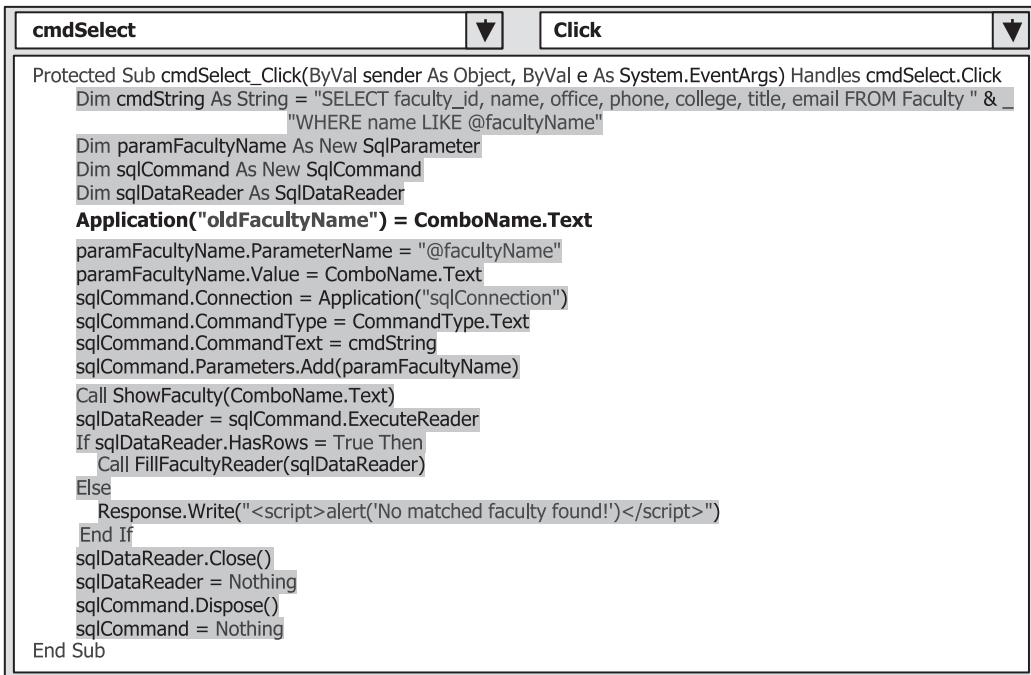
To update or delete an existing record in our sample database, we do not need a new Web page as our user interface; we can use the Faculty page as our user interface to perform those data actions. To meet our data actions' requirements, we need to perform some modifications to the Faculty page.

The first modification to the Faculty Web form is to clean up the Faculty ID text box during the data updating process because we do not want users to modify this piece of information based on our discussion in item 2 in the last section.

### 7.7.2 Modify the Code for the Faculty Page

Besides the coding development for the Update button's click event procedure, which we will discuss in the next section, the only modification to this page is to add one statement into the Select button's click event procedure, which is shown in Figure 7.55. The newly added statement is highlighted in bold, and all code developed in the previous section is indicated with a gray background.

The purpose of this statement is to store the current selected faculty name that is located in the ComboName 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 happens, the updated faculty name that is stored in the txtName text box will be added into the ComboName 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 store it since this is a Web application and each time the server posts back a refreshed Faculty page based on the client's request, the contents of all controls on that page will be refreshed and the old contents will be lost.



The screenshot shows the Visual Studio IDE with the code editor open. The title bar says "cmdSelect" and the tab bar says "Click". The code is written in VB.NET and handles the click event for a button named cmdSelect. It uses Application("oldFacultyName") to store the previous faculty name and a parameter @facultyName to search for matching names in the Faculty table.

```

Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString As String = "SELECT faculty_id, name, office, phone, college, title, email FROM Faculty " & _
        "WHERE name LIKE @facultyName"
    Dim paramFacultyName As New SqlParameter
    Dim sqlCommand As New SqlCommand
    Dim sqlDataReader As SqlDataReader
    Application("oldFacultyName") = ComboName.Text
    paramFacultyName.ParameterName = "@facultyName"
    paramFacultyName.Value = ComboName.Text
    sqlCommand.Connection = Application("sqlConnection")
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add(paramFacultyName)
    Call ShowFaculty(ComboName.Text)
    sqlDataReader = sqlCommand.ExecuteReader
    If sqlDataReader.HasRows = True Then
        Call FillFacultyReader(sqlDataReader)
    Else
        Response.Write("<script>alert('No matched faculty found!')</script>")
    End If
    sqlDataReader.Close()
    sqlDataReader = Nothing
    sqlCommand.Dispose()
    sqlCommand = Nothing
End Sub

```

Figure 7.55. The modified Select button's event procedure.

Now let's develop the code for the Update button's click event procedure.

### 7.7.3 Develop the Code for the Update Button Event Procedure

Open this event procedure by double-clicking the Update button from the Faculty Web form window, and enter the code shown in Figure 7.56 into this event procedure.

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

- An updating query string is declared first, with oldName as the name of the dynamic parameter. This is because when you want to update the faculty name, the original name stored in the ComboName combo box control becomes the old name, and we need to distinguish this old name from the updated name.
- All data objects used in this procedure are created here, and a local integer variable, intUpdate, is also created and is used as a holder to keep the data returned from executing the ExecuteNonQuery() method.
- Before we can perform the data updating, we first need to clean up the Faculty ID text box since we do not want to update this piece of information.
- 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 faculty name that is stored in the text box txtName. If the names are different, the user has

	cmdUpdate	▼	Click	▼
A	Protected Sub cmdUpdate_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click			
	Dim cmdString As String = "UPDATE Faculty SET name = @name, office = @office, phone = @phone, " &			
	"college = @college, title = @title, email = @email WHERE (name LIKE @oldName)"			
B	Dim sqlCommand As New SqlCommand			
	Dim intUpdate As Integer			
C	txtID.Text = String.Empty	'clean the faculty_id textbox		
D	If txtName.Text <> Application("oldFacultyName") Then			
	ComboName.Items.Add(txtName.Text)			
	ComboName.Items.Remove(Application("oldFacultyName"))			
E	End If			
F	sqlCommand.Connection = Application("sqlConnection")			
G	sqlCommand.CommandType = CommandType.Text			
H	sqlCommand.CommandText = cmdString			
	UpdateParameters(sqlCommand)			
I	intUpdate = sqlCommand.ExecuteNonQuery()			
	sqlCommand.Dispose()			
	sqlCommand = Nothing			
	If intUpdate = 0 Then			
	Response.Write("<script>alert('The data updating is failed')</script>")			
	Exit Sub			
	End If			
	End Sub			

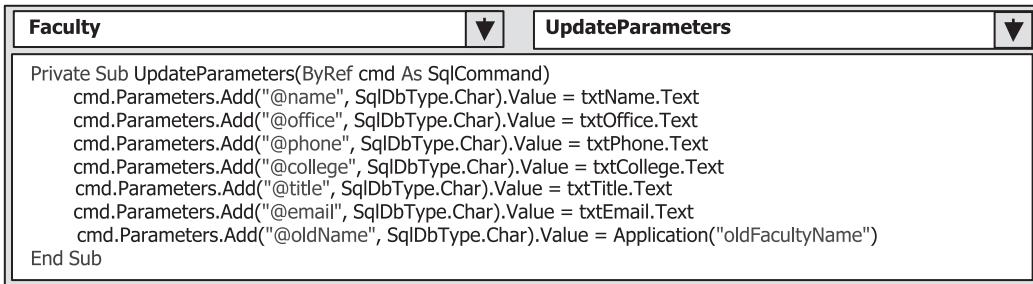
Figure 7.56. The code for the Update button's click event procedure.

updated the faculty name. In that case, we need to add the updated faculty name into the ComboName combo box control and remove the old faculty name from that control to allow users to select this updated faculty name from the combo box list to perform data actions in the database in the future.

- E. The Command object is initialized with the connection object, command type, and command text.
- F. The user-defined subroutine `UpdateParameters()`, whose detailed code is shown below, is called to assign all input parameters to the Command object.
- G. The `ExecuteNonQuery()` method of the Command class is called to execute the data updating operation. This method returns feedback data to indicate whether this data updating is successful or not, and this returned data is stored to the local integer variable `intUpdate`.
- H. A cleaning job is performed to release all data objects used in this procedure.
- I. The data value returned from calling the `ExecuteNonQuery()` is exactly equal to the number of rows that have been successfully updated in the database. If this value is zero, which means that no row has been updated and this data updating is failed, a warning message is displayed and the procedure is exited. Otherwise, if this value is nonzero, this data updating is successful.

The detailed code for the subroutine `UpdateParameters()` is shown in Figure 7.57.

Seven input parameters are assigned to the `Parameters` collection property of the Command object using the `Add()` method. One important point for this parameter's assignment is the last input parameter or the dynamic parameter `oldName`. The original or the old faculty name `oldFacultyName` stored in the Application state must be used as the value for this parameter. Some readers may argue that the original or old



```

Private Sub UpdateParameters(ByRef cmd As SqlCommand)
    cmd.Parameters.Add("@name", SqlDbType.Char).Value = txtName.Text
    cmd.Parameters.Add("@office", SqlDbType.Char).Value = txtOffice.Text
    cmd.Parameters.Add("@phone", SqlDbType.Char).Value = txtPhone.Text
    cmd.Parameters.Add("@college", SqlDbType.Char).Value = txtCollege.Text
    cmd.Parameters.Add("@title", SqlDbType.Char).Value = txtTitle.Text
    cmd.Parameters.Add("@email", SqlDbType.Char).Value = txtEmail.Text
    cmd.Parameters.Add("@oldName", SqlDbType.Char).Value = Application("oldFacultyName")
End Sub

```

Figure 7.57. The code for the subroutine UpdateParameters.

faculty name is located in the ComboName combo box control, and we can directly get it from that control without using this global variable. Well, this statement would be correct for a Windows-based application, but for the Web-based application, it is absolutely wrong. Recall that when the user clicks the Update button to perform a data updating action, this updating request will be sent to the server and the server will post back a refreshed Faculty page to the client. All old or original data stored in text boxes or combo boxes in the previous page will be lost. In other words, the contents of all text boxes and combo boxes in this refreshed page are different from the contents stored in the previous pages. A wrong updating may occur if you still use the faculty name stored in the combo box control ComboName in the current or refreshed page.

At this point, we have finished all coding jobs for the data updating actions in the SQL Server database using the Faculty page. Before we can run the project to test this data updating functionality, make sure that the starting page is the LogIn page and the default faculty image file Default.jpg has been stored in our default folder. To check the starting page, right-click the project icon from the Solution Explorer window, select the Start Options item from the popup menu, and then check the Specific page radio button and select LogIn.aspx as the starting page.

Now let us run the project to test the data updating actions. Click the Start Debugging button to run the project, enter a suitable username and password to the LogIn page, and select the Faculty Information item from the Selection page to open the Faculty page. Keep the default faculty name Ying Bai selected in the ComboName combo box control, and click the Select button to retrieve the information for this selected faculty member from the database and display it in this page.

Now let us test the data updating actions in two steps: first, we update the faculty information without touching the faculty name, and second, we update the faculty information including changing the faculty name.

Let us start with the first step now. Enter the following information into the associated text boxes to update this faculty member's information:

- |                       |                 |
|-----------------------|-----------------|
| ■ Associate Professor | Title text box  |
| ■ MTC-353             | Office text box |
| ■ 750-378-3300        | Phone text box  |

Click the Update button to perform this data updating. To confirm this data updating, first select another faculty name from the ComboName combo box



Figure 7.58. The data updating process.

control and click the Select button to retrieve and display that faculty information. Then select the faculty member Ying Bai, whose information has been just updated, from the combo box control, and click the Select button to retrieve and display it. You can see that the selected faculty member's information has been updated, as shown in Figure 7.58.

Next, let us perform the data updating in the second step: updating the faculty name.

Still keeping the current page unchanged, change the faculty information by entering the following data in the associated text boxes:

- Jones Bai Faculty Name text box
- Professor Title text box
- MTC-555 Office text box
- 750-378-3355 Phone text box
- jbai@college.edu Email text box

Click the Update button to update this faculty information. Immediately you will find that the original faculty name Ying Bai disappears from the ComboName combo box control. To confirm this data updating, in a similar way first select another faculty name from the ComboName combo box control and click the Select button to retrieve and display that faculty information. Then select the faculty member Jones Bai whose information has been just updated, from the combo box control, and click the Select button to retrieve and display it. You can see that the selected faculty information including the faculty name has been updated, as shown in Figure 7.59.

One point to note is the faculty photo. When you update the faculty name, you did not place an updated faculty photo file in the default folder, so a default faculty



Figure 7.59. The data updating process – including the name updating.

photo is displayed for this situation. You can change this situation by placing an updated faculty photo file in the default folder before the project runs if you want the correct faculty photo to be displayed with this data updating.

Our data updating action is very successful.

Next, let us take care of data deletion in the SQL Server database.

Similarly to the data updating, for the data deletion we do not need new Web page as our user interface and can still use the Faculty page to perform the data deletion.

#### **7.7.4 Develop the Code for the Delete Button Event Procedure**

Since deleting a record from a relational database is a complex issue, we divide this discussion into five sections:

1. Relationships between the five tables in our sample database
2. The data deletion sequence
3. Using the Cascade deleting option to simplify the data deletion
4. Creating a stored procedure to perform the data deletion
5. Calling a stored procedure to perform the data deletion

##### **7.7.4.1 Relationships Between Five Tables in Our Sample Database**

As we discussed at the beginning of this section, to delete a record from a relational database, one must follow the correct sequence. In other words, one must first delete the records 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 by the primary and foreign keys. In order to make these relationships clear, we repeat as Figure 2.5 Figure 7.60 to illustrate this issue.

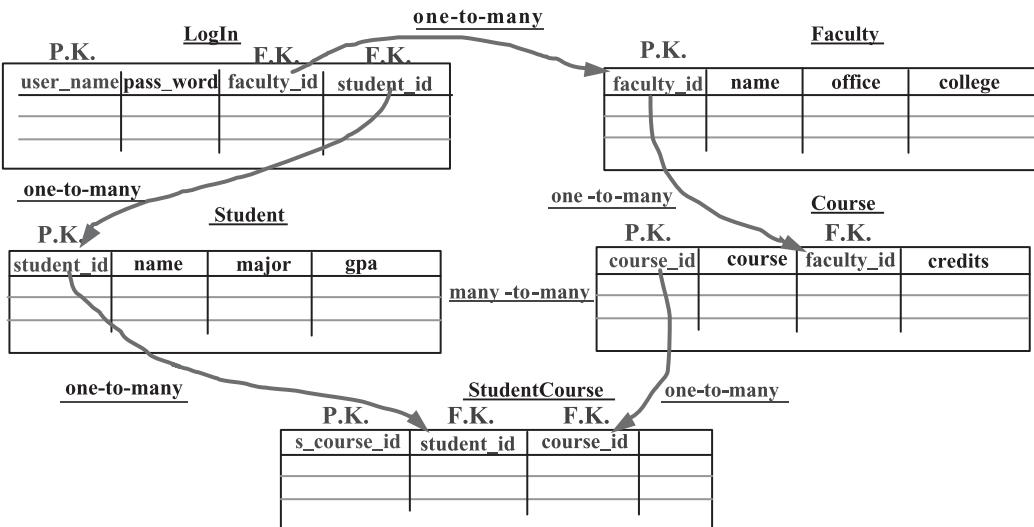


Figure 7.60. The relationships between the five tables.

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 is the relationships that 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 Course tables. Therefore, the Faculty table is a parent table, and the LogIn and Course tables are child tables. Before one can delete record from the Faculty table, one must first delete records that have the faculty.id as the foreign key from the child tables. In other words, one must first delete those records that use the faculty.id as a foreign key from the LogIn and the Course tables.
- When deleting records that use the faculty.id as a foreign key from the Course table, the related course.id that is a primary key in the Course table will also be deleted. The Course table right now is a parent table since the course.id is a primary key for this table. But as we mentioned, to delete any record from a parent table, one must first delete the related records from the child tables. Now the StudentCourse table is a child table for the Course table, 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 are deleted, finally the faculty member can be deleted from the parent table, the Faculty table.

#### 7.7.4.2 Data Deletion Sequence

Similarly, 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 Student-Course table.
2. Delete all records that use the faculty\_id as the foreign key from the LogIn table.
3. Delete all records that use the faculty\_id as the foreign key from the Course table.
4. Delete the desired faculty member from the Faculty table.

You can see how complicated the operation is to delete one record from the relational database in this example.

#### **7.7.4.3 Use the Cascade Deleting Option to Simplify the Data Deletion**

To simplify the data deleting operations, we can use the cascade deleting option provided by the SQL Server 2005 Database Management Studio.

Recall that when we created and built the relationships between our five tables, the following five relationships were built between tables:

1. A relationship between the LogIn and Faculty tables was set up using the faculty\_id as a foreign key, FK\_LogIn\_Faculty, in the LogIn table.
2. A relationship between the LogIn and Student tables was set up using the student\_id as a foreign key, FK\_LogIn\_Student, in the LogIn table.
3. A relationship between the Course and Faculty tables was set up using the faculty\_id as a foreign key, FK\_Course\_Faculty, in the Course table.
4. A relationship between the StudentCourse and Course tables was set up using the course\_id as a foreign key, FK\_StudentCourse\_Course, in the StudentCourse table.
5. A relationship between the StudentCourse and Student tables was set up using the student\_id as a foreign key, FK\_StudentCourse\_Student, in the StudentCourse table.

Refer to the data deleting sequence listed in Section 7.7.4.2; to delete a record from the Faculty table, one needs to perform four deleting operations in that sequence. Comparing those four deleting operations, the first one is the most difficult, and the reason for that is the following.

To perform the first data deletion, 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. The deleting operations in steps 3 and 4 are very easy and each deleting operation needs only one deleting query. So how can we find an easy way to complete the deleting operation in sequence 1?

A good solution is to use the Cascade option in the data deleting and updating setup dialog box provided by the SQL Server 2005 Database Management Studio. The Cascade option allows the SQL Server 2005 database engine to perform the deleting operation in sequence 1 as long as a Cascade option is selected for relationships 4 and 5 listed above.

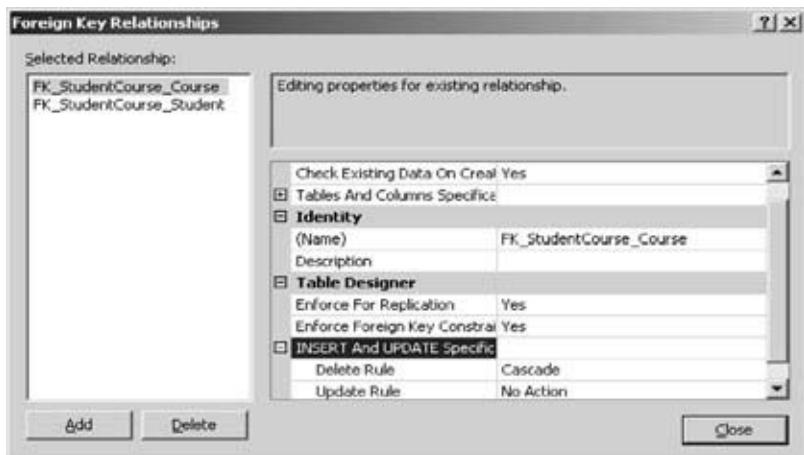


Figure 7.61. The Foreign Key Relationship dialog box.

Now let us use a real example to illustrate how to use this Cascade option to simplify the data deleting operations, especially for the first data deletion in that sequence.

Open SQL Server Management Studio Express by going to Start>All Programs|Microsoft SQL Server 2005|SQL Server Management Studio Express. In the opened Studio Express window, click Database and expand our sample database CSE\_DEPT, and then expand that database to display all five tables. Since we are interested only in relationships 4 and 5, expand the dbo.StudentCourse table and expand the Keys folder to display all keys we set up before. Double-click the FK\_StudentCourse.Course key to open it, as shown in Figure 7.61.

In the opened dialog box, keep our desired foreign key FK\_StudentCourse.Course selected from the left pane, and then click the small plus icon before the item INSERT And UPDATE Specification and select Cascade for the Delete Rule item. Your finished Cascade option setup dialog box should match the one shown in Figure 7.61.

Perform the same operation for the foreign key FK\_StudentCourse.Student in this dialog box.

After this Cascade option is set up, each time you want to delete all records that use the course\_id or the student\_id as the foreign keys in the StudentCourse table, the SQL Server engine will perform those data deleting operations automatically for you. So now you can see how easy it is to perform the data deletion in step 1.

After the first data deleting operation listed in the deleting sequence in Section 7.7.4.2, we can perform 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 start to create our stored procedure. Is that is it possible for us to set up Cascade options for relationships 1, 2, and 3 listed above to

```

ALTER PROCEDURE dbo.DeleteFacultySP
(
    @FacultyName VARCHAR(30)
)
AS
DECLARE @FacultyID VARCHAR(10)
SET @FacultyID = (SELECT faculty_id FROM Faculty
WHERE name LIKE @FacultyName)
DELETE FROM LogIn WHERE faculty_id LIKE @FacultyID
DELETE FROM Course WHERE faculty_id LIKE @FacultyID
DELETE FROM Faculty WHERE faculty_id LIKE @FacultyID
RETURN

```

Figure 7.62. The stored procedure dbo.DeleteFaculty.

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 will leave this as homework and allow students to handle this issue themselves.

Now let's create our stored procedure for this data deleting operation.

#### 7.7.4.4 Create the Stored Procedure to Perform the Data Deletion

This stored procedure contains three deletion queries that can be mapped to three steps listed in Section 7.7.4.2, that is, 2, 3, and 4.

Open Visual Studio.NET 2005, open the Sever Explorer window, expand our database CSE\_DEPT.mdf, right-click the Stored Procedures folder, select the Add New Stored Procedure item from the popup menu, and enter the code shown in Figure 7.62 into this new stored procedure.

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

- The stored procedure's name is dbo.DeleteFacultySP, and the prefix dbo is required by the SQL Server database to create any stored procedure.
- 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.
- A local variable @FacultyID is declared, and it is used to hold the returned value from the execution of the data query to the Faculty table in the step D.
- A data query is executed to pick up the matched faculty\_id from the Faculty table based on the input parameter @FacultyName.
- After the faculty\_id is obtained from the data query, three deleting queries are executed in the order shown in Figure 7.62 to perform three deleting operations. The order is as follows: first one must delete all records that use the faculty\_id as the foreign key from the child tables, such as the LogIn and



Figure 7.63. The Run Stored Procedure dialog box.

Courses tables, and 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 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.

Right-click our new stored procedure dbo.DeleteFacultySP from the Server Explorer window, and click the Execute item from the popup menu to open the Run Stored Procedure dialog box. Enter the input parameter Ying Bai, that is, the faculty member to be deleted from the Faculty table, into the Value box, and your finished parameters dialog box is shown in Figure 7.63.

Click the OK button to run this stored procedure. The running result is displayed in the Output window at the bottom, as shown in Figure 7.64.

One point to note is the number of rows that are affected in Figure 7.64. It indicates that seven rows are affected or deleted from our sample database, but this number is not the total number of rows that have been deleted from our database. According to the records built in our sample database, a total of there should be eleven rows deleted from our database, as shown in Table 7.8.

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

```
Output

Running [dbo].[DeleteFacultySP] ( @FacultyName = Ying Bai ).

(7 row(s) affected)
(0 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[DeleteFacultySP].
```

Figure 7.64. The running result of the stored procedure.

**Table 7.8.** The Total Number of Rows Affected or Deleted

Table	Rows Affected	Number of Rows Affected
LogIn	username = 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 = 10014 (course_id = CSE-438) s_course_id = 10016 (course_id = CSC-132B) s_course_id = 10017 (course_id = CSC-234A)	5
Faculty	Faculty_id = B78880	1

engine are not included in 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:

- LogIn table
- Faculty table
- Course table
- StudentCourse table

It can be found that all records listed in the Rows Affected column in Table 7.8 have been deleted from the associated tables. Our data deletion using the stored procedure is successful.

Next, we need to develop the code in the ASP.NET environment to call this stored procedure to delete a selected faculty record from the database via our Web application project. But before we can develop our code, it is highly recommended that you recover all records that have been deleted from our sample database.

To do that recovering job, you need to close Visual Studio.NET and open SQL Server Management Studio Express, and take the following actions in the following order:

1. Recover the Faculty table by adding the deleted faculty record shown in Table 7.9 into the Faculty table.

**Table 7.9.** The Data to Be Added to the Faculty Table

faculty_id	name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Assistant Professor	ybai@college.edu

**Table 7.10.** The Data to Be Added to the Login Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	

2. Recover the LogIn table by adding the deleted login record shown in Table 7.10 into the LogIn table.
3. Recover the Course table by adding the deleted courses taught by the deleted faculty member, shown in Table 7.11, into the Course table.
4. Recover the StudentCourse table by adding the deleted courses taken by the associated students, shown in Table 7.12, into the StudentCourse table.

Save these changes, and now we can close the SQL Server Management Studio and open Visual Studio.NET to develop our code to call the stored procedure to perform the data deleting actions in the SQL Server database.

#### **7.7.4.5 Develop the Code to Call the Stored Procedure to Perform the Data Deletion**

In 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 from the Faculty Web form window by double-clicking the Delete button. Enter the code shown in Figure 7.65 into this event procedure.

Let's take a closer look at this piece of code 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 SQL Server Management Studio. 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 query string must be exactly identical to the name of the stored procedure we developed in the SQL Server Management Studio, otherwise an error would be encountered as the project runs since the page identifies the stored procedure based on its name.

**Table 7.11.** The Data to Be Added to 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

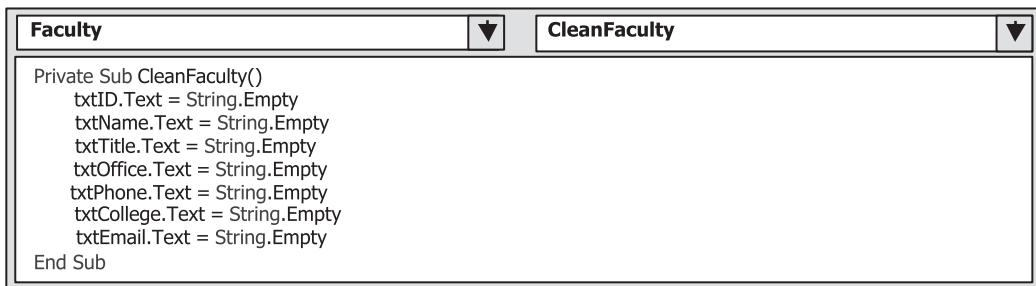
**Table 7.12.** The Data to Be Added to the StudentCourse Table

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

- B. The data object and local variable used in this procedure are declared here. The integer variable intDelete is used to hold the value returned from calling the ExecuteNonQuery() method of the Command class later.
- C. The Command object is initialized by assigning the connection object that 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 from 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 ComboName combo box control in the Faculty page. Similarly, the name of this input parameter must be identical to the name of the input parameter of the stored procedure we defined 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 it is assigned to the local variable intDelete.

	cmdDelete	▼
	Click	▼
<b>A</b> <b>B</b> <b>C</b> <b>D</b> <b>E</b> <b>F</b> <b>G</b> <b>H</b> <b>I</b>	<pre> Protected Sub cmdDelete_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdDelete.Click     Dim cmdString As String = "dbo.DeleteFacultySP"     Dim sqlCommand As New SqlCommand     Dim intDelete As Integer      sqlCommand.Connection = Application("sqlConnection")     sqlCommand.CommandType = CommandType.StoredProcedure     sqlCommand.CommandText = cmdString     sqlCommand.Parameters.Add("@FacultyName", SqlDbType.Char).Value = ComboName.Text     intDelete = sqlCommand.ExecuteNonQuery()     sqlCommand.Dispose()     sqlCommand = Nothing     If intDelete = 0 Then         Response.Write("&lt;script&gt;alert('The data Deleting is failed')&lt;/script&gt;")         Exit Sub     End If     CleanFaculty() End Sub </pre>	

Figure 7.65. The code for the Delete button's click event procedure.



The screenshot shows a Microsoft Visual Studio code editor window. The title bar has tabs for 'Faculty' and 'CleanFaculty'. The 'CleanFaculty' tab is active. The code in the editor is:

```

Private Sub CleanFaculty()
    txtID.Text = String.Empty
    txtName.Text = String.Empty
    txtTitle.Text = String.Empty
    txtOffice.Text = String.Empty
    txtPhone.Text = String.Empty
    txtCollege.Text = String.Empty
    txtEmail.Text = String.Empty
End Sub

```

Figure 7.66. The code for the subroutine CleanFaculty.

- G. A cleaning job is performed to release all objects used in this procedure.
- H. The value returned 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 from or affected in our database and this data deletion has failed, a warning message is displayed and the procedure is exited. Otherwise, if a nonzero value is returned, at least one row in our database has been deleted (all rows should also be deleted) from our database and this data deletion is successful.
- I. A user-defined subroutine CleanFaculty(), whose detailed code is shown below, is executed to clean up the contents of all text boxes that stored the deleted faculty information.

The code for the subroutine CleanFaculty() is shown in Figure 7.66.

This piece of code is easy to understand. All text boxes are cleaned up by assigning an empty string to their Text property.

At this point, we finished all coding jobs to delete data in the SQL Server database using a stored procedure. Before we can run the project to test this deleting functionality, make sure that the starting page is the LogIn page. As the project runs, enter a suitable username and password to complete the login process, open the Faculty page by clicking the Faculty Information item from the Selection page, keep the default faculty name, Ying Bai, selected from the combo box control, and then click the Select button to retrieve and display this faculty member's information.

Click the Delete button to run the stored procedure dbo.DeleteFacultySP to delete this faculty record from our database. Immediately all information stored in the seven text boxes is deleted.

To confirm this data deletion, open our sample database and you will find that all records related to that faculty member, as shown in Tables 7.9–7.12, have been deleted from our database. Yes, our data deletion is successful.

Before you close the SQL Server Management Studio, we highly recommend that you restore all deleted records to the associated tables. Refer to Tables 7.9–7.12 to add those records back to the associated tables.

The completed Web application project SQLWebUpdateDelete is located in the folder **DBProjects\Chapter 7** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

## 7.8 DEVELOP AN ASP.NET WEB APPLICATION TO UPDATE AND DELETE DATA IN ORACLE DATABASES

Because of the coding similarity between SQL Server and Oracle databases, we emphasize only the important differences between the coding for these two databases. To save time and space, we will modify the existing Web application project OracleWebInsert that we developed in the previous section to make it into our new project, OracleWebUpdateDelete. To do that, open the Windows Explorer and create a new folder, such as **Chapter 7**, if you have not already created it. Copy the project OracleWebInsert from the folder **DBProjects\Chapter 7**, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) and paste it in the folder **Chapter 7**. Rename the project OracleWebUpdateDelete.

We divide this section into two parts in terms of the coding functionalities:

1. The first part is to modify the new project to perform the data updating actions in the Oracle database.
2. The second part is to develop stored procedures to perform the data deleting actions in the Oracle database.

Now let us start from the first part – modify the new project to make it perform the data updating in the Oracle database.

### 7.8.1 Modify the Project to Perform the Data Updating

Open Visual Studio.NET, go to the File|Open Web Site menu item, browse to our folder **Chapter 7**, select our new project OracleWebUpdateDelete, and then click the Open button to open it.

The modifications to this page can be divided into the following two parts:

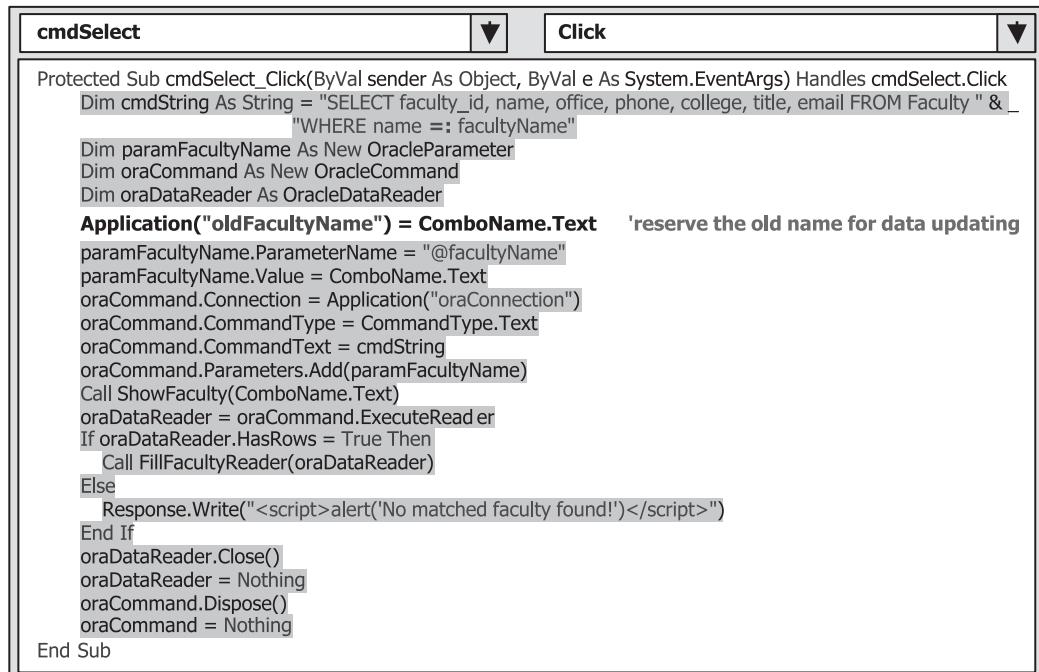
1. Modify the Select button's click event procedure by adding one statement to reserve the original or old faculty name stored in the ComboName combo box control for the possible faculty name updating operation later.
2. Add the code for the Update button's click event procedure and the user-defined subroutine `UpdateParameters()`.

Let's begin with the first modification.

#### 7.8.1.1 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 event procedure should match the one shown in Figure 7.67. The newly added statement is highlighted in bold, and the code developed in the previous section is highlighted with a gray background.

The purpose of this statement is to store the current selected faculty name that is located in the ComboName 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 happens, the updated faculty name that is stored in the txtName text box will be added into the ComboName combo box control and the original



The screenshot shows the Visual Studio IDE with the code editor open. The title bar says "cmdSelect" and the tab bar says "Click". The code is written in VB.NET and handles the click event for the "cmdSelect" button. It uses an Oracle database connection to select a faculty record based on the value in the "ComboName" dropdown. It also reserves the original name from the dropdown for updating purposes. If no match is found, it displays an alert message. Finally, it closes the database connection.

```

Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim cmdString As String = "SELECT faculty_id, name, office, phone, college, title, email FROM Faculty " &
        "WHERE name =:facultyName"
    Dim paramFacultyName As New OracleParameter
    Dim oraCommand As New OracleCommand
    Dim oraDataReader As OracleDataReader
    Application("oldFacultyName") = ComboName.Text 'reserve the old name for data updating
    paramFacultyName.ParameterName = "@facultyName"
    paramFacultyName.Value = ComboName.Text
    oraCommand.Connection = Application("oraConnection")
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add(paramFacultyName)
    Call ShowFaculty(ComboName.Text)
    oraDataReader = oraCommand.ExecuteReader
    If oraDataReader.HasRows = True Then
        Call FillFacultyReader(oraDataReader)
    Else
        Response.Write("<script>alert('No matched faculty found!')</script>")
    End If
    oraDataReader.Close()
    oraDataReader = Nothing
    oraCommand.Dispose()
    oraCommand = Nothing
End Sub

```

Figure 7.67. The modified Select button's click event procedure.

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 the server posts back a refreshed Faculty page based on the client's request, the contents of all controls on that page will be refreshed and the old contents will be lost.

Now let's develop the code for the Update button's click event procedure.

### 7.8.1.2 Add the Code to the Update Button and UpdateParameters Procedures

Open the Update button's click event procedure by double-clicking the Update button from the Faculty Web form, and enter the code shown in Figure 7.68 to this event procedure.

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

- An updating query string is declared first with `oldName` as the name of the dynamic parameter. This is because when you want to update the faculty name, the original name stored in the `ComboName` combo box control becomes the old name, and we need to distinguish this old name from the updated name.
- 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 data value returned from executing the `ExecuteNonQuery()` method later.

	cmdUpdate	▼	Click	▼
A	Protected Sub cmdUpdate_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click			
	Dim cmdString As String = "UPDATE Faculty SET name =: name, office =: office, phone =: phone, " & _			
	"college =: college, title =: title, email =: email WHERE (name =: oldName)"			
B	Dim oraCommand As New OracleCommand			
	Dim intUpdate As Integer			
C	txtID.Text = String.Empty	'clean the faculty_id textbox		
D	If txtName.Text <> Application("oldFacultyName") Then			
	ComboName.Items.Add(txtName.Text)			
	ComboName.Items.Remove(Application("oldFacultyName"))			
E	End If			
F	oraCommand.Connection = Application("oraConnection")			
G	oraCommand.CommandType = CommandType.Text			
H	oraCommand.CommandText = cmdString			
	UpdateParameters(oraCommand)			
I	intUpdate = oraCommand.ExecuteNonQuery()			
	oraCommand.Dispose()			
	oraCommand = Nothing			
	If intUpdate = 0 Then			
	Response.Write("<script>alert('The data updating is failed')</script>")			
	Exit Sub			
	End If			
	End Sub			

Figure 7.68. The modified Update button's click event procedure.

- C. Before we can perform the data updating, we first need to clean up the Faculty ID text box since we do not want to update this piece of information.
- D. 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 faculty name that is stored in the text box txtName. If the names are different, it means the user has updated the faculty name. In that case, we need to add the updated faculty name into the ComboName combo box control and remove the old faculty name from that control to allow users to select this updated faculty name from the combo box list to perform data actions against the database in the future.
- E. The Command object is initialized with the connection object, command type, and command text.
- F. The user-defined subroutine UpdateParameters(), whose detailed code is shown below, is called to assign all input parameters to the command object.
- G. The ExecuteNonQuery() method of the Command class is called to execute the data updating operation. This method returns feedback data to indicate whether this data updating is successful or not, and this returned data is stored to the local integer variable intUpdate.
- H. A cleaning job is performed to release all data objects used in this procedure.
- I. The data value returned from calling the ExecuteNonQuery() is exactly equal to the number of rows that have been successfully updated in the database. If this value is zero, which means that no row has been updated and this data updating has failed, a warning message is displayed and the

<b>Faculty</b>	<span style="font-size: 2em;">▼</span>	<b>UpdateParameters</b>	<span style="font-size: 2em;">▼</span>
<pre>Private Sub UpdateParameters(ByRef cmd As OracleCommand)     cmd.Parameters.Add("name", OracleType.Char).Value = txtName.Text     cmd.Parameters.Add("office", OracleType.Char).Value = txtOffice.Text     cmd.Parameters.Add("phone", OracleType.Char).Value = txtPhone.Text     cmd.Parameters.Add("college", OracleType.Char).Value = txtCollege.Text     cmd.Parameters.Add("title", OracleType.Char).Value = txtTitle.Text     cmd.Parameters.Add("email", OracleType.Char).Value = txtEmail.Text     cmd.Parameters.Add("oldName", OracleType.Char).Value = Application("oldFacultyName") End Sub</pre>			

**Figure 7.69.** The modified subroutine UpdateParameters.

procedure is exited. Otherwise, if this value is nonzero, this data updating is successful.

Finally, let's develop the code for the user-defined subroutine UpdateParameters(). Create this user-defined subroutine by typing the code shown in Figure 7.69 inside the Faculty class.

Seven input parameters are assigned to the Parameters collection property of the Command object using the Add() method. One important point for this subroutine is the last input parameter or the dynamic parameter oldName. The original or old faculty name oldFacultyName stored in the Application state must be used as the value for this parameter. Some readers may argue with me: the original or old faculty name is located in the ComboName combo box control, and we can get it directly from that control without using this global variable. Well, this statement would be correct for a Windows-based application, but for the Web-based application, it is absolutely wrong. Recall that when the user clicks the Update button to perform a data updating action, this updating request is sent to the server and the server posts back a refreshed Faculty page to the client. All old or original data stored in text boxes or combo boxes in the previous page will be lost. In other words, the contents of all text boxes and combo boxes in this refreshed page are different from the contents of the previous pages. A wrong updating may occur if you still use the faculty name stored in the ComboName combo box control in the current or refreshed page.

At this point we have finished all code development for the data updating actions in the Oracle database using the Faculty page. Before we can run the project to test this data updating functionality, make sure that the starting page is the LogIn page and the default faculty image file Default.jpg has been stored in our default folder. To check the starting page, right-click the project icon from the Solution Explorer window, select the Start Options item from the popup menu, and then check the Specific page radio button and select LogIn.aspx as the starting page.

Now you can run the project to test the data updating functionality in the Faculty page against the Oracle database.

### 7.8.2 Develop Stored Procedures to Perform the Data Deletion

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 Course tables, and then one can delete the desired record from the Faculty table. The reason is that 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 Course tables, and then delete the target record from the parent table. An easy way to perform these multiple deleting queries is to use a stored procedure to perform this data deletion.

### 7.8.2.1 Delete an Existing Record from the Faculty Table

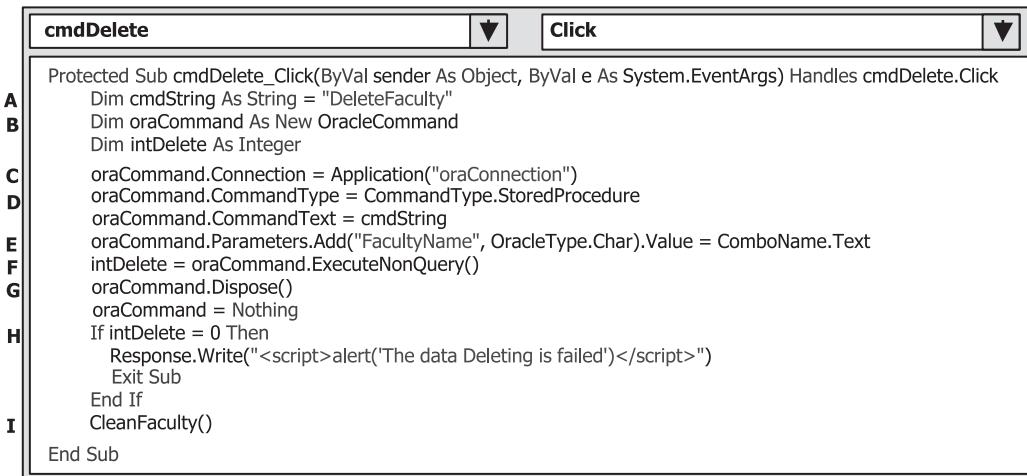
Recall that in Section 6.8.4 in Chapter 6, we discussed how to develop a stored procedure in the Oracle database and use that stored procedure to perform the data deleting operation in the Oracle database. In this section we still want to use our Faculty table as an example to show how to delete an existing record from related tables.

In our sample database, there are two child tables related to our Faculty table, the LogIn table and the Course table. These two child tables are connected with the Faculty table by using the faculty\_id, which is a primary key in the Faculty table and a foreign key in the 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 tables, such as the LogIn and 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 four to six records related to each faculty member in the Course table since each faculty member can teach four to six courses.
3. Delete the faculty member from the parent table, 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 complex data deleting operation. The complete data deleting operation can be divided into the following three steps:

1. Develop the stored procedure in the Oracle database to perform the multi-table data deleting functionality.
2. Call the stored procedure from the ASP.NET Web application to perform the data deletion in the Oracle database.
3. Validate the data deletion after the data deleting operation.



```

cmdDelete_Click
Protected Sub cmdDelete_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdDelete.Click
    Dim cmdString As String = "DeleteFaculty"
    Dim oraCommand As New OracleCommand
    Dim intDelete As Integer
    oraCommand.Connection = Application("oraConnection")
    oraCommand.CommandType = CommandType.StoredProcedure
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add("FacultyName", OracleType.Char).Value = ComboName.Text
    intDelete = oraCommand.ExecuteNonQuery()
    oraCommand.Dispose()
    oraCommand = Nothing
    If intDelete = 0 Then
        Response.Write("<script>alert('The data Deleting is failed')</script>")
        Exit Sub
    End If
    CleanFaculty()
End Sub

```

**Figure 7.70.** The code for the Delete button's click event procedure.

To save time and space, we will not provide a duplicated discussion of how to create a stored procedure in the Oracle database environment to perform this data deleting operation since we discussed this topic in great detail in Section 6.8.4.1. Refer to that section to get more detailed material about this issue. We will use the stored procedure DeleteFaculty, which was developed in Section 6.8.4.1, to perform data deletion in this section.

In the following part, we assume that we have finished developing the stored procedure DeleteFaculty and we only take care of the coding for steps 2 and 3.

### 7.8.2.2 Develop the Code for the Delete Button's Click Event Procedure

Open the Delete button's click event procedure by double-clicking the Delete button from the Faculty Web form window, and enter the code shown in Figure 7.70 into this event procedure.

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

- The content of the query string is now equal to the name of the stored procedure DeleteFaculty that we created in the Oracle database in Section 6.8.4.1. Refer to that section to get the detailed code 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.
- The data object and local variable used in this procedure are declared here. The integer variable intDelete is used to hold the value returned by executing the data updating method ExecuteNonQuery() of the Command class later.
- The Command object is initialized with the associated objects. The first object is the Connection object oraConnection that is stored in the Application state.
- 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 while the project runs.

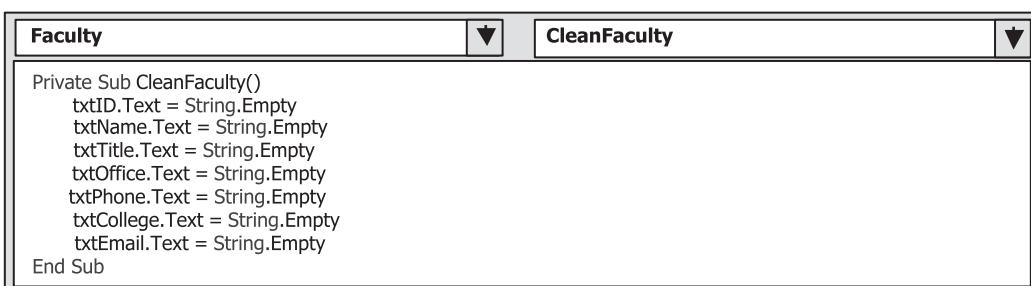
- E. The dynamic parameter is initialized with the real parameter faculty name that is stored in the ComboName combo box control. One point to note is that you must use the faculty name stored in this combo box control, not the faculty name stored in the faculty name text box control for this dynamic parameter since the latter is an updated faculty name, 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 the ExecuteNonQuery() method is equal to the number of rows that have been successfully deleted from the database. If this value is zero, which means that no row has been deleted from the database and this data deletion has failed, a warning message is displayed and the procedure is exited. Otherwise, if this value is nonzero, at least one row has been deleted from the database and this data deletion is successful.
- I. A user-defined subroutine CleanFaculty(), whose detailed code is shown below, is called to clean up all faculty information stored in the seven text boxes.

The code for the subroutine CleanFaculty() is shown in Figure 7.71.

The functionality of this piece of code is straightforward and easy understand. An Empty property of the String class is assigned to all text boxes to make them empty to clean them up.

Now we have finished all code development for the data deleting action in the Oracle database using the Faculty page. Before we run the project to test the data deleting functionality, make sure that the starting page is the LogIn page and a default faculty photo file has been stored in the default folder in which our Web application project is located.

Now we can run the project to test the data deleting functionality. Click the Start Debugging button to run the project. Enter a suitable username and password in the LogIn page, and select Faculty Information from the Selection page to open the Faculty page. Then keep the default faculty name, Ying Bai, selected in the



The screenshot shows a Microsoft Word document window. The title bar says "Faculty" on the left and "CleanFaculty" on the right. The main content area contains the following VB.NET code:

```
Private Sub CleanFaculty()
    txtID.Text = String.Empty
    txtName.Text = String.Empty
    txtTitle.Text = String.Empty
    txtOffice.Text = String.Empty
    txtPhone.Text = String.Empty
    txtCollege.Text = String.Empty
    txtEmail.Text = String.Empty
End Sub
```

Figure 7.71. The code for the subroutine CleanFaculty.

ComboName combo box control and click the Select button to retrieve and display this faculty member's information in the Faculty page. Now click the Delete button from this page to try to delete this record from the Faculty table in our sample database. Immediately you will find that all seven text boxes that contain the selected faculty member's information are cleaned up. Does that mean our data deletion was successful? Let us perform the following steps to confirm it.

### 7.8.2.3 Validate the Data Deleting Actions

There are two ways to confirm this data deletion. The first way is to try to retrieve this deleted faculty record from the database; the data deletion would be successful if no such faculty information could be found and retrieved from the database. The second way is to open the database to check the associated tables to confirm this data deletion.

First, let us do this confirmation using the first way. Still in the Faculty page, keep the faculty name Ying Bai selected in the ComboName combo box control and click the Select button to retrieve this faculty member's information from the database and display it in the Faculty page. The warning message "No matched faculty found!" is displayed, which means that the piece of faculty information has been successfully deleted from the database.

Next, let us open the Oracle database to check the associated tables to confirm this data deletion. Open the Oracle Database 10g XE home page by going to the Start>All Programs|Oracle Database 10g Express Edition|Go To Database Home Page menu item. On the opened Login page, enter the username and password and then click the Login button to open the Home page. Click the drop-down arrow on the Object Browser icon and select the item Browse|Tables to open the Table page.

On the opened Table page, click the FACULTY table from the left pane, and then click the Data tab to open this table, which is shown in Figure 7.72. You can

The screenshot shows the Oracle Database 10g XE Object Browser in Microsoft Internet Explorer. The title bar reads "Object Browser - Microsoft Internet Explorer". The address bar shows the URL: "Http://127.0.0.1:9080/index/F?p=4500:1001:2216947051557957::NO::P1\_CURRENT\_TYPE:TABLE". The left sidebar lists various database tables. The main content area is titled "FACULTY" and shows a grid of data with columns: EDIT, FACULTY\_ID, NAME, OFFICE, PHONE, COLLEGE, and TITLE. The data grid contains the following rows:

EDIT	FACULTY_ID	NAME	OFFICE	PHONE	COLLEGE	TITLE
	A52990	Black Anderson	MTC-218	750-378-9907	Virginia Tech	Professor
	A77567	Debby Angles	MTC-320	750-330-2276	University of Chicago	Associate Professor
	B06750	Alice Brown	MTC-257	750-330-6650	University of Florida	Assistant Professor
	B08590	Satish Bhalla	MTC-214	750-378-1061	University of Notre Dame	Associate Professor
	H09118	Jeff Henry	MTC-336	750-330-6650	Ohio State University	Associate Professor
	J03406	Steve Johnson	MTC-118	750-330-1116	Harvard University	Distinguished Professor
	K08880	Jenney King	MTC-324	750-378-1230	East Florida University	Professor

Figure 7.72. The Faculty table after the data deletion.

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 relationships with other tables, such as the LogIn and Course tables. To be precise, the Faculty table has relationships with all four other tables in our sample database, which include the Student and StudentCourse tables. But at this moment, we only take care of the LogIn and Course tables, and we will discuss the other two tables in the next section.

Open the LogIn and Course tables by clicking each of them one by one from the left pane, and you can find that the records related to the faculty member Ying Bai have been deleted from the LogIn and Course tables. The relationship between the Faculty and LogIn tables as well as between the Faculty and Course tables is set up by the faculty\_id, which is a primary key in the Faculty table and a foreign key in both the LogIn and Course tables.

This confirms that our data deletion was successful.

But the story is not finished. As you know, the Faculty table has relationships with all four other tables in our sample database, which include the Student and 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, identified by the student\_id, who took the four courses taught by the deleted faculty member Ying Bai have also been deleted from the StudentCourse table! Why were those records 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 foreign and primary keys. Now let's have a close look at those to try to solve our problem in the next section.

#### 7.8.2.4 The On Delete Cascade Constraint in the Data Table

Recall that in Section 2.11.4 in Chapter 2, we used the constraint property to set up the foreign key and create the relationships between tables. When we add a foreign key to a table, we need to indicate the Constraint Name and the Constraint Type. For example, to create a foreign key for the StudentCourse table and set up a relationship between the Course and StudentCourse tables, we selected the course\_id as the primary key for the Course table and used it as a foreign key for the StudentCourse table. To create this foreign key to the StudentCourse table, the Constraint Name and the Constraint Type were STUDENTCOURSE\_COURSE\_FK and Foreign Key.

The important point is that there is a check box named **On Delete Cascade**, which is located at the right of the Constraint Type text box. We made this check box checked when we created this foreign key for the StudentCourse table. To make this issue clear, we redisplay Figure 2.65, as Figure 7.73 in this section.

It can be found seen in this figure that the On Delete Cascade check box is checked. This means that all records related to the foreign key course\_id in this

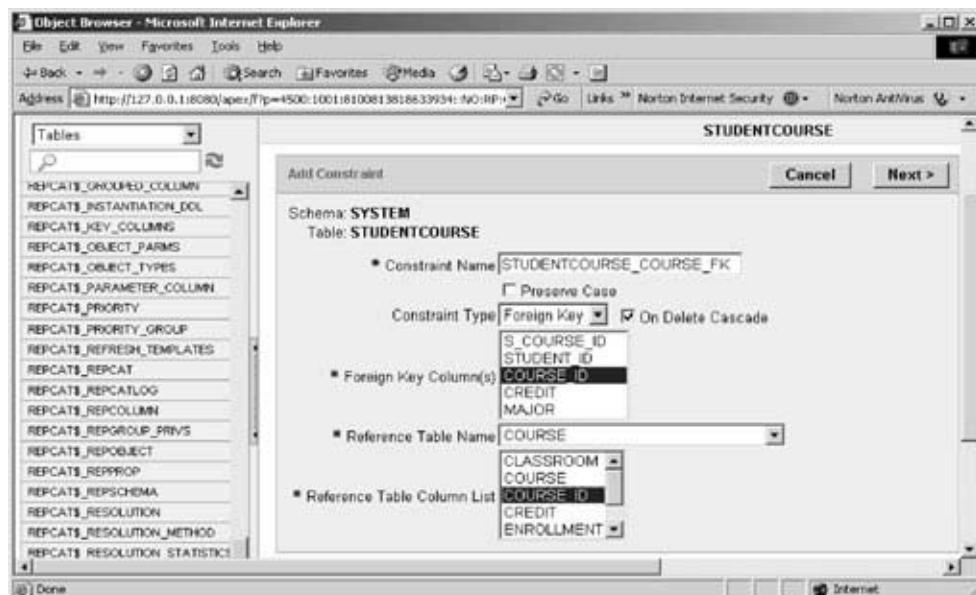


Figure 7.73. Create the foreign key between the StudentCourse and the Course table.

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 or On Delete Cascade. The word *cascade* means series, and the cascaded deleting means that if the records containing 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 been also 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 check box On Delete Cascade was checked when we set up the relationship between these two tables, all records related to this foreign key course\_id in the StudentCourse table are 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, identified by their course\_id, that are taught by the faculty member Ying Bai are deleted from the Course table, all records related to that course\_id, which is a foreign key in the StudentCourse table, will also be deleted since the course\_id is a primary key in the Course table. It is the Oracle database engine that performs this cascaded or series data deleting if the check box On Delete Cascade is checked when the relationship is set up between tables. A similar thing happens to the student.id that is also a foreign key in the StudentCourse table.

Before we close Oracle Database 10g XE, it is highly recommended to restore all deleted records to the associated tables. Refer to Tables 7.13–7.16 to add those records back to the associated tables. Now you can close Oracle Database 10g XE.

**Table 7.13.** The Data to Be Added to the Faculty Table

faculty_id	name	office	phone	college	title	email
B78880	Ying Bai	MTC-211	750-378-1148	Florida Atlantic University	Assistant Professor	ybai@college.edu

**Table 7.14.** The Data to Be Added to the Login Table

user_name	pass_word	faculty_id	student_id
ybai	reback	B78880	

**Table 7.15.** The Data to Be Added to 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

The completed Web application project OracleWebUpdateDelete is located in the folder **DBProjects\Chapter 7** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354).

## 7.9 CHAPTER SUMMARY

A detailed and completed introduction to ASP.NET and the .NET Framework was 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 Web application developments and implementations.

**Table 7.16.** The Data to Be Added to 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

Following the introduction section, a detailed discussion on how to install ASP.NET and configure the environment to develop ASP.NET Web applications was provided. Some essential tools, such as the Web server, IIS, and FrontPage Server Extensions 2000, as well as the installation process of these tools were introduced and discussed in detail.

Starting from Section 7.3, the detailed development and building process of ASP.NET Web applications to access databases were discussed with six real Web application projects. Two popular databases, SQL Server and Oracle, were utilized as the target databases. The six real ASP.NET Web application projects included the following:

1. Developing an ASP.NET Web application to select and display data from a Microsoft SQL Server database.
2. Developing an ASP.NET Web application to select and display data from an Oracle database.
3. Developing an ASP.NET Web application to insert data into a Microsoft SQL Server database.
4. Developing an ASP.NET Web application to insert data into an Oracle database.
5. Developing an ASP.NET Web application to update and delete data in a Microsoft SQL Server database.
6. Developing an ASP.NET Web application to update and delete data in an Oracle database.

Stored procedures are utilized in the last two projects to perform the data updating and deleting actions in two popular of kinds databases more efficiently and conveniently. The detailed discussion of the data deletion order is provided to help readers understand the integrity constraint built into the relational database. It is not easy to update or delete data from related tables in a relational database, and the clear and deep discussion provided on this topic should significantly benefit the readers and improve their knowledge of and hands-on experience with these issues.

## **7.10 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 the .NET Framework are two different models that provide development environments for 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 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 is used to build Web applications.

- \_\_\_\_\_ 6. The fundamental component of ASP.NET is the Web form. A Web form is a 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. Like traditional Web pages, an ASP.NET Web page can only run scripts on the client side.
- \_\_\_\_\_ 9. The controls you add to a Web form will run on the Web server when the 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 a 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 send 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 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(n) \_\_\_\_\_ 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. Default.aspx file
  - c. Config file
  - d. Application state
6. One needs to use the \_\_\_\_\_ method to display a message 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 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
10. To display any message on a running Web page, one must use the \_\_\_\_\_ method.
  - a. MessageBox.Show()
  - b. Response()
  - c. Response.Redirect()
  - d. Response.Write()

### III. Exercises

1. Write a paragraph to answer the following questions:
  - a. What is ASP.NET?
  - b. What is the main component of the ASP.NET Web application?
  - c. How is an ASP.NET Web application executed?
2. Suppose we want to delete one record from the Student table in our sample database CSE\_DEPT based on one student\_id, H10210. List all deleting steps and deleting queries including the data deletion from the child and the parent tables.
3. Following is a piece of code developed in the Page\_Load event procedure. Explain the functionality of the If Not IsPostBack Then block.

(Page Events)	▼	Load	▼
<pre>Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load     If Application("oraConnection").State &lt;&gt; ConnectionState.Open Then         Application("oraConnection").Open()     End If     If Not IsPostBack Then         'these items can only be added into the combo box in one time         ComboName.Items.Add("Ying Bai")         ComboName.Items.Add("Satish Bhalla")         ComboName.Items.Add("Black Anderson")         ComboName.Items.Add("Steve Johnson")         ComboName.Items.Add("Jenney King")         ComboName.Items.Add("Alice Brown")         ComboName.Items.Add("Debby Angles")         ComboName.Items.Add("Jeff Henry")     End If End Sub</pre>			

4. Add a Web page and develop the code to perform data deletion for the Student form in the SQLWebUpdateDelete project (the project file is located in the folder **DBProjects\Chapter 7** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354)).
5. Use the Cascade option for relationships 1, 2, and 3 listed in Section 7.7.4.3 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 7.7.4.3 to get a detailed discussion of this issue).
6. Develop a Web page called Student.aspx and create a stored procedure to delete one record from the Student table by using the project OracleWebUpdateDelete (the project file is located in the folder **DBProjects\Chapter 7** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), and 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 the record from the Student table.

---

# 8

---

## ASP.NET Web Services

In the last chapter, we discussed ASP.NET Web applications in detail. In this chapter, we will concentrate on another ASP.NET-related topic – ASP.NET Web Services.

Unlike 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, ASP.NET Web Services provide an automatic way to search, identify, and return the information desired by the user through a set of methods installed in the Web server, and these methods can be accessed by a computer program, not the user, via the Internet. Another important difference between ASP.NET Web applications and ASP.NET Web Services is that the latter do not provide any graphical user interfaces (GUIs) and users need to create those GUIs themselves to access the Web Services via the Internet.

After finishing this chapter, you will

- Understand the structure and components of ASP.NET Web Services, such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI)
- Create correct SOAP namespaces for the Web Services to make used names and identifiers unique in the user's document
- Create suitable security components to protect the Web methods
- Build professional ASP.NET Web Service projects to access our sample database to obtain required information
- Build client applications to provide GUIs to consume a Web Service
- Build professional ASP.NET Web Service projects to access our sample database to insert new information into the database
- Build professional ASP.NET Web Service projects to access our sample database to update and delete information in the database

In order to help readers understand this chapter completely, first we will provide a detailed discussion of ASP.NET Web Services and their components.

## 8.1 WHAT ARE WEB SERVICES AND THEIR COMPONENTS?

Essentially, Web Services can be considered a set of methods that are installed in a Web server and can be called through the Internet by computer programs installed on the clients. These methods can be used to locate and return the target information required by the computer programs. Web Services do not require the use of browsers or HTML, and therefore Web Services are sometimes called *application services*.

To effectively find, identify, and return the target information required by computer programs, a Web Service needs the following components:

1. XML (Extensible Markup Language)
2. SOAP (Simple Object Access Protocol)
3. UDDI (Universal Description, Discovery, and Integration)
4. WSDL (Web Services Description Language)

The functionality of each component is listed below.

XML is a text-based data storage language that uses a series of tags to define and store data. The 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. The Web Services platform is XML + HTTP (hypertext transfer protocol), and HTTP is the most popular Internet protocol. But 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 code usable in Web Services and recognizable by applications developed in different platforms and programming languages, XML is used for coding in Web Services.

SOAP is a communication protocol used for communication between applications. Essentially, SOAP is a simple XML-based protocol to help applications developed in different platforms and languages exchange information over HTTP. Therefore, SOAP is a platform-independent and language-independent protocol, which means that it can run in any operating system with any programming language. 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, it 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 the way 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 when 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 messages can be used to communicate with a Web Service using 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 messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly and then are bound to a concrete network protocol and message format to define an endpoint.

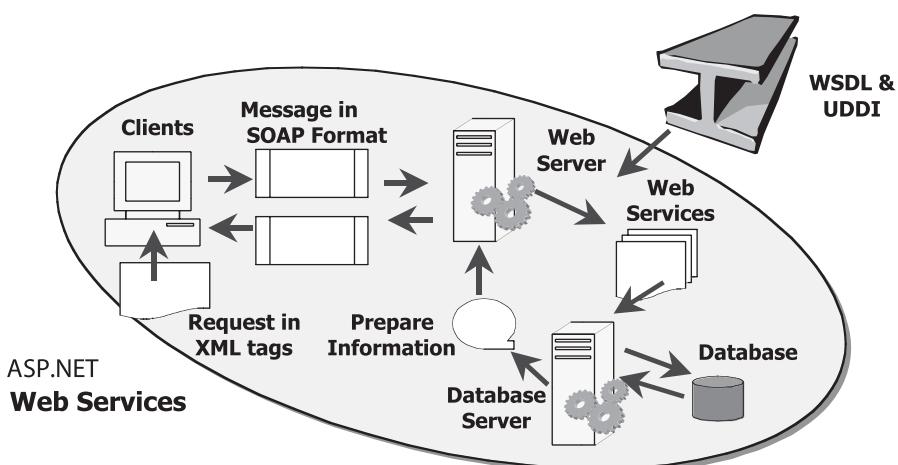
UDDI is an XML-based directory for businesses to list themselves on the Internet, and the goal of this directory is to enable companies to find one another on the Web and make their systems interoperable for E-commerce. UDDI is often considered to be like the yellow and white pages in a telephone book. It allows businesses to list themselves by name, products, locations, or the Web Services they offer.

In summary, based on these components and their roles discussed above, we can conclude the following:

- 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 describe the Web Services available in a WSDL document format.
- UDDI is used to list all Web Services that are available to users and businesses.

Figure 8.1 illustrates these components and their roles in an ASP.NET Web Service process.

By now, we have obtained fundamental knowledge about ASP.NET Web Services and their components. Next, let's see how to build a Web Service.



**Figure 8.1.** A typical process of a Web Service.

## 8.2 PROCEDURES TO BUILD A WEB SERVICE

Different methods and languages can be used to develop different Web Services such as C# Web Services, Java Web Services, and Perl Web Services. In this section we only concentrate on developing ASP.NET Web Services using Visual Basic.NET 2005. Before we start building a real Web Service project, let's first take a closer look at the structure of a Web Service project.

### 8.2.1 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 the real code 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 contains the real Visual Basic.NET code for a simple Web Service. Visual Web Developer includes the top of this page three default declarations to help users to develop Web Services, at, which are as follows:

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
```

By default, a new code-behind file contains a class named Service, defined with the WebService and WebServiceBinding attributes. The class defines a default Web method named HelloWorld that is a placeholder, and you can replace it with your own method or methods later when you develop your own Web Service project.

3. The main Web Service page file Service.asmx is used to display information about the Web Service's methods and provide access to the Web Service's WSDL information. The extension .asmx means that this is an Active Service Method file, and the letter *x* is just a rotation of the plus symbol ASP.NET was called ASP+ in the early days. 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 a 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, the Connection string, and the 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 code related to building a Web Service is located in this page, and our coding development will concentrate on this page too.

### **8.2.2 The Real Considerations When Building a Web Service Project**

On the basis of the structure of a typical Web Service project, some issues related to building an actual Web Service project are emphasized here. 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 Internet Information Services (IIS) on the client computer, which works as a pseudo server. Then 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 code to power the Web Service, to trigger the associated Web methods to search, find, and retrieve the information required by the client, pack it into 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 are unique; that is, the names and identifiers cannot conflict with any name or 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, and 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; that would make our project's size terribly large if we needed a lot of information. A good solution is to instantiate an object based on that class and integrate 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, which is a convenient way to return all data.
3. We need to handle 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 code

to protect our real class that will be instantiated to an object that contains all the information we need, so this real class should be a child class inherited from the base class.

4. Since Web Services do not provide any GUIs, we need to develop GUIs in either Windows-based or Web-based applications to interface with Web Services and display returned information on GUIs.

Keeping these real issues in mind, now let's get a clear picture of the steps to build a Web Service project using an ASP.NET Web Service template.

### 8.2.3 Procedures to Build an ASP.NET Web Service

As mentioned in the last section, a Web Service comprises 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 follow the following steps:

1. Create a new ASP.NET Web Service project.
2. Create a base class to handle the error checking to protect our real class.
3. Create our real Web Service class to hold all Web methods and code to response to requests.
4. Add all Web methods into our Web Service class.
5. Develop the code for those Web methods to perform the Web Services.
6. Build Windows-based and Web-based projects to consume the Web Service to pick up and display the required information on the GUI.
7. Store our ASP.NET Web Service project files in a safe location.

In this chapter, we try to develop the following projects to illustrate the building and implementation process of a Web Services project:

- Build professional ASP.NET Web Service projects to access an SQL Server database to obtain required information
- Build client applications to provide GUIs to consume a Web Service
- Build professional ASP.NET Web Service projects to access an Oracle database to obtain required information
- Build professional ASP.NET Web Service projects to insert new information into an SQL Server database
- Build professional ASP.NET Web Service projects to update and delete information in an SQL Server database
- Build professional ASP.NET Web Service projects to insert new information into an Oracle database
- Build professional ASP.NET Web Service projects to update and delete information in an Oracle database

On the basis of the procedures discussed above, we can start building our first Web Service project.

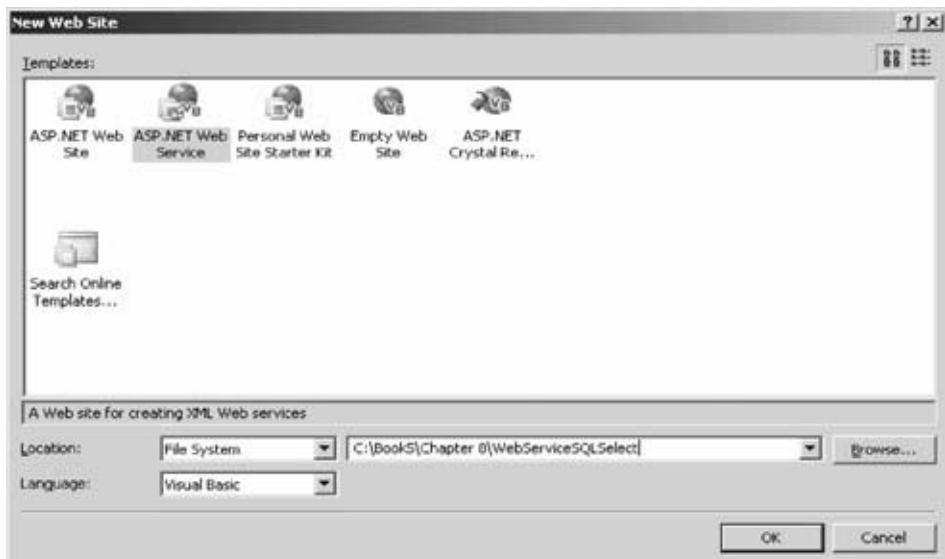


Figure 8.2. Create a new Web Service project.

### **8.3 BUILD AN ASP.NET WEB SERVICE PROJECT TO ACCESS AN SQL SERVER DATABASE**

To create a new ASP.NET Web Service project, open Visual Studio.NET 2005, and then go to the File|New Web Site menu item. In the New Web Site dialog box, select the ASP.NET Web Service item from the Templates list and enter our Web Service project name, WebServiceSQLSelect, into the box next to the Location box, which is shown in Figure 8.2.

One point to note is that Visual Studio.NET 2005 introduced a Web project model that can use either IIS or the Local File System to develop Web applications. This model is good only when developing ASP.NET Web Services and Web Pages that are running locally on a pseudo Web server. In our case, we will run our Web Service on our local machine and use it as a development server, so the File System is used for our server location, as shown in Figure 8.2.

Click the OK button to create this new project in our default folder, **C:\Book5\Chapter 8**.

#### **8.3.1 Files and Items Created in the New Web Service Project**

When this new Web Service project is created, four items are produced in our new project in the Solution Explorer window.

As discussed in the last section, the main service page file Service.asmx, which is used to display information about the Web Service's methods and provide access to the Web Service's WSDL information, and the configuration file web.config, which is used to set up a configuration for our new Web Service project, such as the namespaces for all kinds of Web components, connection strings for data components, and Web Services and Windows authentication modes, are automatically created and added into our new project. More importantly, the page file Service.asmx is

```

(General) (Declarations)
A Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
B <WebService(Namespace:="http://tempuri.org/")> _
C <WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
D <Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
E Public Class Service
  Inherits System.Web.Services.WebService
F   <WebMethod()>
    Public Function HelloWorld() As String
      Return "Hello World"
    End Function
  End Class

```

Figure 8.3. The default code for the code-behind page Service.vb.

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 Service.asmx only contains a compile directive when a new Web Service project is created and opened from the File System.

Two folders, App\_Code, which is used to store our code-behind page Service.vb, and App\_Data, which is used to save the project data, are also created. The code-behind page Service.vb is the place where we need to create and develop the code for our Web Services. This page contains a default class named Service that is defined with the WebService and WebServiceBinding attributes. The class defines a default Web method named HelloWorld that is a placeholder, and we can replace it with our own method or methods later based on the requirement of our Web Service project.

Now double-click this code-behind page Service.vb, which is shown in Figure 8.3, and let's have a closer look at the code in this page.

- The Web Services-related namespaces that contain 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 of those namespaces and their functionalities is shown in Table 8.1.
- Some WebService attributes are defined in this part. Generally WebService attributes are used to identify additional descriptive information about

**Table 8.1. The Web Service Namespaces**

Namespace	Functionality
System.Web	Enable browser and server communication using the .Net Framework
System.Web.Services	Enable creation of XML Web Services using ASP.NET
System.Web.Services.Protocol	Define the protocol used to transmit data across the wire during the communication between Web Service clients and servers

deployed Web Services. The namespace attribute is one example. As discussed in the last section, we need to use our own namespace to store and hold names and identifiers used in our SOAP messages to distinguish them from any other SOAP messages used by other Web Services. Here in this new project, Microsoft 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 Profile 1.1. Here, 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 design time by the designer.
- E. Our Web Service class Service 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 the string “Hello World” when it is returned to the client.

Next, double-click the main service page file Service.asmx that is the entry point of our project to open it, and only one line of code 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/
Service.vb" Class = "Service" %>
```

---

As mentioned in the last section, this code indicates the programming language in which the Web Service’s code-behind file is written, the code-behind file’s name and location, and the class that defines the Web Service. In a real application, both the code-behind file and the class should be renamed to match the file and class names used in our project, respectively. We will do this later when we develop our real project in the following sections. But first let’s run the default HelloWorld Web Service project to get a feeling of what it looks like and how it works.

Click the Start Debugging button to run the default HelloWorld project.

### **8.3.2 A Feeling of the HelloWorld Web Service Project As It Runs**

After the project starts, a message box is displayed with the warning shown in Figure 8.4.

Generally, a Web Service project should not be debugged when it is deployed, and this is defined in the web.config file with a control for 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 selection and click the OK



Figure 8.4. The Debugging Not Found message box.

button to continue to run our project. Our Service.asmx page should be the starting page, and a page is displayed as shown in Figure 8.5.

This page displays the Web Service class name Service 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, along with a recommendation that suggests that we create

```
C#
[WebService(Namespace = "http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

Figure 8.5. The running status of the default Web Service project.

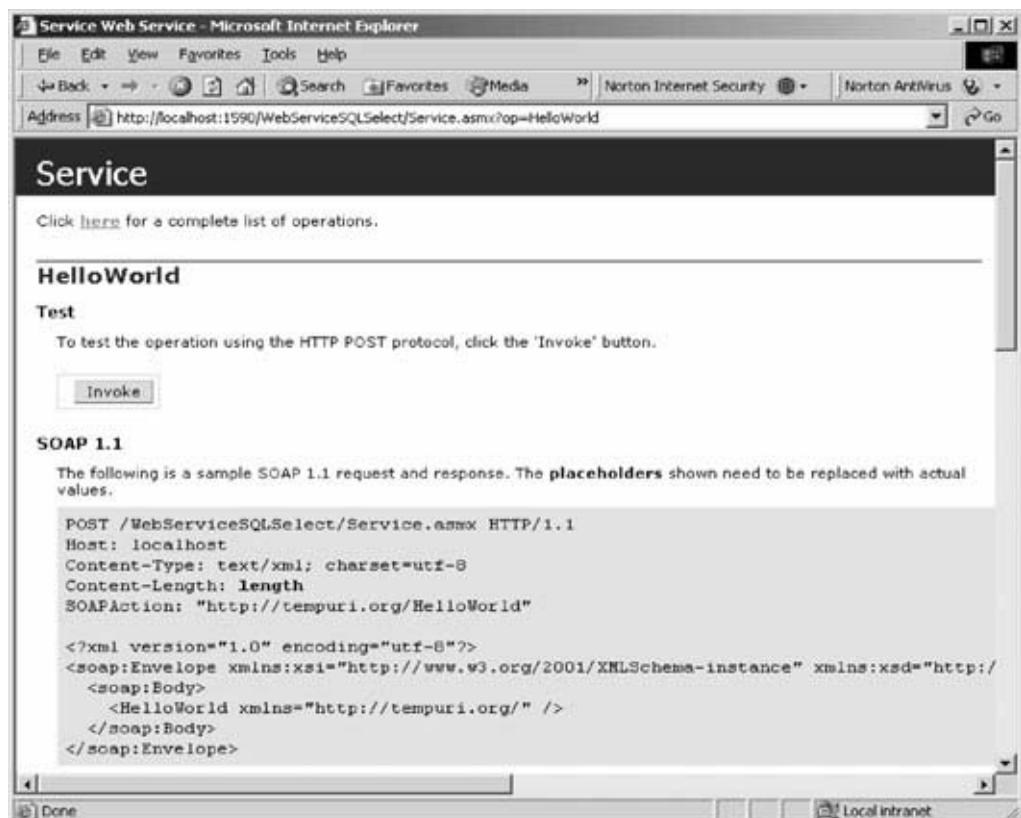


Figure 8.6. The test method page.

our own namespace to store our Web Service project. Following this recommendation, some example namespaces used in C#, Visual Basic, and C++ are listed.

Now let's access our Web Service by clicking the HelloWorld method. The test method page is shown in Figure 8.6.

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 the different protocols are displayed. These include the requesting message and responding message created in SOAP 1.1, SOAP 1.2, and HTTP Post. The placeholder 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 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 is displayed, as shown in Figure 8.7. When the default method HelloWorld is executed, the main service page Service.asmx sends a request to IIS, and furthermore, 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, pack the return data as a SOAP message, and send it back to the client. The returned message contains only the string object "Hello World" for this default method.

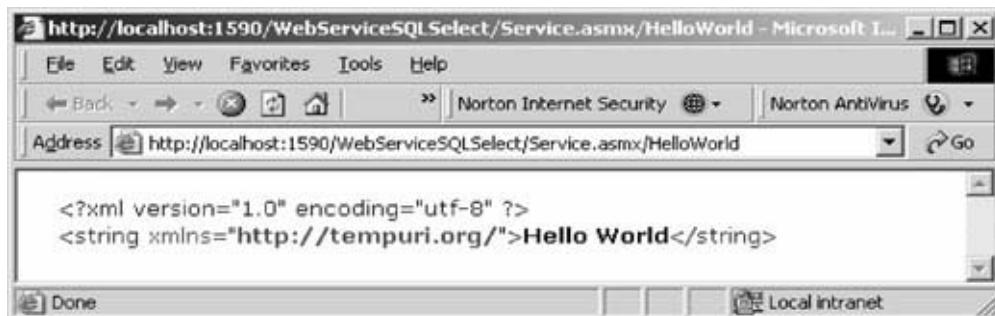


Figure 8.7. The running status of the default method.

In this returned result, the version and the encoding of the XML code used is indicated first. The `xmlns` attribute is used to indicate the namespace used by this `String` object, which contains only a string of “Hello World.”

As discussed in the previous section, ASP.NET Web Services do not provide GUIs, so the running result of this default project is represented using XML codes in some Web interfaces we have seen. This is because those Web interfaces are only provided and used for the testing of the default Web Service. In a real application, no such Web interface would be provided and displayed.

Click the Close button that is located on the upper at right corner of the browser for two browser pages.

At this point, we should have a basic understanding and feeling of a typical Web Service project and its structure as well as its operation. Next, we will create our own Web Service project by modifying this default project to perform the request to our sample database, to the Faculty table, to get the desired faculty information.

We will develop our Web Service project in the following sequence:

1. Modify the default Web Service project to make it our new Web Service project.
2. Create a base class to handle error-checking codes to protect our real Web Service class.
3. Create our real Web Service class to hold all Web methods and codes to 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 code for those Web methods to perform the Web Services.
6. Build Windows-based and Web-based projects to consume the Web Service to pick up and display the required information in a GUI.
7. Deploy our completed Web Service to IIS.

The modifications defined in step 1 include the renaming of the main service page, the code-behind page, the class, and the namespace defined in the code-behind page and the main service page.

Let's start from step 1.

### 8.3.3 Modify the Default Web Service Project

We will modify the default Web Service project to make it our new Web Service project and allow it to access our sample SQL Server database to pick up the desired faculty information.

The following modifications must be made to this default project:

- Rename the main service page from the default name Service to our new name, WebServiceSQLSelect.
- Rename the code-behind page from Service.vb to our new name, WebServiceSQLSelect.vb.
- Modify the names of the code-behind page and class in the main service page.
- Rename the namespace defined in the code-behind page.

Right-click the main service page, Service from the Solution Explorer window, select the Rename item from the popup menu, and rename this page WebServiceSQLSelect.asmx.

Perform a similar operation to the code-behind page Service.vb, and rename it WebServiceSQLSelect.vb.

Now open our new main service page WebServiceSQLSelect.asmx by double-clicking it, and change the compiler directive from

---



---

**CodeBehind = “~/App\_Code/Service.vb”**

---



---

to

---



---

**CodeBehind = “~/App\_Code/WebServiceSQLSelect.vb”**

---



---

Also change the class name from

---



---

**Class = “Service”**

---



---

to

---



---

**Class = “WebServiceSQLSelect”**

---



---

Go to the File|Save All menu item to save these modifications.

Now open our new code-behind page WebServiceSQLSelect.vb by double-clicking it from the Solution Explorer window, and perform the modifications that are shown in bold in Figure 8.8.

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

- A. We need to use our own namespace to replace the default namespace used by Microsoft to tell the ASP.NET runtime the location from which our Web

```

(General) (Declarations)
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
A <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
B     <WebMethod()>
        Public Function HelloWorld() As String
            Return "Hello World"
        End Function
    End Class

```

Figure 8.8. The modified code-behind page.

Service can be found and loaded as it runs. This specific namespace is unique because it is the home page of Cambridge University Press 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.

- B. The default class, Service, is replaced by a new class, WebServiceSQLSelect, which is our desired Web Service class used in this new project.

Now if you run this new project, a default Web interface is displayed with our new project name (Figure 8.9).

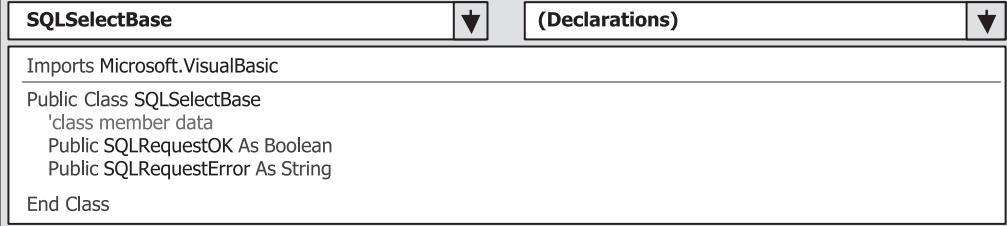
If you click the default method HelloWorld and then the Invoke button to test that method, you can find that the namespace has been updated to our new specific namespace.

### 8.3.4 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 likely



Figure 8.9. The running status of our new Web Service project.



The screenshot shows a code editor window with the title bar "SQLSelectBase". Below the title bar, there are two dropdown menus: one labeled "(Declarations)" and another with a downward arrow. The code area contains the following VB.NET code:

```

Imports Microsoft.VisualBasic

Public Class SQLSelectBase
    'class member data
    Public SQLRequestOK As Boolean
    Public SQLRequestError As String
End Class

```

**Figure 8.10.** The class member data.

reasons for that is the security issue. To report any errors or problems that occur in the processing of requests, a parent or base class is a good candidate to perform those jobs. We name this base class SQLSelectBase, and it has the following two member data items:

- **SQLRequestOK As Boolean:** True if the request is fine, otherwise False
- **SQLRequestError As String:** A string used to report the errors or problems

To create a new base class in our new project, right-click our new service project that is located at the top of the Solution Explorer window, and select the item Add New Item from the popup menu. In the Add New Item dialog box, select the Class item from the Template list, and enter SQLSelectBase into the name box as our new class name. Then click 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.

Now double-click this newly added class and enter the code shown in Figure 8.10 into this class as the class member data.

Two public class member data items, SQLRequestOK and SQLRequestError, are added into this new base class. These two data items will work together to report possible errors or problems during the processing of the request.

### **8.3.5 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 the base class SQLSelectBase we just created. We name this class the SQLSelectResult.

Right-click our new Web Service project from the Solution Explorer window, and select the item Add New Item from the popup menu. In the Add New Item dialog box, select the Class item from the Template list. Enter SQLSelectResult into the Name box as the name for this new class, and then click 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 this newly added class, and enter the code shown in Figure 8.11 into this class as the member data for this class.

Since this class will be instantiated to an object that will be returned to us with our desired faculty information as the Web method is called, all desired faculty information should be added into this class as the member data. When we make

```

SQLSelectResult
(Declarations)

Imports Microsoft.VisualBasic

Public Class SQLSelectResult
    Inherits SQLSelectBase

    'member data
    Public FacultyID As String
    Public FacultyOffice As String
    Public FacultyPhone As String
    Public FacultyCollege As String
    Public FacultyTitle As String
    Public FacultyEmail As String

End Class

```

**Figure 8.11.** The member data for the class SQLSelectResult.

a request to this Web Service project, and furthermore to our sample database, the following desired faculty information should be included and returned:

- Faculty\_id
- Faculty office
- Faculty phone
- Faculty college
- Faculty title
- Faculty email

All these pieces of information, which can be exactly mapped to columns in the Faculty table in our sample database, need to be added into this class as the member data. This does not look like a professional schema. A better option is not 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 these 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 will start our project with an easier task, 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 the added member data for this class.

As mentioned before, this class is a child class of our base class SQLSelectBase. In other words, this class is inherited from that base class. All six pieces of faculty information are declared here as the member data for this class.

Next, we need to take care of the Web method that will respond to our request and return the desired faculty information to us as this method is called.

### 8.3.6 Add Web Methods into Our Web Service Class

Before we can add a Web method to our project and perform the coding for it, we want to emphasize an important point that may be confused by users, that is, the Web Service class and the 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 the code 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. Keeping this difference in mind will help you understand them better as you develop a new Web Service project.

We can modify the default method HelloWorld and make it 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 sections to contain and return the faculty information we require.

### **8.3.7 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 functionalities as this method is called:

1. Set up a valid connection to our sample database.
2. Create all required data objects and local variables to perform the necessary data operations later.
3. Instantiate a new object from the application class SQLSelectResult, and use it as the returned object that contains all required faculty information.
4. Execute the associated data object's method to perform the data query to the Faculty table based on the input parameter Faculty Name.
5. Assign each 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.

#### **8.3.7.1 Web Service Connection Strings**

Among these functionalities, function 1 is one of the most challenging tasks. 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 one is a better way since the web.config file provides the attribute connectionStrings for this purpose and ASP.NET 2.0 recommends storing the data components' connection string in the web.config file.

In this project, we will use the second way to store our connection string. To do that, open the web.config file by double-clicking it, and enter the following code into the attribute connectionStrings:

---

```
<connectionStrings>
    <add name = "sql_conn" connectionString = "Server = localhost
    \SQLEXPRESS;
    Integrated Security = SSPI;Database = CSE_DEPT;" />
</connectionStrings>
```

---

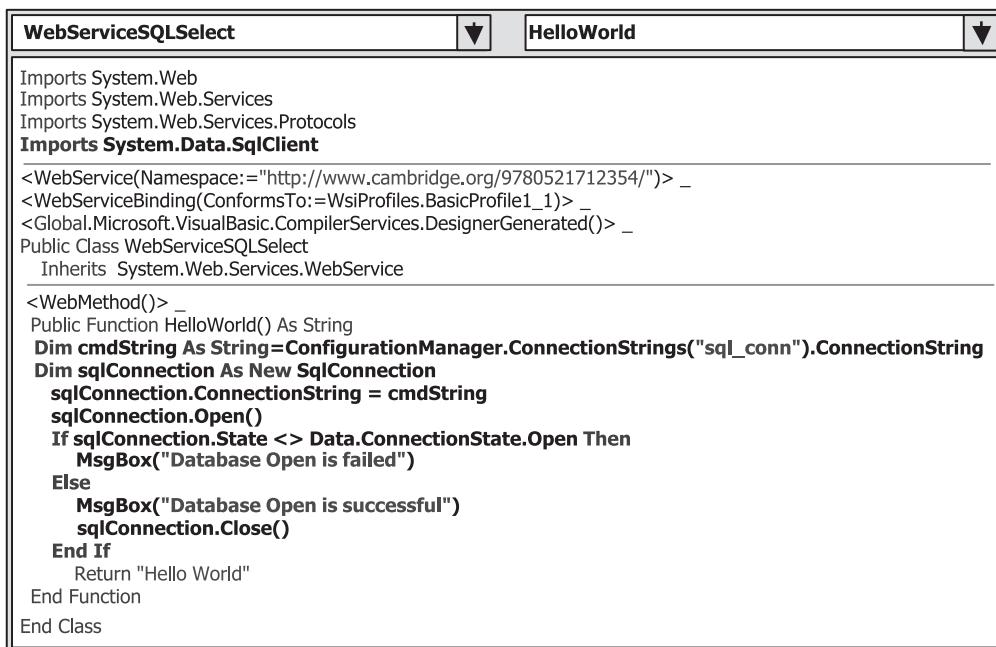
The following important points should be noted when creating this connection string:

1. This connectionStrings attribute must be written in a single line in the web.config file. However, because of the limitation of space, we used three lines to represent this attribute. But in your real code, 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. But 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 2005 database; instead, we are using SQL Server 2005 Express, so we have to add the InstanceName of our database, which is SQLEXPRESS, into this connection string to tell the ASP.NET runtime to make the correct connection. Attach this instance name after the localhost in the ServerName item.

To test and confirm this connectionString, we can develop some code and modify the code of the default HelloWorld Web method in the code-behind page. Close the web.config file and open the code-behind page WebServiceSQLSelect.vb by double-clicking it from the Solution Explorer window, and enter the codes shown in Figure 8.12 into this page.

All modified code is highlighted in bold. Let's see how this piece of code works to test our connection string defined in the web.config file.

- A 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.



```

WebServiceSQLSelect
HelloWorld

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 8.12. The modified code to test the connection string.

- 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 a default connection configuration. The connection name sql\_conn that works as an argument for this property must be identical to the name we used for the connection name in the web.config file. When this property is used, it returns a ConnectionStringSettingsCollection object containing the contents of the ConnectionStringsSection object for the current application's default configuration, and a ConnectionStringsSection 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 success message is displayed and the connection is closed.

Now you can run the project by clicking the Start Debugging button. Click the HelloWorld method from the built-in Web interface, and then click the Invoke button to execute that method to test our database connection.

A success message should be displayed if this connection is fine. Click 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 is that the MsgBox() is used and is displayed behind the current Web page when it is activated. So you need to move the current page by dragging it down, and then you can find that MsgBox. Click OK to that MsgBox and the project will continue and the running result will be displayed.

Another issue is that this piece of code is used only for testing, and we will modify this piece of code and place it into a user-defined function called SQLConn() later when we develop our real project.

### **8.3.7.2 Modify the Existing Web Method**

Now let's take care of our Web methods. In this project, we want to modify the default method HelloWorld as our first Web method and develop code for this method to complete those functionalities (2–7) listed at the beginning of this section.

Open the Web Service code-behind page if it is not open, 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 the 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 SQLResult.
5. Create the following data components used in this method:
  - a. SQL Command object sqlCommand
  - b. SQL Data Reader object sqlReader
6. Replace the default returned object in the method from the “Hello World” string to the newly created object SQLResult.
7. Move the connection testing code we developed in this section into a user-defined function SQLConn().

Your finished Web method GetSQLSelect() should match the one shown in Figure 8.13.

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

- A. Modification steps 1, 2, and 3 listed above are performed in this line. The method's name and the returned data type are updated to GetSQLSelect and SQLSelectResult, respectively. Also, the input parameter FacultyName is added into this method as an argument.
- B. Modification step 4 is performed in this line, and an instance of the application class SQLSelectResult is created here.
- C. Modification step 5 is performed in this line, and two SQL data objects are created: sqlCommand and sqlReader.
- D. Modification step 6 is performed in 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 code we developed to test the connection string above.

```

WebServiceSQLSelect           GetSQLSelect
A  <WebMethod()>_
B  Public Function GetSQLSelect (ByVal FacultyName As String) As SQLSelectResult
C    Dim sqlConnection As New SqlConnection
D    Dim SQLResult As New SQLSelectResult
E    Dim sqlCommand As New SqlCommand
F    Dim sqlReader As SqlDataReader
G    Return SQLResult
End Function

Protected Function SQLConn() As SqlConnection
  Dim cmdString As String = ConfigurationManager.ConnectionStrings("sql_conn").ConnectionString
  Dim conn As New SqlConnection
  conn.ConnectionString = cmdString
  conn.Open()
  If conn.State <> Data.ConnectionState.Open Then
    MsgBox("Database Open is failed")
    conn = Nothing
  End If
  Return conn
End Function

```

**Figure 8.13.** The modified Web method GetSQLSelect.

- F. If this connection fails, 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 code to execute the associated data object's method to perform the data query to the Faculty table based on the input parameter Faculty Name.

### **8.3.7.3 Develop the Code to Perform the Database Queries**

To perform the database query via our Web Service project, we need to perform the following coding jobs:

- Add the main code to perform the data query into our Web method.
- 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 as the project runs.

Now let's first concentrate on adding the code to perform the data query to our sample database CSE\_DEPT.

Open our code-behind page and add the code shown in Figure 8.14 into our Web method. The code developed in the previous sections is highlighted a gray background.

Let's take a closer look at the newly added code to see how it works.

- 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 paid attention to is that a plus symbol is used here to replace the concatenating operator (&) that was 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 code is shown in Figure 8.13, 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 code is shown below, 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

WebServiceSQLSelect	▼	GetSQLSelect	▼
---------------------	---	--------------	---

```

Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
A Imports System.Data
B 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 GetSQLSelect(ByVal FacultyName As String) As SQLSelectResult
        Dim cmdString As String = "SELECT faculty_id, office, phone, college, title, email FROM Faculty " + _
            "WHERE name LIKE @facultyName"
        Dim sqlConnection As New SqlConnection
        Dim SQLResult As New SQLSelectResult()
        Dim sqlCommand As New SqlCommand
        Dim sqlReader As SqlDataReader

C SQLResult.SQLRequestOK = True
D sqlConnection = SQLConn()
E If sqlConnection Is Nothing Then
            SQLResult.SQLRequestError = "Database connection is failed"
            ReportError(SQLResult)
            Return Nothing
F End If
        sqlCommand.Connection = sqlConnection
        sqlCommand.CommandType = CommandType.Text
        sqlCommand.CommandText = cmdString
        sqlCommand.Parameters.AddWithValue("@facultyName", SqlDbType.Text).Value = FacultyName
        sqlReader = sqlCommand.ExecuteReader

G If sqlReader.HasRows = True Then
            Call FillFacultyReader(SQLResult, sqlReader)
H Else
            SQLResult.SQLRequestError = "No matched faculty found"
            ReportError(SQLResult)
I End If

J If Not sqlReader Is Nothing Then sqlReader.Close()
        sqlReader = Nothing
K If Not sqlConnection Is Nothing Then sqlConnection.Close()
        sqlConnection = Nothing
        Return SQLResult
    End Function

```

**Figure 8.14.** The modified code for the Web method.

input parameter @facultyName is assigned with the real input parameter FacultyName that is an input parameter to the Web method. One issue is the data type for this parameter. For this application, it does not matter whether SqlDbType.Char or SqlDbType.Text is used.

- G. The ExecuteReader() method of the command class is called to invoke the DataReader and to perform the data query to our Faculty table.
- 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 an object returned 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, 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.

#### 8.3.7.4 Develop the Code for Subroutines Used in the Web Method

The code for the subroutine FillFacultyReader() is shown in Figure 8.15.

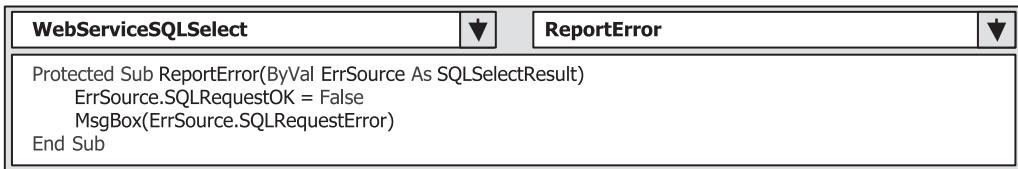
Let's have a look at the code in this subroutine to see how it works.

- A. The Read() method of the DataReader is executed to read out the queried data row. In our case, only one row, which is matched to the input faculty name, is read out and fed into the DataReader object sReader.
- B. A With...End With statement is utilized here to simplify the assignment operations. The object sResult that is our returning object is attached after the keyword With, and all member data items 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

WebServiceSQLSelect	FillFacultyReader	
<b>A</b> <pre>Protected Sub FillFacultyReader(ByRef sResult As SQLSelectResult, ByVal sReader As SqlDataReader)     If sReader.Read() = True Then         With sResult             .FacultyID = Convert.ToString(sReader("faculty_id"))             .FacultyOffice = Convert.ToString(sReader("office"))             .FacultyPhone = Convert.ToString(sReader("phone"))             .FacultyCollege = Convert.ToString(sReader("college"))             .FacultyTitle = Convert.ToString(sReader("title"))             .FacultyEmail = Convert.ToString(sReader("email"))         End With     End If End Sub</pre>	<b>B</b> <pre>Protected Sub FillFacultyReader(ByRef sResult As SQLSelectResult, ByVal sReader As SqlDataReader)     If sReader.Read() = True Then         With sResult             .FacultyID = Convert.ToString(sReader("faculty_id"))             .FacultyOffice = Convert.ToString(sReader("office"))             .FacultyPhone = Convert.ToString(sReader("phone"))             .FacultyCollege = Convert.ToString(sReader("college"))             .FacultyTitle = Convert.ToString(sReader("title"))             .FacultyEmail = Convert.ToString(sReader("email"))         End With     End If End Sub</pre>	

Figure 8.15. The code for the subroutine FillFacultyReader.



```
WebServiceSQLSelect
ReportError

Protected Sub ReportError(ByVal ErrSource As SQLSelectResult)
    ErrSource.SQLRequestOK = False
    MsgBox(ErrSource.SQLRequestError)
End Sub
```

Figure 8.16. The code for the subroutine ReportError.

of each column in the query string cmdString must be used to locate each data item if this method is used.

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 code for the subroutine ReportError() is shown in Figure 8.16.

The input parameter to this subroutine is our returning object. A False is assigned to the SQLRequestOK member data item, 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 the member data defined in the base class.

At this point, we have finished all coding jobs for our Web Service project. Now let's run our project to test the data query functionality. Click the Start Debugging button to run the project, and the built-in Web interface is displayed, which is shown in Figure 8.17.

Click our Web method GetSQLSelect to open the built-in Web interface for our Web method, which is shown in Figure 8.18. Enter the faculty name Ying Bai into the Value box of the FacultyName as the input parameter, and then click 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 8.19.



Figure 8.17. The running status of the Web Service.



Figure 8.18. The running status of our Web method.

Each returned data is enclosed by a pair of XML tags to indicate or mark up its facility. For example, B78880, which is the queried faculty\_id, is enclosed by the tags <FacultyID>...</FacultyID>, and the name of this tag is defined in our child class SQLSelectResult. Our first Web Service is very successful.

As we mentioned before, a Web Service does not provide any user interface and one needs to develop a user interface oneself to consume a Web Service if one

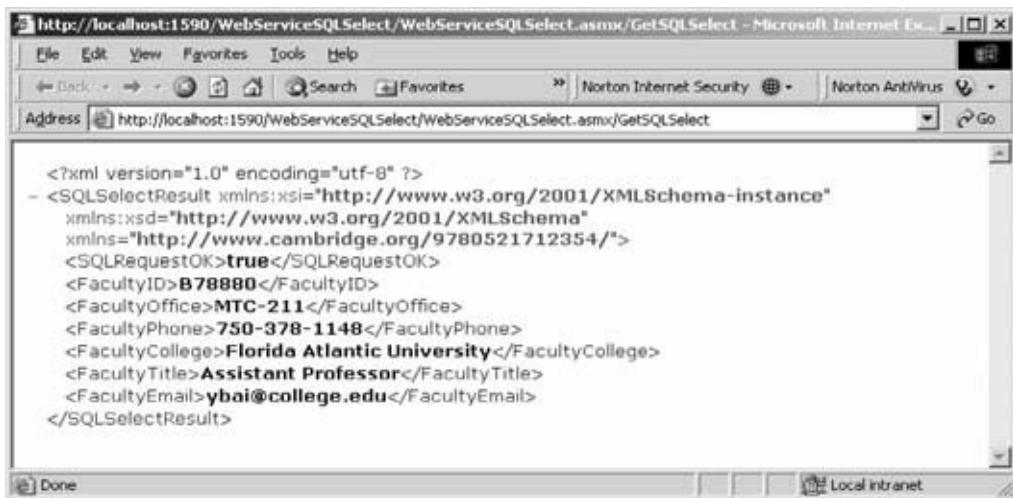


Figure 8.19. The running result of our Web Service project.

wants to display the information obtained from the Web Service. Here, a built-in Web interface is provided by Microsoft to help users display queried information from the Web Services. In real applications, users need to develop user interfaces themselves to perform the data display or other operations.

Click the Close button that is located at the upper right corner of the page to close our Web Service project.

### 8.3.8 Develop a Stored Procedure to Perform the Data Query

An optional and better way to perform the data query via Web Service is to use a stored procedure. The advantage of using this method is that the code can be greatly simplified and most of the query job is performed in the database side, and therefore the query execution speed can be improved. The query efficiency can also be improved, and the query operations can be integrated into a single group or block of code to strengthen the integrity of the query.

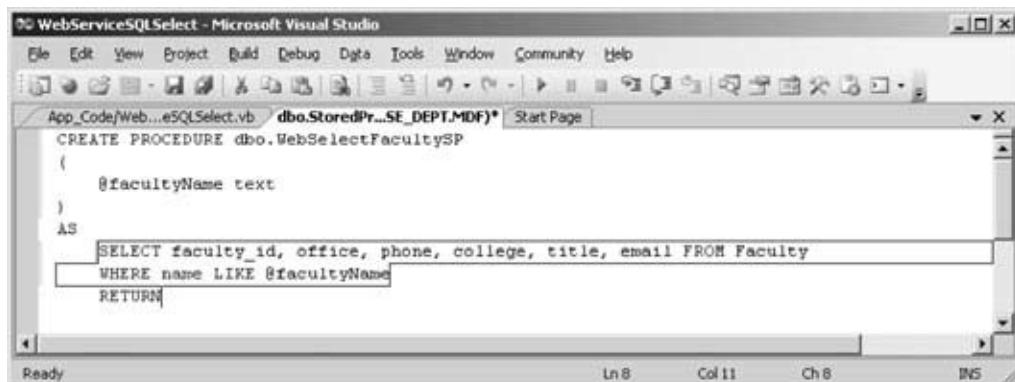
#### 8.3.8.1 Develop the Stored Procedure WebSelectFacultySP

Now let's first develop our stored procedure in the Server Explorer window in Visual Studio.NET environment.

Open Visual Studio.NET, open the Server Explorer window, and click the small plus icon in front of our SQL Server data file CSE\_DEPT.mdf to expand our sample database. Then right-click the Stored Procedures folder and select the item Add New Stored Procedure from the popup menu to open a new stored procedure. Enter the code, shown in Figure 8.20 into this new 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. Right-click this newly created stored procedure from the Server Explorer window and select the Execute item from the popup menu to open the Run Stored Procedure dialog box. Enter the faculty name Ying Bai in the Value box as the input parameter, and click the OK button to run this stored

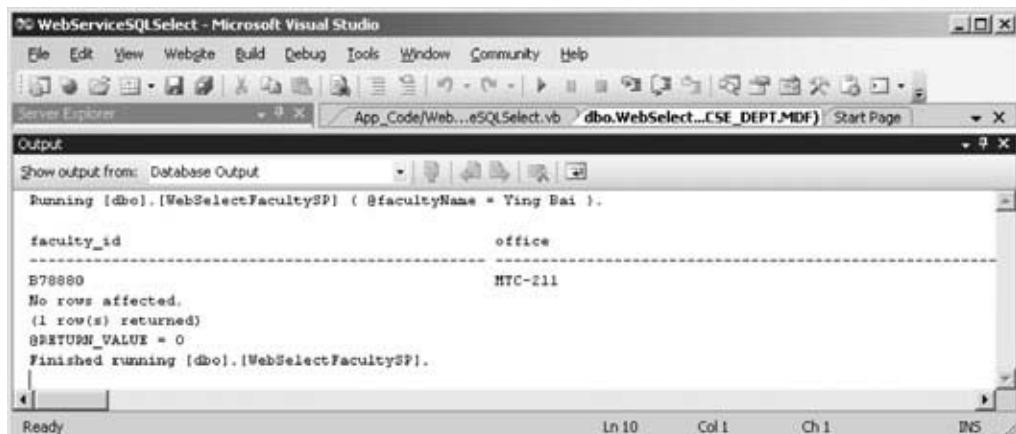


The screenshot shows the Microsoft Visual Studio IDE with the title bar "WebServiceSQLSelect - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, Help. The toolbar has various icons for file operations. The main code editor window displays the following T-SQL code:

```
App_Code\Web...eSQLSelect.vb  dbo.StoredPr...SE_DEPT.MDF* Start Page
CREATE PROCEDURE dbo.WebSelectFacultySP
(
    @facultyName text
)
AS
    SELECT faculty_id, office, phone, college, title, email FROM Faculty
    WHERE name LIKE @facultyName
    RETURN
```

The status bar at the bottom shows "Ready", "Ln 8", "Col 11", "Ch 8", and "INS".

Figure 8.20. The stored procedure.



**Figure 8.21.** The running result of the stored procedure.

procedure. The running result is displayed in the Output window located at the bottom of this running dialog box, which is shown in Figure 8.21.

All six queried columns, which include the faculty\_id, office, phone, college, title, and email in the Faculty table are displayed in this Output window. You need to move the horizontal bar at the bottom to see all six columns. Each column name and its data value are separated with a dashed line.

Our stored procedure is successful.

Now let's handle the coding in our Web Service project to call this stored procedure to perform the data query.

### 8.3.8.2 Add Another Web Method to Call the Stored Procedure

To distinguish it from the first Web method we developed in the previous section, we should add another Web method to perform this data query by calling the stored procedure. To do that, highlight and select the entire code body of our first Web method GetSQLSelect(), including both the method header and the code body; copy this whole body; and paste it at the bottom of our code-behind page (it must be inside our Web Service class). Perform the following modifications to this copied Web method to make it into our second Web method:

- A. Change the Web method's name by attaching two letters, SP, at the end of the original Web method's name, so 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 equal to the name of the stored procedure we developed in the last section. Otherwise a running error may be encountered as the project runs because the project cannot find the desired stored procedure.
- C. Change the CommandType property of the Command object from CommandType.Text to CommandType.StoredProcedure. This is very important since we need to call a stored procedure to perform the data query. We must

WebServiceSQLSelect	▼	GetSQLSelectSP	▼
---------------------	---	----------------	---

```

<WebMethod()>_
Public Function GetSQLSelectSP(ByVal FacultyName As String) As SQLSelectResult
    Dim cmdString As String = ""dbo.WebSelectFacultySP""
    Dim sqlConnection As New SqlConnection
    Dim SQLResult As New SQLSelectResult()
    Dim sqlCommand As New SqlCommand
    Dim sqlReader As SqlDataReader
    SQLResult.SQLRequestOK = True
    sqlConnection = SQLConn()
    If sqlConnection Is Nothing Then
        SQLResult.SQLRequestError = "Database connection is failed"
        ReportError(SQLResult)
        Return Nothing
    End If
    sqlCommand.Connection = sqlConnection
    sqlCommand.CommandType = CommandType.StoredProcedure
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add("@facultyName", SqlDbType.Text).Value = FacultyName
    sqlReader = sqlCommand.ExecuteReader
    If sqlReader.HasRows = True Then
        Call FillFacultyReader(SQLResult, sqlReader)
    Else
        SQLResult.SQLRequestError = "No matched faculty found"
        ReportError(SQLResult)
    End If
    If Not sqlReader Is Nothing Then sqlReader.Close()
    sqlReader = Nothing
    If Not sqlConnection Is Nothing Then sqlConnection.Close()
    sqlConnection = Nothing
    Return SQLResult
End Function

```

**Figure 8.22.** The modified Web method GetSQLSelectSP.

tell the ASP.NET runtime that a stored procedure should be called when the command object is executed.

The modified Web method is shown in Figure 8.22. All modifications are highlighted in bold.

Now you can run the project to test this new Web method. Click the second Web method, GetSQLSelectSP, as the project runs, and click the Invoke button to run it. The same running result as we got from the last project can be obtained.

So you can see how easy it is to develop code to perform a data query by calling a stored procedure in the Web Service project.

The completed Web Service project WebServiceSQLSelect that contains two Web methods can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

Next, we want to discuss how to use a DataSet as a returning object to contain all information we need from running a Web Service project.

### 8.3.9 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 an application class to instantiate a returning object.

Another advantage is that a DataSet can contain multiple records coming from different tables, and we do not need to create multiple member data items in our application class to hold those records. Finally, the size of our code will be greatly reduced when a DataSet is used, especially for a large block of data that is queried via the Web Service project.

To distinguish it from the Web methods we developed before, we can add another new Web method named GetSQLSelectDataSet into our Web Service project. To do that, open our code-behind page if it is not open, highlight and select the entire code body of our first Web method GetSQLSelect, including the method header and code body, and copy it and paste it at the bottom of our page (it must be inside our Web Service class). Perform the modifications shown in Figure 8.23 to this copied Web method to make it our third Web method. The modified parts have been highlighted in bold.

Let's have a closer look at this modified Web method to see how it works.

- The Web method's name is modified by attaching “DataSet” to the end of the original method name; also, the nominal name of the returning object

WebServiceSQLSelect		GetSQLSelectDataSet	
---------------------	--	---------------------	--

```

<WebMethod()>_
Public Function GetSQLSelectDataSet(ByVal FacultyName As String) As DataSet
    Dim cmdString As String = "SELECT faculty_id, office, phone, college, title, email FROM Faculty " + _
        "WHERE name LIKE @facultyName"
    Dim sqlConnection As New SqlConnection
    Dim SQLResult As New SQLSelectResult()
    Dim sqlCommand As New SqlCommand
A Dim FacultyAdapter As New SqlDataAdapter
B Dim dsFaculty As New DataSet
B Dim intResult As Integer
    SQLResult.SQLRequestOK = True
    sqlConnection = SQLConn()
    If sqlConnection Is Nothing Then
        SQLResult.SQLRequestError = "Database connection is failed"
        ReportError(SQLResult)
        Return Nothing
    End If
    sqlCommand.Connection = sqlConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add("@facultyName", SqlDbType.Text).Value = FacultyName
C FacultyAdapter.SelectCommand = sqlCommand
D intResult = FacultyAdapter.Fill(dsFaculty, "Faculty")
E If intResult = 0 Then
        E     SQLResult.SQLRequestError = "No matched faculty found"
        E     ReportError(SQLResult)
E End If
F If FacultyAdapter IsNot Nothing Then FacultyAdapter.Dispose()
G FacultyAdapter = Nothing
    If sqlConnection IsNot Nothing Then sqlConnection.Close()
        sqlConnection = Nothing
    Return dsFaculty
End Function

```

**Figure 8.23.** The modified Web method – GetSQLSelectDataSet.

- is changed to “DataSet,” which means that this Web method will return a DataSet.
- B. Two new data objects, FacultyAdapter and dsFaculty, are created, and these two objects work as a DataAdapter and DataSet, respectively. A local integer variable, intResult, is also created, and it will be used later to hold the returned value from calling the Fill() method of the DataAdapter to perform the data query.
  - 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, to fill the faculty table in the DataSet dsFaculty.
  - D. The Fill() method of the DataAdapter class is executed to fill the Faculty table in our DataSet. This method will return an integer value to indicate whether this calling is successful or not, and this returned value is stored into the local integer variable intResult that will be checked later.
  - E. If the returned value is zero, it means that no row has been retrieved from the Faculty table in our sample database and no row has been filled into our Faculty table in our DataSet dsFaculty, and 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 and filled.
  - G. Finally, the filled DataSet dsFaculty, 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 the Start Debugging button to run the project. Now we have three Web methods available to this Web Service, which is shown in Figure 8.24.

Click the second method, GetSQLSelectDataSet, from the built-in Web interface window, enter the faculty name Ying Bai into the Value box as our desired faculty, and then click the Invoke button to call this Web method to perform the data query. The running result shown in Figure 8.25 is returned.

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 8.25.

At this point, we have finished all developing jobs in our Web Service project on the server side. Next, we want to develop some professional Windows-based or Web-based applications with beautiful graphical user interfaces to use or to



Figure 8.24. Three Web methods in built-in Web interface window.

consume the Web Service application we developed in the previous sections. Those Windows-based or Web-based applications can be considered Web Service clients.

### 8.3.10 Build Windows-based Web Service Clients to Consume the Web Services

To use or consume a Web Service, we first 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.

#### 8.3.10.1 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 2005 will

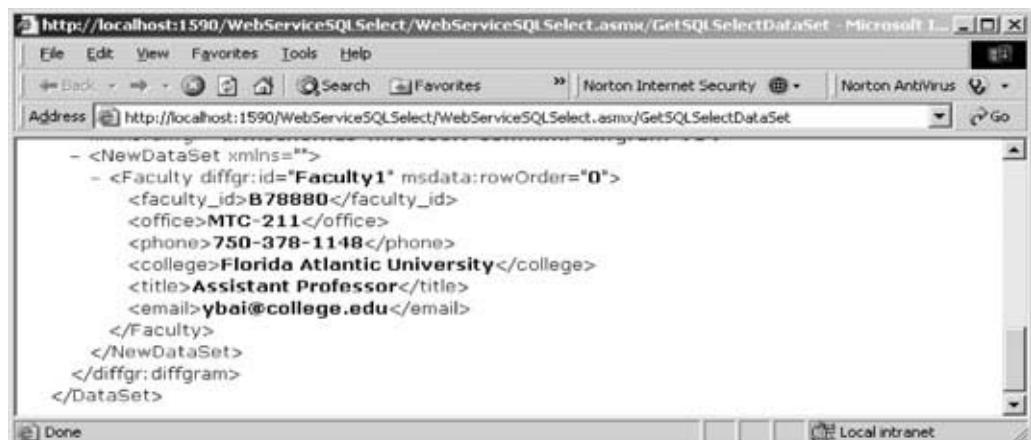


Figure 8.25. The running result of the Web method GetSQLSelectDataSet.

try to find all Web Services available to our applications. The following operations will be performed by Visual Studio.NET 2005 during this process:

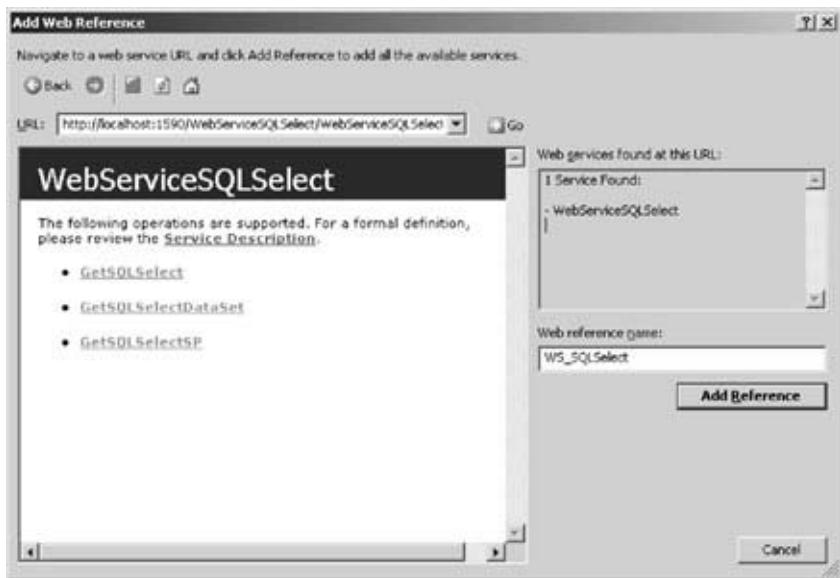
1. When looking for Web Services from the local computers, Visual Studio.NET 2005 will check all files that include a Web Service main page with an .asmx extension, a WSDL file with a .wsdl extension, or a Discovery file with a .disco extension.
2. When searching for Web Services from the Internet, Visual Studio.NET 2005 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 of each Web Service and its Web methods. Once you have 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 are referenced to the client project, a group of necessary files or documents are also created by Visual Studio.NET 2005. These files include the following:
  - 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. These Web Services are obtained through a process called XML Web Services Discovery.
  - c. A WSDL file that provides a detailed description and definition of those Web Services in an abstract manner.

To add a Web reference to our client project, we first need to create a client project. Now let's create a Windows-based application to consume the Web Service we developed in the previous sections. Then, we can add a Web reference to our new project.

Open Visual Studio.NET 2005 and create a new Windows-based project, and name this project WinClientSQLSelect.

In Visual Studio.NET, right-click our new project WinClientSQLSelect from the Solution Explorer window, and select the item Add Web Reference from the popup menu to open the Add Web Reference dialog box, which is shown in Figure 8.26.

There are two ways we can use to select the desired Web Service and add it as a reference to our client project; one way is to use the Browser provided by the Visual Studio.NET 2005 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 dialog box. The second way needs you to first run the Web Service and then copy its URL and paste it to the URL box in this dialog if you did not deploy



**Figure 8.26.** The Add Web Reference dialog box.

that Web Service to IIS. If you did deploy that Web Service, you can directly type that URL into the URL box in this dialog box.

Because we developed our Web Service using the File System on our local computer and have not deployed our Web Service to IIS, we must use the second way to find our Web Service. Open our Web Service project and click the Start Debugging button to run it. Copy the URL from the Address bar and then switch back to our client project WinClientSQLSelect, and paste that URL into the URL box in the Add Web Reference dialog. Click the Go button to allow Visual Studio.NET 2005 to search for it.

When the Web Service is found, the name of our Web Service is displayed in the right pane, as shown in Figure 8.26.

Alternately, you can change the name of this Web reference from localhost to any meaningful name, such as WS\_SQLSelect in our case. Click the Add Reference button to add this Web Service as a reference to our new client project. Click the Close button from our Web Service's built-in Web interface window to terminate our Web Service project.

Immediately you can find that the Web Service WS\_SQLSelect, which is under the folder Web References, has been added into the Solution Explorer window in our project. This reference is the so-called Web Service proxy class.

Next, let's develop the graphical user interface by adding useful controls to interface with our Web Service and display the queried information.

### **8.3.10.2 Develop the Graphical 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 graphical user interface located in the project SQLUpdateDeleteRTOBJECT we developed in Chapter 6. Open that project from the folder **DBProjects\Chapter 6** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). Then open the Faculty form window, select all controls from that form by going to Edit|Select All, and go to Edit|Copy 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 the controls we copied from the Faculty form in the project SQLUpdateDeleteRTOBJECT. The only modification to these controls is to remove the Name text box and the associated label control since we do not need this piece of information in this project. Your finished graphical user interface is shown in Figure 8.27.

The Method combo box control is used to select from three different methods developed in our Web Service project to get our desired information:

1. A method that uses an object to return our queried information.
2. A method that uses a stored procedure to return our queried information.
3. A method that uses a DataSet to return our queried information.

The Faculty Name combo box control is used to select the desired faculty 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 functionality of this project is as follows: When the project runs and 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 connect to our Web Service based on the Web reference we provided, and will call the selected

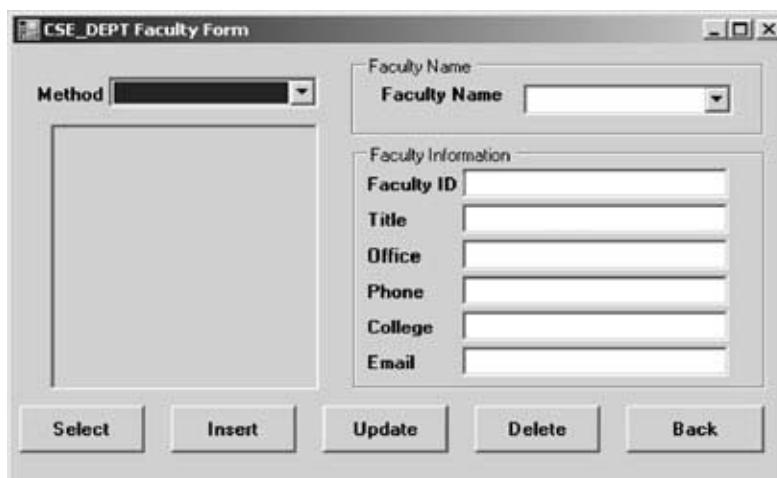


Figure 8.27. The graphical user interface.

method based on the method chosen from the Method combo box control to perform the data query to retrieve the desired faculty information from our sample database and display it in this graphical user interface.

Now let's take care of the coding for this project to connect to our Web Service using the Web reference we developed in the last section.

### **8.3.10.3 Develop the Code to Consume the Web Service**

The coding job can be divided into four parts:

1. The coding for the Form\_Load event procedure to initialize the Method combo box control and the Faculty Name combo box control. The first initialization will set up three Web methods that can be selected by the user to perform the data query from the Web Service. The second one is to set up a default list of faculty members that can be selected by the user to perform the associated faculty information query.
2. The coding for the Back button's click event procedure to terminate the project.
3. The coding for the Select button's click event procedure.
4. The coding for other subroutines and procedures, such as the subroutines ShowFaculty(), ProcessObject(), FillFacultyObject(), and FillFacultyDataSet().

The main coding job is performed inside the Select button's click event procedure. As we discussed, when this button is clicked by the user, a connection to our Web Service needs to be established using the Web reference we set up in the previous section. So we first need 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 invoking the Web methods. The protocol to instantiate a proxy class is as follows:

---

```
Dim newInstanceName As New WebReferenceName.WebServiceName
```

---

After this new instance is created, then a connection between our client project and our Web Service can be set up by using this instance. The pseudo code for this event procedure is 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 can be used as a mapping of the real object SQLSelectResult developed in the Web Service. We can easily access the Web method to perform our data query and pick up the result from that returning object by assigning it to the mapping object.
- C. A new DataSet object is created and is used to call the Web method that returns a DataSet.
- D. Based on the method selected by the user from the Method combo box control, different Web methods can be called to perform the data query.

- E. The returned data that is 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 graphical user interface.
- 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 text boxes in the Client form with the information picked up from the DataSet.

#### 8.3.10.3.1 Develop the Code for the Form\_Load Event Procedure

Now let's begin to develop the code 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 the Load item from the Method Name combo box from the code window of the Faculty Form. Enter the code shown in Figure 8.28 into this event procedure.

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

- A. Eight default faculty members are added into the ComboName combo box control using the Add() method, and this will allow users to select one desired faculty member from this combo box control to perform the data query as the project runs. The default faculty member is made to be the first one by setting the SelectedIndex property to zero.
- B. Three Web methods are also added into another combo box control, ComboMethod, to allow users to select one of them to perform the associated data query via our Web Service. The first one is selected as the default method.

The coding for the Back button's click event procedure is 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 this button as the project runs. The Close() method tells Visual Basic.NET to terminate the current project.

The screenshot shows the Visual Studio code editor with the title bar '(FacultyForm Events)' and the method name 'Load' selected. The code is divided into two sections labeled A and B.

```
Private Sub FacultyForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    A
    ComboName.Items.Add("Ying Bai")
    ComboName.Items.Add("Satisch Bhalla")
    ComboName.Items.Add("Black Anderson")
    ComboName.Items.Add("Steve Johnson")
    ComboName.Items.Add("Jenney King")
    ComboName.Items.Add("Alice Brown")
    ComboName.Items.Add("Debby Angles")
    ComboName.Items.Add("Jeff Henry")
    ComboName.SelectedIndex = 0
    B
    ComboMethod.Items.Add("Object Method")
    ComboMethod.Items.Add("Stored Procedure Method")
    ComboMethod.Items.Add("DataSet Method")
    ComboMethod.SelectedIndex = 0
End Sub
```

Figure 8.28. The code for the Form\_Load event procedure.

A	cmdSelect	Click
	Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click	
A	Dim wsSQLSelect As New WS_SQLSelect.WebServiceSQLSelect	
	Dim wsSQLResult As New WS_SQLSelect.SQLSelectResult	
	Dim wsDataSet As New DataSet	
B	If ComboMethod.Text = "Object Method" Then	
	Try	
	wsSQLResult = wsSQLSelect.GetSQLSelect(ComboName.Text)	
C	Catch err As Exception	
	MsgBox("Web service is wrong: " & err.Message)	
	End Try	
D	Call ProcessObject(wsSQLResult)	
E	ElseIf ComboMethod.Text = "Stored Procedure Method" Then	
	Try	
F	wsSQLResult = wsSQLSelect.GetSQLSelectSP(ComboName.Text)	
	Catch err As Exception	
G	MsgBox("Web service is wrong: " & err.Message)	
H	End Try	
	Call ProcessObject(wsSQLResult)	
I	Else	
	Try	
J	wsDataSet = wsSQLSelect.GetSQLSelectDataSet(ComboName.Text)	
	Catch err As Exception	
	MsgBox("Web service is wrong: " & err.Message)	
	End Try	
	Call FillFacultyDataSet(wsDataSet)	
	End If	
	End Sub	

**Figure 8.29.** The code for the Select button's click event procedure.

### 8.3.10.3.2 Develop the Code for the Select Button's Click Event Procedure

Open the Select button's click event procedure and enter the code shown in Figure 8.29 into this event procedure.

Let's have a closer look at this piece of code 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 of the real object SQLSelectResult developed in the Web Service so that we can easily access the Web method to perform our data query and pick up the result from the returned object by assigning it to this mapping object later, and 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 ComboMethod control, a Try...Catch block is used to call the associated Web method GetSQLSelect(), which is developed in our Web Service, through the instantiated reference class to perform the data query. The selected faculty name that is located in the Text property of the ComboName combo box control is passed as a parameter for this calling.
- C. The Catch statement is used to collect any possible exceptions if any errors occurred for this calling, and the error message is displayed using a message box.

- D. If no exception occurred, the subroutine ProcessObject() is executed to pick up all pieces of retrieved information from the returned object and display them in this form window.
- E. If the user selected the Stored Procedure Method, the associated Web method GetSQLSelectSP(), which is developed in our Web Service, is called via the instance of the Web-referenced class to perform the data query.
- F. The Catch statement is used to collect any possible exceptions if any errors occurred for this calling, and the error message is displayed using a message box.
- G. Similarly, the subroutine ProcessObject() is executed to pick up all pieces of retrieved information from the returned object and display them in this form window.
- H. If the user 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.
- I. The Catch statement is used to collect any possible exceptions if any errors 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 display them in this form window.

#### 8.3.10.3.3 Develop the Code for Other Subroutines

The code for the subroutines ProcessObject() and FillFacultyObject() is shown in Figure 8.30.

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 functionality of this code is as follows:

- A. If the member data SQLRequestOK that is stored in the instance of our child class or returned object is set to True, which means that our Web method is

	<b>FacultyForm</b>	<b>ProcessObject</b>	
<b>A</b>	<pre>Private Sub ProcessObject(ByRef wsResult As WS_SQLSelect.SQLSelectResult)     If wsResult.SQLRequestOK = True Then         Call FillFacultyObject(wsResult)     Else         MsgBox("Faculty information cannot be retrieved: " &amp; wsResult.SQLRequestError)     End If End Sub</pre>		
<b>B</b>			
<b>C</b>	<pre>Private Sub FillFacultyObject(ByRef sqlResult As WS_SQLSelect.SQLSelectResult)     txtID.Text = sqlResult.FacultyID     txtOffice.Text = sqlResult.FacultyOffice     txtPhone.Text = sqlResult.FacultyPhone     txtCollege.Text = sqlResult.FacultyCollege     txtTitle.Text = sqlResult.FacultyTitle     txtEmail.Text = sqlResult.FacultyEmail End Sub</pre>		

Figure 8.30. The code for two subroutines.

<b>FacultyForm</b>		<b>FillFacultyDataSet</b>	
<pre> Private Sub FillFacultyDataSet(ByRef ds As DataSet)     Dim FacultyTable As New DataTable     Dim FacultyRow As DataRow     FacultyTable = ds.Tables("Faculty")     FacultyRow = FacultyTable.Rows(0)          'only one rwo in the Faculty table     txtID.Text = FacultyRow("faculty_id").ToString     txtOffice.Text = FacultyRow("office").ToString     txtPhone.Text = FacultyRow("phone").ToString     txtCollege.Text = FacultyRow("college").ToString     txtTitle.Text = FacultyRow("title").ToString     txtEmail.Text = FacultyRow("email").ToString End Sub </pre>			

**Figure 8.31.** The code for the subroutine FillFacultyDataSet.

executed successfully, the subroutine FillFacultyObject() is called and executed with the returned object that contains our requested faculty information as an argument to pick up each piece of information from that returned object and display it in this form window.

- A. Otherwise, some exceptions occur and a warning message is displayed with a message box.
- B. 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 text boxes in this form to be displayed.

The code for the subroutine FillFacultyDataSet() is shown in Figure 8.31. The argument passed into this subroutine is an instance of DataSet we created in the Select button's click event procedure.

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

- A. Two data objects, FacultyTable, which is a new object of the DataTable class, and FacultyRow, which is a new instance of the DataRow class, are created first since we need to use these two objects to access the DataSet to pick up all requested faculty information later.
- B. The returned Faculty table that is embedded in our returned DataSet is assigned to our 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 the string "Faculty." For a typed DataSet, you can directly use the table name to access the desired table without needing any string.
- C. Since we only request one record or one row from the Faculty table, the returned Faculty table contains only one row of information, which is located at the top row with an index of zero. This one row of information is assigned to our FacultyRow object we created above.
- D. We can access each column of the returned row of 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 (with 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 text box and will be displayed there.

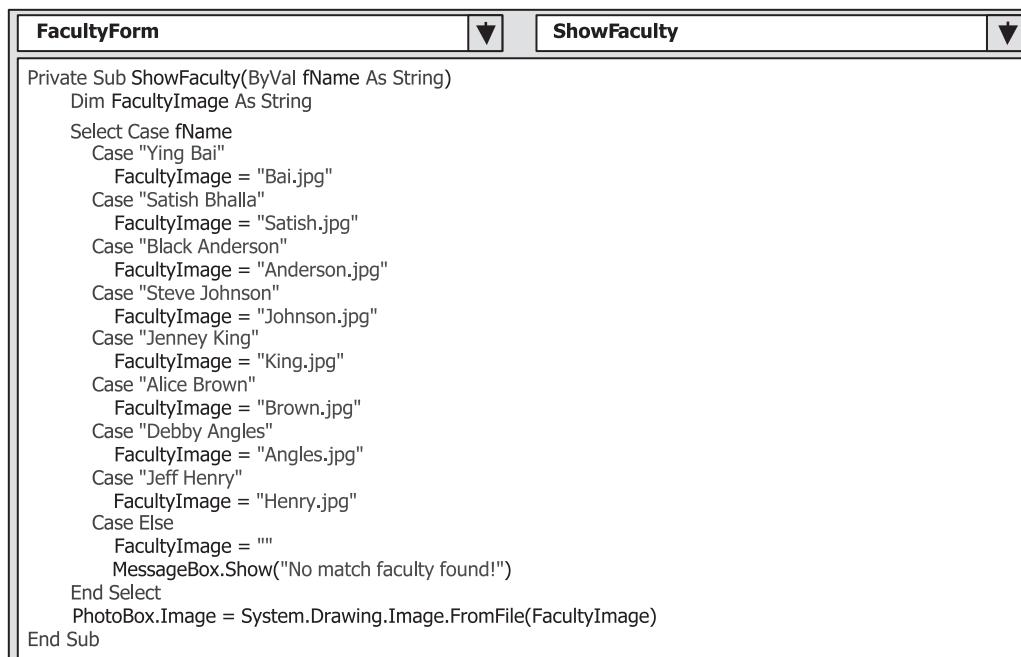
At this point we have almost finished the coding for this Windows-based Web Service client project. We have one more step to go to complete this client project, which is to add a subroutine `ShowFaculty()` to display the requested faculty photo in the image box in this form.

#### 8.3.10.3.4 Develop a Subroutine `ShowFaculty` to Display the Faculty Image

All default faculty images can be found from most of the projects we developed in previous chapters. To display the requested faculty image in the form, the following jobs need to be performed:

1. Copy all default faculty photos from the previous projects that are located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354), for example, from the project **SQLWeb-Select** in the folder **DBProjects\Chapter 7**, and paste them into our current client project folder, that is, into the **Debug** folder that is under our project's bin folder. For example, one needs to paste all faculty photo files into the folder **C:\Book5\Chapter 8\WinClientSQLSelect\bin\Debug**.
2. Develop the code to perform this faculty image displaying.

Now let's develop the code for this subroutine. Open the code window from our client project, and type the code shown in Figure 8.32 into this code window to create our subroutine `ShowFaculty()`.



```

FacultyForm
ShowFaculty

Private Sub ShowFaculty(ByVal fName As String)
    Dim FacultyImage As String
    Select Case fName
        Case "Ying Bai"
            FacultyImage = "Bai.jpg"
        Case "Satish Bhalla"
            FacultyImage = "Satish.jpg"
        Case "Black Anderson"
            FacultyImage = "Anderson.jpg"
        Case "Steve Johnson"
            FacultyImage = "Johnson.jpg"
        Case "Jenney King"
            FacultyImage = "King.jpg"
        Case "Alice Brown"
            FacultyImage = "Brown.jpg"
        Case "Debby Angles"
            FacultyImage = "Angles.jpg"
        Case "Jeff Henry"
            FacultyImage = "Henry.jpg"
        Case Else
            FacultyImage = ""
            MessageBox.Show("No match faculty found!")
    End Select
    PhotoBox.Image = System.Drawing.Image.FromFile(FacultyImage)
End Sub

```

Figure 8.32. The code for the subroutine `ShowFaculty`.

The coding for this subroutine is 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 method Image.FromFile() and is then displayed in the PhotoBox control in this form window. A warning message will be displayed if no matched faculty image is found.

To call this subroutine to display the selected faculty image, add one more statement, which 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.

But wait a moment! There is one important issue you need to note before you can run this project, which is that you must first run our Web Service project WebServiceSQLSelect to make our Web Service available to all clients, and then you can run our client project to access and interface with our Web Service to perform the data query. Otherwise, you may encounter some running exceptions, such as that the Web Service or remote computer cannot be found, as your client project runs.

Once our Web Service project runs, you can stop it and access it using our client project without any problem. But one point you need to remember is that our Web Service project must be kept in the open status (even if it does not run) in order to allow our client project to access it and interface with it. An exception will be encountered if you close our Web Service when you try to access it using our client project.

Make sure that our Web Service has been run and is in the open status, which can be identified by a small Web Service running icon in the task bar at the bottom of your screen. Start our client project by clicking the Start Debugging button from the project WinClientSQLSelect. The running status is shown in Figure 8.33.

Keep the default Web method and the faculty name Ying Bai selected, and click the Select button to call the associated Web method to retrieve the desired faculty information. The returned faculty information is displayed in the associated text boxes along with the faculty photo, as shown in Figure 8.33.

You can try to select other Web methods such as 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 the Back button to terminate our project.

The completed Windows-based Web Service client project WinClientSQLSelect is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**. You need to load both this client project and our Web Service project from this folder and install them on your computer if you want to run and test this client



Figure 8.33. The running status of our client project.

project. Also, you must run our Web Service project first 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.

### 8.3.11 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 whether this Web Service is consumed by a Windows-based or a Web-based client application.

To save time and space, we can modify the existing ASP.NET Web application SQLWebSelect we developed in Chapter 7 to make it into our new Web-based Web Service client project, WebClientSQLSelect. We could copy and rename that entire project as our new Web-based client project, but here we prefer to create a new ASP.NET Web site project and only copy and modify the Faculty page.

This modification involves the following steps:

1. Create a new ASP.NET Web site project, WebClientSQLSelect, and add the existing Web page Faculty.aspx from the project SQLWebSelect into our new project.
2. Add a Web Service reference to our new project, and modify the Web form window of the Faculty.aspx to meet our data query requirements.

3. Modify the code in the related event procedures of the Faculty.aspx.vb file to call the associated Web method to perform our data query. The code modifications include the following sections:
  - a. Modify the code 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 to those we developed in the last Windows-based Web Service client project, WinClientSQLSelect, and one can copy and paste them into our new project. The only modification is for the subroutine ProcessObject().
  - d. Modify the code in the Back button's click event procedure.

Now let's start with the first step listed above.

### **8.3.11.1 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 Web site project. Enter “C:\Chapter 8\WebClientSQLSelect” into the name box that is next to the Location box, and click the OK button to create this new project.

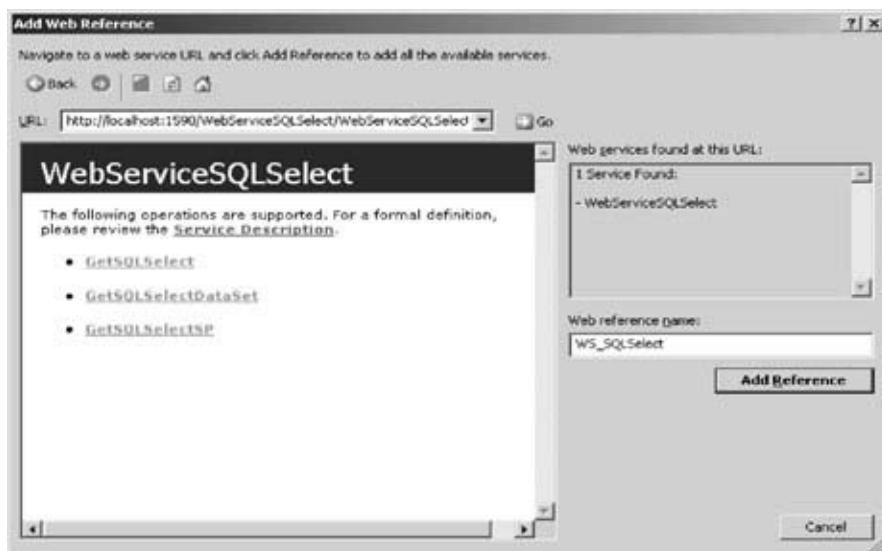
In the new project window, right-click our new project icon WebClientSQLSelect from the Solution Explorer window, and select the item Add Existing Item from the popup menu to open the Add Existing Item dialog box. Browse to our Web project SQLWebSelect, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 7**, select it, and then click the Open button to open all existing items for this Web site project.

Select the items Faculty.aspx and Faculty.aspx.vb from the list, and click the Add button to add these two items into our new Web site project.

One issue we need to emphasize is that we must add all faculty photo files into our new project before we can continue to the next step. In this way, the selected faculty photo can be displayed as that faculty member's information is queried. This step is highly recommended since we need those faculty photo files later when we display each of them for the selected faculty member as we perform the data query. To do this, right-click our new project icon WebClientSQLSelect from the Solution Explorer window, and select Add Existing Item from the popup menu. Browse to our project SQLWebSelect, select it, and then click the Open button to open all existing items for this Web site project. Select all files that have the extension .jpg, and click the Add button to add them into our new project.

### **8.3.11.2 Add a Web Service Reference and Modify the Web Form Window**

Just as we did in the last project, to add a Web reference of our Web Service to this new Web site project, right-click our new project icon from the Solution Explorer window and select the item Add Web Reference from the popup menu. Now open our Web Service project WebServiceSQLSelect and click the Start Debugging button to run it. As the project runs, copy the URL from the Address box and paste it into the URL box in our Add Web Reference dialog box. Then click the green Go button to add this Web Service as a reference to our client project. You can modify



**Figure 8.34.** Adding a Web reference.

this Web reference name to any name you want. In this application, we prefer to change it to WS\_SQLSelect. Your finished Add Web reference dialog box should match the one shown in Figure 8.34.

Click the Add Reference button to finish this Web reference adding process. Immediately you will 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

The modifications to the Web form of the Faculty.aspx include two steps. The first one is to remove the Name text box and its associated label since we do not need this control in this application. Delete these two controls from the Web form window, and use the Backspace key to remove the space and align the controls below those removed items. The second one is to add one more DropDownList control and the associated label at the top of the faculty image box control. Name this DropDownList as ComboMethod and label it Method. This DropDownList control is used to store the three Web methods developed in our Web Service and allow users to select one of them to perform the associated data query as the project runs. Your modified Faculty.aspx Web form window should match the one shown in Figure 8.35.

Go to the File|Save All menu item to save these modifications.

### 8.3.11.3 Modify the Code for the Related Event Procedures

The first modification is to change the code in the Page\_Load event procedure and some global variables.



Figure 8.35. The modified Faculty Web page.

### 8.3.11.3.1 Modify the Code 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 text box 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 code to add and display three Web methods in the ComboMethod combo box control.

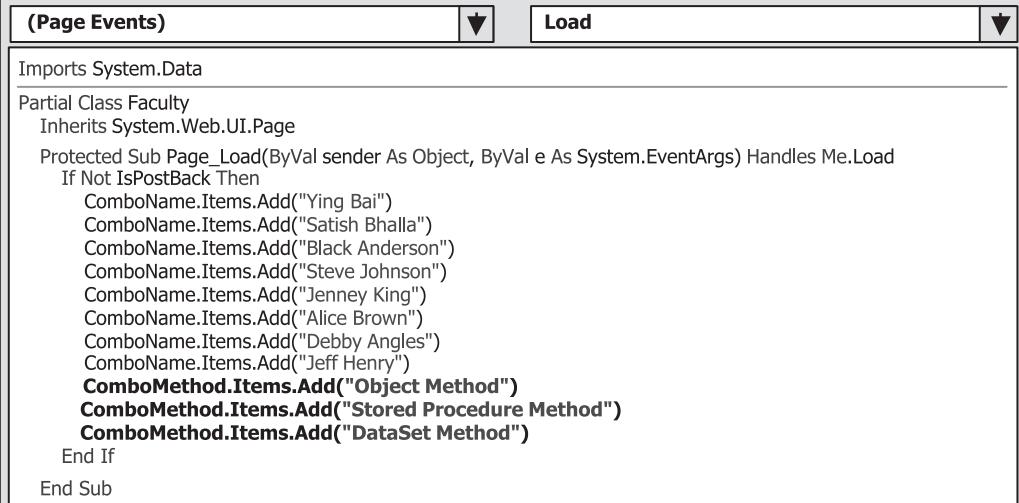
Your finished code for the Page\_Load event procedure should match that shown in Figure 8.36. The modifications are highlighted in bold.

The next modification is to change the code inside the Select button's click event procedure.

### 8.3.11.3.2 Modify the Code in the Select Button Event Procedure

Replace the code in this event procedure with the following modified code:

- 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 that is used to store the possible error message.
- C. Call the subroutine ShowFaculty(), whose code is shown below, to display the selected faculty photo.



The screenshot shows a code editor window with the title bar '(Page Events)' and a 'Load' button. The code in the editor is as follows:

```

Imports System.Data
Partial Class Faculty
    Inherits System.Web.UI.Page
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        If Not IsPostBack Then
            ComboName.Items.Add("Ying Bai")
            ComboName.Items.Add("Satis Bhalla")
            ComboName.Items.Add("Black Anderson")
            ComboName.Items.Add("Steve Johnson")
            ComboName.Items.Add("Jenney King")
            ComboName.Items.Add("Alice Brown")
            ComboName.Items.Add("Debby Angles")
            ComboName.Items.Add("Jeff Henry")
ComboMethod.Items.Add("Object Method")
ComboMethod.Items.Add("Stored Procedure Method")
ComboMethod.Items.Add("DataSet Method")
        End If
    End Sub

```

Figure 8.36. The modified Page\_Load event procedure.

- D. If the user selects the Web Object method, a Try...Catch block is used to call the 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 text boxes in our Web page to display it.
- F. If the user selects 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 an error occurred for this calling, and the error message is displayed using the Write() method of the Response object of the server. Similarly, the subroutine ProcessObject() is executed to pick up all pieces of retrieved information from the returned object and display them in this form page.
- G. If the user selects 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 an 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 display them in this form page.

All these modification steps are shown in Figure 8.37.

```

cmdSelect Click
Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim wsSQLSelect As New WS_SQLSelect.WebServiceSQLSelect
    Dim wsSQLResult As New WS_SQLSelect.SQLSelectResult
    Dim wsDataSet As New DataSet
    Dim errMsg As String
    Call ShowFaculty(ComboName.Text)
    If ComboMethod.Text = "Object Method" Then
        Try
            wsSQLResult = wsSQLSelect.GetSQLSelect(ComboName.Text)
        Catch err As Exception
            errMsg = "Web service is wrong: " & err.Message
            Response.Write("<script>alert('" + errMsg + "')</script>")
        End Try
        ProcessObject(wsSQLResult)
    ElseIf ComboMethod.Text = "Stored Procedure Method" Then
        Try
            wsSQLResult = wsSQLSelect.GetSQLSelectSP(ComboName.Text)
        Catch err As Exception
            errMsg = "Web service is wrong: " & err.Message
            Response.Write("<script>alert('" + errMsg + "')</script>")
        End Try
        ProcessObject(wsSQLResult)
    Else
        Try
            wsDataSet = wsSQLSelect.GetSQLSelectDataSet(ComboName.Text)
        Catch err As Exception
            errMsg = "Web service is wrong: " & err.Message
            Response.Write("<script>alert('" + errMsg + "')</script>")
        End Try
        Call FillFacultyDataSet(wsDataSet)
    End If
End Sub

```

Figure 8.37. The modified code for the Select button's click event procedure.

### 8.3.11.3.3 Add Three User-Defined Subroutines

We need to add three user-defined subroutines, `ProcessObject()`, `FillFacultyObject()`, and `FillFacultyDataSet()`, into this project. The code for these three subroutines is basically identical to that developed in the last Windows-based Web Service client project, `WinClientSQLSelect`, and one can copy and paste it into our new project with a slight modification.

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 needed is for the `MsgBox()` method in the subroutine `ProcessObject()`. In the Web site 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 for this subroutine to hold the possible error message. The modified code for this subroutine should match that shown in Figure 8.38. The modifications are highlighted in bold.

There is no modification needed for the other two subroutines, `FillFacultyObject()` and `FillFacultyDataSet()`.

<b>Faculty</b>		<b>ProcessObject</b>	
<pre>Private Sub ProcessObject(ByRef wsResult As WS_SQLSelect.SQLSelectResult)     Dim errmsg As String     errmsg = "Faculty information cannot be retrieved: " &amp; wsResult.SQLRequestError     If wsResult.SQLRequestOK = True Then         Call FillFacultyObject(wsResult)     Else         Response.Write("&lt;script&gt;alert('" + errmsg + "')&lt;/script&gt;")     End If End Sub</pre>			

Figure 8.38. The modified code for the subroutine ProcessObject.

#### 8.3.11.3.4 Modify the Code 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 event procedure should match the one shown in Figure 8.39. The modifications are highlighted in bold.

Now it is time for us to run our Web-based Web Service client project to test the functionality of our data query and our Web Service. But before we 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. To confirm that, right-click the Faculty.aspx page from the Solution Explorer window and select the item Set As Start Page from the popup menu.
2. Make sure that our Web Service WebServiceSQLSelect has been run at least once and that the Web Service status is open, which can be identified by a small white icon located in the task bar at the bottom of the screen.

Now click the Start Debugging button to run our project. The Faculty page is displayed, as shown in Figure 8.40.

Keeping the default Web method in the ComboMethod combo box control and the faculty name in the ComboName combo box control unchanged, click the Select button to call the associated Web method developed in our Web Service to retrieve the selected faculty member's information from our sample database via the Web server. The query result is shown in Figure 8.40.

You can try to select different Web methods with different faculty members to test this project. Our Web-based Web Service client project is very successful.

The completed Web-based Web Service client project WebClientSQLSelect is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

<b>cmdBack</b>		<b>Click</b>	
<pre>Protected Sub cmdBack_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdBack.Click     Response.Write("&lt;script&gt;window.close()&lt;/script&gt;") End Sub</pre>			

Figure 8.39. The modified code for the Back button event procedure.

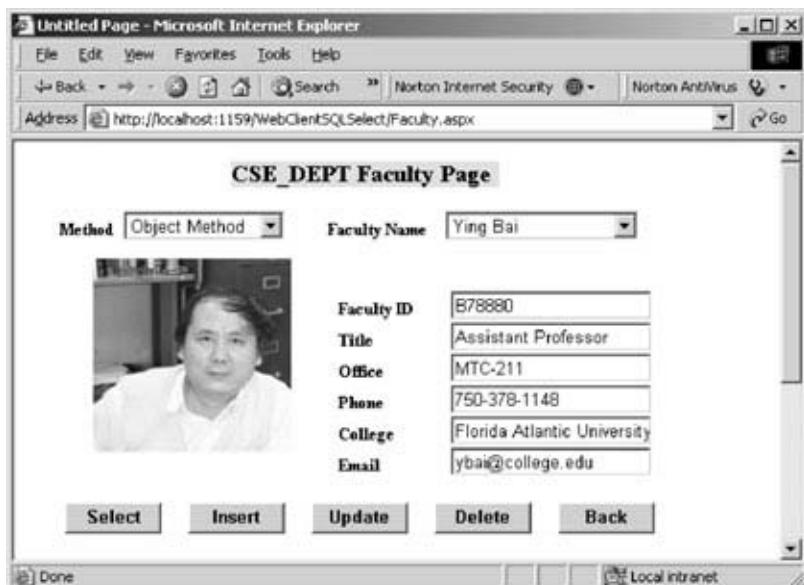


Figure 8.40. The running status of our Web-based client project.

### 8.3.12 Deploy the Completed Web Service to Production Servers

When we finish 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 could have discussed this topic in the earlier section when we finished developing our Web Service project. The reason we delay this discussion until here is to show readers that we do not have to perform this Web Service deployment if we are running our Web Service and accessing it using a client project in our local computer (development server). However, you must deploy your Web Service to IIS if you want to run it in a formal Web server (production server).

Basically, we have two ways to do this deployment; one way is to simply copy our Web Service files to a server running 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 the 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 Internet Information Services (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 the Administrative Tools icon from the Control Panel. In the dialog box, double-click the icon Computer Management, then expand the Services and Applications item from the dialog box and continue to expand the item Internet Information Services. Three items are listed under this icon: Default FTP Site, Default Web Site, and Default SMTP Virtual Server.
3. Right-click the second item, Default Web Site, and select the item New|Virtual Directory from the popup menu to open the Creation Wizard. Click Next to go to the next step.
4. Enter WSSQLSelect into the Alias box as the name for this virtual directory, and then click the Next button to continue.
5. In the next window, click the Browse button to find the folder we created at step 1, which is WSSQLSelect. Click OK and then Next to go to the next step.
6. Keep all default settings in the window and click the Next button to continue.
7. Click the Finish button to complete this process.

Our virtual directory is created, but the story is not finished. As you know, there is no Default.asmx page in our Web Service project. So in order to allow the Web server to find our starting page, we need to modify the default page for this virtual directory. Follow the steps below to finish this modification:

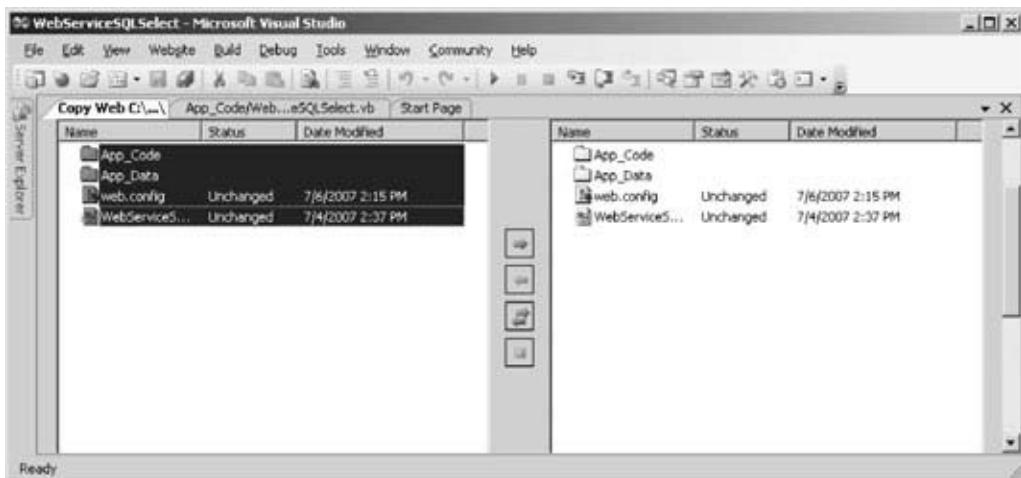
1. Right-click our newly created virtual directory WSSQLSelect from the Computer Management window, and select the Properties item to open the Property window.
2. Click the Documents tab from the Properties window, and remove all items from the list box by selecting them and clicking the Remove button.
3. Enter our starting page, WebServiceSQLSelect.asmx, into the Default Document Name box as our default page, then click the OK button.
4. Click the Apply button and then the OK button to close this window.

After our virtual directory is set up, we can deploy our Web Service by either copying files to this virtual directory or performing a precompile process. First, let's do that by copying all files to the virtual directory since it is relatively simple.

### 8.3.12.1 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 the Website|Copy Web Site menu item to open the Copy Web Site window, which is shown in Figure 8.41.
3. Click the Connect button that is located to the right of the Connections text box.
4. In the opened window, click 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 the Open button.
5. Select all files and folders from our Web Service project, and click the right-arrow button to copy all files to our virtual directory, as shown in Figure 8.41.



**Figure 8.41.** Copying Web Service files to the virtual directory.

6. Now go to the File|Open Web Site menu item to open the Open Web Site dialog window. Click the Local IIS icon from the left pane and select our virtual directory WSSQLSelect, and then click the Open button. Click Yes in the message box to allow our site to be configured to use ASP.NET 2.0 if this message box is displayed.
7. On the opened Web Service project, open the web.config file and change the compilation debug attribute from true to false.
8. Rebuild our Web Service project and run it again, and 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.

### **8.3.12.2 Publish a 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 8.3.12 to create this virtual directory if you have not already done so.

Now open our Web Service project if it is not open. Go to the Build|Publish Web Site menu item to open the Publish Web Site dialog box. 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 the OK button to begin the publishing process.

As the publishing process is completed, the processing and the output of this publishing process are displayed in the Output window. To see what happens in this process, open this Output window by going to View|Other Windows|Output. A sample processing result is shown in Figure 8.42.

Another way to check this publishing result is to open the virtual directory we created for this published Web Service to inspect the associated compiled files. To

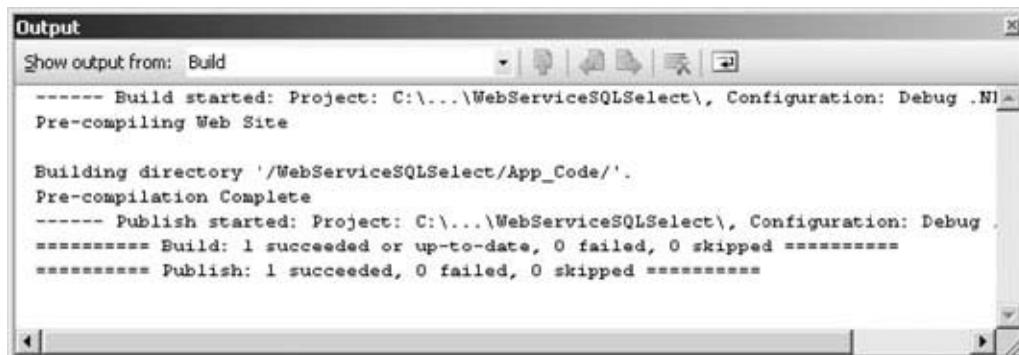


Figure 8.42. The processing result of the Web Service publishing.

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 Internet Explorer (IE) and type our virtual directory <http://localhost/WSSQLSelectCompile> into the Address box as the URL to try to open our service page.

At this point, we have finished discussing how to create and consume a Web Service using Windows-based and Web-based Web Service client projects. In the following sections, we will expand these discussions to perform data insertion, data updating, and data deleting actions in the database through the Web Services.

## **8.4 BUILD AN ASP.NET WEB SERVICE PROJECT TO INSERT DATA INTO AN SQL SERVER DATABASE**

In this section we want to discuss how to insert data into our sample database through a Web Service developed in Visual Studio.NET. The data table we want to use for this data action is the Course table. In other words, we want to insert a new course for the selected faculty member into the Course table via a Web Service we will develop in this section.

To save time and space, we can copy and modify the existing Web Service project WebServiceSQLSelect we developed in the previous section to make it into our new Web Service project, WebServiceSQLInsert.

### **8.4.1 Modify an Existing Web Service Project**

First, let's create a new folder, such as **Chapter 8**, in our root directory using the Windows Explorer, and then copy the WebServiceSQLSelect project from the [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**, and paste it into our newly created folder **C:\Chapter 8**. Rename it WebServiceSQLInsert, and perform the following modifications to this project:

1. Change the main Web Service page from WebServiceSQLSelect.asmx to WebServiceSQLInsert.asmx.
  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. Open Visual Studio.NET and our new project WebServiceSQLInsert, then open our entry page WebServiceSQLInsert.asmx by double-clicking 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”**

---

---

5. Remove the child class SQLInsertResult from this project since the data insertion has no data to be returned.
6. Open the base class SQLInsertBase and perform the following modifications:
  - a. Change the class name from SQLSelectBase to SQLInsertBase.
  - b. Change the two member data items from SQLSelectOK to SQLInsertOK, and from SQLSelectError to SQLInsertError.
  - c. Add the following seven member data items into this class:
    - i. Public FacultyID As String
    - ii. Public CourseID(10) As String
    - iii. Public Course As String
    - iv. Public Schedule As String
    - v. Public Classroom As String
    - vi. Public Credit As Integer
    - vii. Public Enrollment As Integer

Go to the File|Save All menu item to save these modifications.

### 8.4.2 Web Service Project Development Procedure

We want 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. These methods are listed below:

1. Develop a Web method SetSQLInsertSP() to call a stored procedure to perform this new course insertion.
2. Develop a Web method GetSQLInsert() to retrieve the newly 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; 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 the 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 convenient for the user.

The main code developments and modifications are performed in our code-behind page WebServiceSQLInsert.vb. Most modifications will be performed on the code in the four Web methods listed above.

### 8.4.3 Develop and Modify the Code for the Code-Behind Page

Open our new project WebServiceSQLInsert if it is not open, and in the Solution Explorer window, right-click our child class SQLInsertResult and select Delete from the popup menu to remove this class from our project.

Open the code window of our code-behind page WebServiceSQLInsert.vb and change our main Web Service class's name from WebServiceSQLSelect to WebServiceSQLInsert.

Another modification is to remove the user-defined subroutine FillFacultyReader() since we do not need to return any data for this data insertion operation.

The last modification to this page is to change the code of the 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 second line from SQLRequestOK to SQLInsertOK.

```

WebServiceSQLInsert SetSQLInsertSP
<WebService(Namespace:="http://www.cambridge.org/9780521712354")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class WebServiceSQLInsert
    Inherits System.Web.Services.WebService

    <WebMethod()>
    Public Function SetSQLInsertSP(ByVal FacultyName As String, ByVal CourseID As String, ByVal Course As String, _
        ByVal Schedule As String, ByVal Classroom As String, ByVal Credit As Integer, ByVal Enroll As Integer) As _
        SQLInsertBase
        Dim cmdString As String = "dbo.InsertFacultyCourse"
        Dim sqlConnection As New SqlConnection
        Dim SetSQLResult As New SQLInsertBase
        Dim sqlCommand As New SqlCommand
        Dim intInsert As Integer
        Set SQLResult.SQLInsertOK = True
        sqlConnection = SQLConn()
        If sqlConnection Is Nothing Then
            SetSQLResult.SQLInsertError = "Database connection is failed"
            ReportError(SetSQLResult)
            Return Nothing
        End If
        sqlCommand.Connection = sqlConnection
        sqlCommand.CommandType = CommandType.StoredProcedure
        sqlCommand.CommandText = cmdString
        sqlCommand.Parameters.Add("@FacultyName", SqlDbType.Text).Value = FacultyName
        sqlCommand.Parameters.Add("@CourseID", SqlDbType.Char).Value = CourseID
        sqlCommand.Parameters.Add("@Course", SqlDbType.Text).Value = Course
        sqlCommand.Parameters.Add("@Schedule", SqlDbType.Char).Value = Schedule
        sqlCommand.Parameters.Add("@Classroom", SqlDbType.Text).Value = Classroom
        sqlCommand.Parameters.Add("@Credit", SqlDbType.Int).Value = Credit
        sqlCommand.Parameters.Add("@Enroll", SqlDbType.Int).Value = Enroll
        intInsert = sqlCommand.ExecuteNonQuery()
        sqlCommand.Dispose()
        sqlCommand = Nothing
        If Not sqlConnection Is Nothing Then sqlConnection.Close()
        sqlConnection = Nothing
        If intInsert = 0 Then
            SetSQLResult.SQLInsertError = "Data insertion is failed"
            ReportError(SetSQLResult)
        End If
        Return SetSQLResult
    End Function

```

Figure 8.43. The modification to the first Web method.

3. Change the member data in the third line from SQLRequestError to SQLInsertError.

Now let's start our modification to the first Web method.

#### **8.4.3.1 Develop and Modify the First Web Method – SetSQLInsertSP**

Perform the following modifications to the Web method GetSQLSelect(), as shown in Figure 8.43, to get our new Web method, SetSQLInsertSP.

This Web method uses a stored procedure to perform the data insertion. Recall that in Section 5.7.1.2 in Chapter 5, we developed a stored procedure dbo.InsertFacultyCourse in the SQL Server database and used it to insert a new

course into the Course table. We will use this stored procedure in this Web method to reduce our coding load. Refer to that section to get more detailed information on how to develop this stored procedure. Seven input parameters are used for this stored procedure, and they are @FacultyName, @CourseID, @Course, @Schedule, @Classroom, @Credit, and @Enroll. All these parameters will be input by the user as this Web Service project runs.

Let's take a closer look at the code for this Web method to see how it works.

- 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 the new data for a newly created course record. The returned object should be an instance of our modified base class SQLInsertBase.
- C. The content of the query string must be equal to the name of the stored procedure we developed in Section 5.7.1.2 in Chapter 5. 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 data to be returned for the data insertion action. However, in order to allow our client project to get clear feedback from executing this Web Service, we prefer to return an object that contains the information indicating whether this Web Service has been executed successfully or not.
- E. A local integer variable intInsert is declared. This variable is used to hold the value returned from calling the ExecuteNonQuery() method of the Command class, and that method will run the stored procedure to perform the data insertion action. This returned value is equal to the number of rows that have been successfully inserted into our database.
- F. Initially we set the member data SQLInsertOK that is located in our modified base class SQLInsertBase to True to indicate that our Web Service running status is good.
- G. If the connection to our sample database fails, which is indicated by a returned Connection object containing Nothing, an error message is assigned to another member data item, SQLInsertError, which is also located in our modified base class SQLInsertBase, to log this error, and 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 the new course data to be inserted into the Course table. One important point to note is that these input parameters' names must be identical to the names defined in

the stored procedure dbo.InsertFacultyCourse developed in Section 5.7.1.2 in Chapter 5. Refer to that section to get a detailed description of those parameters' names defined in that stored procedure.

- J. The ExecuteNonQuery() method is called to run the stored procedure to perform this data insertion. This method returns an integer that is stored in our local variable intInsert.
- K. A cleaning job is performed to release data objects used in this method.
- L. The value returned from calling 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 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 code development and modification for this Web method. Now we can run this Web Service project to test the insertion of new course information to our sample database via this Web Service. Click the Start Debugging button to run the project. The built-in Web interface is shown in Figure 8.44.

Click the second Web method SetSQLInsertSP to select it to open another built-in Web interface to display the input parameter window, which is shown in Figure 8.45.

Enter the following parameters to this Web method:

■ FacultyName:	Ying Bai
■ CourseID:	CSE-556
■ Course:	Advanced Fuzzy Systems
■ Schedule:	M-W-F: 1:00-1:55 PM
■ Classroom:	TC-315
■ Credit:	3
■ Enroll:	28



Figure 8.44. The running built-in Web interface.



Figure 8.45. The input parameter interface.

Click 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 8.46.

The returned member data SQLInsertOK = True indicates that our data insertion was successful. To confirm this, first click the Close button that is located at the upper right corner of this Web interface to terminate our Web Service, and then you can open our sample database CSE\_DEPT using SQL Server Management Studio to check this newly inserted course.

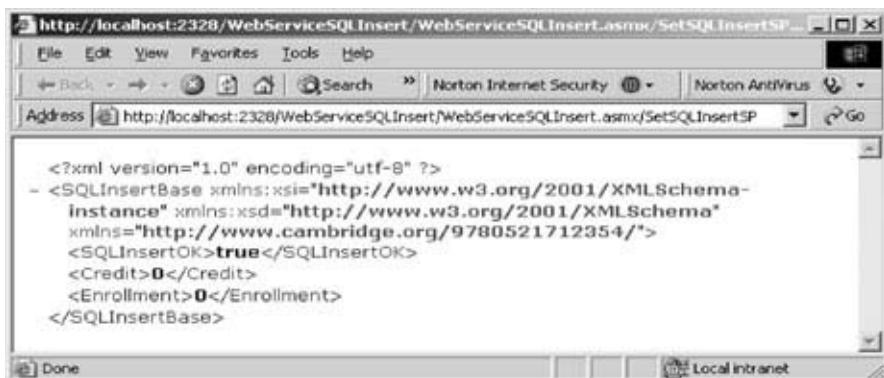


Figure 8.46. The running result of the first Web method.

It can be found from this running result that the values for the 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.

#### **8.4.3.2 Develop the Second Web Method – GetSQLInsert**

The functionality of this Web method is to retrieve all course\_id, including 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 4.18.6 in Chapter 4, we developed a joined-table query to perform the data query from the Course table to get all course\_id based on the faculty name. The reason for that is because there is no faculty name column available in the Course table, and each course or course\_id is related to a faculty\_id in the Course table. In order to get the faculty\_id that is associated with the selected faculty name, one must first go to the Faculty table to perform a query to obtain it. In this situation, a joined-table query is the desired method to complete this functionality.

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 code shown in Figure 8.47 into this page to create our new Web method, GetSQLInsert().

Let's have a closer look at the code in this Web method to see how it works.

- A. The returning data type for this Web method is our modified base class SQLInsertBase, and all course information is stored in the different member data items in this class. The input parameter for this Web method is a selected faculty name.
- B. The joined-table query string is defined here, and the ANSI 92 standard, which is the up-to-date standard, is used for the syntax of this query string. 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 4.18.6 in Chapter 4 to get a more detailed discussion of this topic). The nominal name of the input dynamic parameter for this query is @name.
- C. All used data objects are declared here, such as the Connection, Command, and DataReader objects. A returned object GetSQLResult that is instantiated from our base class SQLInsertBase is also created, and it will be returned to the calling procedure to send back the queried course information.
- D. Initially we set the running status of our Web method to OK.
- E. The user-defined function SQLConn() is called to connect to our sample database. A warning message is assigned to the member data in our returned object, the user-defined subroutine ReportError() is executed to report this error, and 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 @name using the Add() method.

```

WebServiceSQLInsert           GetSQLInsert
<WebMethod()>_
Public Function GetSQLInsert(ByVal FacultyName As String) As SQLInsertBase
    Dim cmdString As String = "SELECT Course.course_id FROM Course JOIN Faculty " +
        "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.name LIKE @name)"
    Dim sqlConnection As New SqlConnection
    Dim GetSQLResult As New SQLInsertBase
    Dim sqlCommand As New SqlCommand
    Dim sqlReader As SqlDataReader
    GetSQLResult.SQLInsertOK = True
    sqlConnection = SQLConn()
    If sqlConnection Is Nothing Then
        GetSQLResult.SQLInsertError = "Database connection is failed"
        ReportError(GetSQLResult)
        Return Nothing
    End If
    sqlCommand.Connection = sqlConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add("@name", SqlDbType.Text).Value = FacultyName
    sqlReader = sqlCommand.ExecuteReader()
    If sqlReader.HasRows = True Then
        Call FillCourseReader(GetSQLResult, sqlReader)
    Else
        GetSQLResult.SQLInsertError = "No matched course found"
        ReportError(GetSQLResult)
    End If
    If Not sqlReader Is Nothing Then sqlReader.Close()
    sqlReader = Nothing
    If Not sqlConnection Is Nothing Then sqlConnection.Close()
    sqlConnection = Nothing
    sqlCommand.Dispose()
    Return GetSQLResult
End Function

```

**Figure 8.47.** The code for our second Web method, GetSQLInsert.

- 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 code will be discussed below, is called to assign the returned course\_id to each associated member data item in our returned object GetSQLResult.
- J. Otherwise, if this reading fails, 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 code for our user-defined subroutine FillCourseReader() is shown in Figure 8.48.

```

WebServiceSQLInsert
FillCourseReader

Protected Sub FillCourseReader(ByRef sResult As SQLInsertBase, ByVal sReader As SqlDataReader)
    Dim pos As Integer
    While sReader.Read()
        sResult.CourseID(pos) = Convert.ToString(sReader.GetSqlString(0))      'the 1st column is course_id
        pos = pos + 1
    End While
End Sub

```

**Figure 8.48.** The code for the subroutine FillCourseReader.

The functionality of this piece of code 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.

Now let's test this Web method by running this project. Click the Start Debugging button to run our project, and the built-in Web interface is displayed, as shown in Figure 8.49.

Click 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 8.50.

Click the Invoke button to execute this Web method, and the running result of this method is shown in Figure 8.51.

It can be seen that all courses (that is, all course.id), including our newly inserted course CSE-556, taught by the selected faculty member, Ying Bai, are listed in an XML format.

Our second Web method is successful. Click the Close button located at the upper right corner of this page to terminate our Web Service project. Also go to File|Save All to save all methods we have developed.

#### 8.4.3.3 Develop and Modify the Third Web Method – SQLInsertDataSet

The functionality of this Web method is similar to that of the first one: to insert a new course into the Course table based on the selected faculty member. The difference is



**Figure 8.49.** The running status of our Web Service project.

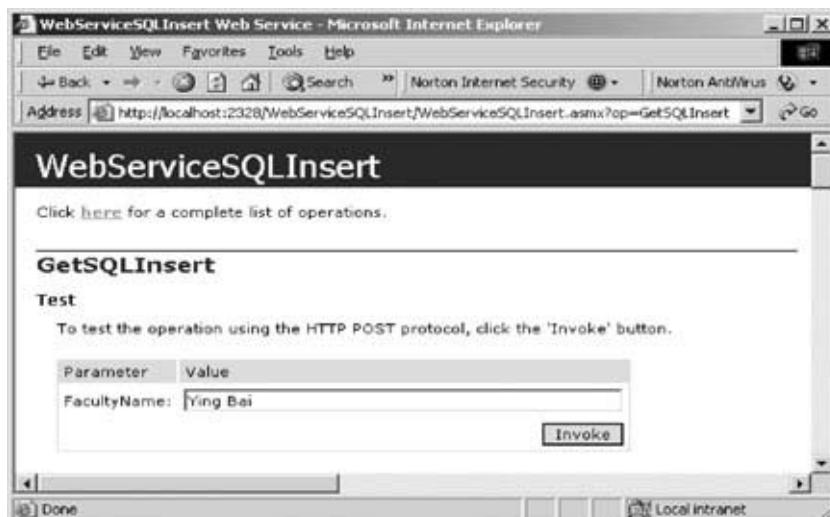


Figure 8.50. The running status of our Web Service project.

that this Web method uses multi-query to insert a new course record into the Course table and uses a DataSet as the returned object, and the returned DataSet contains the updated Course table that includes the newly inserted data. The advantages of using a DataSet as the returned object are as follows:

1. Unlike Web methods 1 and 2, which are a pair of methods in which the first one is used to insert data into the database and the second one is used to retrieve the newly inserted data from the database to confirm the data

```

<?xml version="1.0" encoding="utf-8" ?>
- <SQLInsertBase xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema-
  Schema" xmlns="http://www.cambridge.org/9780521712354">
  <SQLInsertOK>true</SQLInsertOK>
  - <CourseID>
    <string>CSC-132B</string>
    <string>CSC-234A</string>
    <string>CSE-434</string>
    <string>CSE-438</string>
    <string>CSE-556</string>
    <string xsi:nil="true" />
    <string xsi:nil="true" />
    <string xsi:nil="true" />
    <string xsi:nil="true" />
    <string xsi:nil="true" />
  </CourseID>
  <Credit>0</Credit>
  <Enrollment>0</Enrollment>
</SQLInsertBase>

```

Figure 8.51. The running result of our Web method GetSQLInsert.

- insertion, method 3 contains both insertion and retrieving functionalities. 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, but method 3 has both data insertion and data validation functionalities, so it can be called separately.
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 on or log 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.

Modify our existing Web method GetSQLSelectDataSet() and make it into our new Web method SQLInsertDataSet. Your finished Web method should match the one shown in Figure 8.52.

Let's have a closer look at the code in this Web method to see how it works.

- A. The name of the Web method is SQLInsertDataSet(). Seven input parameters are passed into this method as the data for a newly created record, and the returned data type is DataSet.
- B. The data insertion query string is declared here. In all, 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 and each course is related to a faculty\_id. Second, we can insert a new course record into the Course table by executing another query based on the faculty\_id obtained from the first query. The query string declared here is the second string.
- C. All data objects and variables used in this Web method are declared here, which include the Connection, Command, DataAdapter, DataSet, and an instance of our base class SQLInsertBase. The local integer variable intResult is used to hold the value returned from calling the ExecuteNonQuery() method, and the local string variable FacultyID is used to reserve the faculty\_id that is obtained from the first query.
- D. The member data SQLInsertOK is initialized to the normal case.
- E. The subroutine SQLConn() is called to perform the database connection. A warning message will be displayed and reported using the subroutine ReportError() if this connection encounters an 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 @Name is assigned with the actual input parameter FacultyName.
- I. The ExecuteScalar() method of the Command object is called to perform the first query to pick up the faculty\_id, and the returned faculty\_id is assigned to the local string variable FacultyID. One point to note is the data type that the ExecuteScalar() method return. An Object type is returned from calling this

A	WebServiceSQLInsert	▼	SQLInsertDataSet	▼
B	<WebMethod()> _			
C	Public Function SQLInsertDataSet(ByVal FacultyName As String, ByVal CourseID As String, _			
D	ByVal Course As String, ByVal Schedule As String, ByVal Classroom As String, ByVal Credit As Integer, _			
E	ByVal Enroll As Integer) As DataSet			
F	Dim cmdString As String = "INSERT INTO Course VALUES (@course_id, @course, @credit, @classroom, " + _			
G	"@schedule, @enrollment, @faculty_id)"			
H	Dim sqlConnection As New SqlConnection			
I	Dim SetSQLResult As New SQLInsertBase			
J	Dim sqlCommand As New SqlCommand			
K	Dim CourseAdapter As New SqlDataAdapter			
L	Dim dsCourse As New DataSet			
M	Dim intResult As Integer			
N	Dim FacultyID As String			
O	SetSQLResult.SQLInsertOK = True			
P	sqlConnection = SQLConn()			
Q	If sqlConnection Is Nothing Then			
R	SetSQLResult.SQLInsertError = "Database connection is failed"			
S	ReportError(SetSQLResult)			
T	Return Nothing			
	End If			
	sqlCommand.Connection = sqlConnection			
	sqlCommand.CommandType = CommandType.Text			
	sqlCommand.CommandText = "SELECT faculty_id FROM Faculty WHERE name LIKE @Name"			
	sqlCommand.Parameters.Add("@Name", SqlDbType.Text).Value = FacultyName			
	FacultyID = sqlCommand.ExecuteScalar()			
	sqlCommand.CommandText = cmdString			
	sqlCommand.Parameters.Add("@faculty_id", SqlDbType.Text).Value = FacultyID			
	sqlCommand.Parameters.Add("@course_id", SqlDbType.Char).Value = CourseID			
	sqlCommand.Parameters.Add("@course", SqlDbType.Text).Value = Course			
	sqlCommand.Parameters.Add("@schedule", SqlDbType.Char).Value = Schedule			
	sqlCommand.Parameters.Add("@classroom", SqlDbType.Text).Value = Classroom			
	sqlCommand.Parameters.Add("@credit", SqlDbType.Int).Value = Credit			
	sqlCommand.Parameters.Add("@enrollment", SqlDbType.Int).Value = Enroll			
	CourseAdapter.InsertCommand = sqlCommand			
	intResult = CourseAdapter.InsertCommand.ExecuteNonQuery()			
	If intResult = 0 Then			
	SetSQLResult.SQLInsertError = "No matched course found"			
	ReportError(SetSQLResult)			
	End If			
	sqlCommand.CommandText = "SELECT * FROM Course WHERE faculty_id LIKE @FacultyID"			
	sqlCommand.Parameters.Add("@FacultyID", SqlDbType.Text).Value = FacultyID			
	CourseAdapter.SelectCommand = sqlCommand			
	CourseAdapter.Fill(dsCourse, "Course")			
	CourseAdapter.Dispose()			
	CourseAdapter = Nothing			
	If sqlConnection IsNot Nothing Then sqlConnection.Close()			
	sqlConnection = Nothing			
	sqlCommand.Dispose()			
	Return dsCourse			
	End Function			

**Figure 8.52.** The code for the Web method SQLInsertDataSet.

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 for the INSERT command are initialized by assigning them with the actual input values. The point to note is the data

types of the last two parameters. Both “credit” and “enrollment” are integers, 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 courses, from the database based on the input faculty\_id, is assigned to the CommandText property of the Command object.
- P. The dynamic parameter faculty\_id is initialized with the actual faculty\_id obtained from the first query as we did above.
- Q. The initialized Command object is assigned to the SelectCommand property of the DataAdapter.
- R. The Fill() method of the DataAdapter is executed to retrieve all courses, including the 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 more coding is involved in this method. Yes, that’s true. But this method has two functionalities: inserting data into the database and validating the inserted data. In order to validate the data insertion for the first method, the second Web method must be executed. So, the third Web method has less coding compared with the first one in terms of data insertion and data validation.

Now let’s run our Web Service project to test this Web method using the built-in Web interface. Click the Start Debugging button to run the project, and click our third Web method SQLInsertDataSet from the built-in Web interface to start it. The parameters dialog is displayed, which is shown in Figure 8.53. Enter the following parameters into each associated Value box as the data of a new course:

■ FacultyName:	Ying Bai
■ CourseID:	CSE-665
■ Course:	Advanced Fuzzy Systems
■ Schedule:	T-H: 1:00-2:25 PM
■ Classroom:	TC-309
■ Credit:	3
■ Enroll:	32



Figure 8.53. The finished parameter dialog box.

Your finished parameter dialog box should match the one shown in Figure 8.53.

Click the Invoke button to run this Web method to perform this new course insertion. The running result is shown in Figure 8.54.

All five courses, including the course CSE-665, that is, the newly inserted course, are displayed using the XML format or tags in this running result dialog box.

One point to note is that you can only insert this new course record into the database once, which means that after this new course has been inserted into the database, you cannot continue to click the Invoke button to perform another insertion with the same course information since data inserted into the database must be unique.

Click the Close button located at the upper right corner of this Web interface to terminate our service. The completed Web Service project WebServiceSQLInsert can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

Next, let's develop our fourth Web method.

#### **8.4.3.4 Develop the Fourth Web Method – GetSQLInsertCourse**

The functionality 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.



```

<faculty_id>B78880</faculty_id>
</Course>
- <Course diffgr:id="Course4" msdata:rowOrder="3">
  <course_id>CSE-438</course_id>
  <course>Advd Logic & Microprocessor</course>
  <credit>3</credit>
  <classroom>TC-213</classroom>
  <schedule>M-W-F: 11:00-11:55 AM</schedule>
  <enrollment>35</enrollment>
  <faculty_id>B78880</faculty_id>
</Course>
- <Course diffgr:id="Course5" msdata:rowOrder="4">
  <course_id>CSE-665</course_id>
  <course>Advanced Fuzzy Systems</course>
  <credit>3</credit>
  <classroom>TC-309</classroom>
  <schedule>T-H: 1:00-2:25 PM</schedule>
  <enrollment>32</enrollment>
  <faculty_id>B78880</faculty_id>
</Course>
</NewDataSet>
</diffgr:diffgram>
</DataSet>

```

Figure 8.54. The running result of our third Web method.

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 for this method. Relatively speaking, the stored procedure is more efficient compared with the normal query, so we will use the former to perform this query.

Now let's first create our stored procedure WebSelectCourseSP.

#### 8.4.3.4.1 Create the Stored Procedure WebSelectCourseSP

Open Visual Studio.NET 2005 and the Server Explorer window, and click our sample database folder CSE\_DEPT.mdf to connect it. Then expand to the Stored Procedures folder. To create a new stored procedure, right-click this folder and select the item Add New Stored Procedure to open the Add New Stored Procedure dialog box.

Enter the code shown in Figure 8.55 into this dialog box to create our new stored procedure.

Go to File|Save StoredProcedure1 to save this stored procedure.

To test this stored procedure, go to the Server Explorer window and right-click this newly created stored procedure, and then select the Execute item from the popup menu to open the Run Stored Procedure dialog box. Enter CSE-438 into the Value box in this dialog as the input course\_id, and click the OK button to run this stored procedure. The running result is displayed in the Output window, which is shown in Figure 8.56.

One row is found and returned from the Course table in our sample database. To view all returned columns, move the horizontal bar at the bottom of this dialog box to the right. Our stored procedure works fine.

The screenshot shows the Microsoft Visual Studio interface with the title bar "WebServiceSQLInsert - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has icons for New, Open, Save, Print, and others. A Server Explorer window on the left shows a database connection named "CSE\_DEPT" under "Data Connections". The main code editor window displays the following T-SQL code:

```

CREATE PROCEDURE dbo.WebSelectCourseSP
(
    @CourseID text
)
AS
SELECT course, credit, classroom, schedule, enrollment, faculty_id FROM Course
WHERE course_id LIKE @CourseID
RETURN

```

Below the code editor, status bars show "Ln 9 Col 1 Ch 1" and "INS".

Figure 8.55. The code for the stored procedure WebSelectCourseSP.

Right-click our database folder CSE\_DEPT.mdf, and select the item Close Connection from the popup menu to close this database connection.

#### 8.4.3.4.2 Develop the Code to Call This Stored Procedure

Now let's develop the code for our fourth Web method GetSQLInsertCourse() to call this stored procedure to perform the course information query.

Open the code-behind page WebServiceSQLInsert.vb, and add the code shown in Figure 8.57 into this page to create this Web method.

Let's take a look at the code in this Web method to see how it works.

- 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.
- The content of the query string is the name of the stored procedure we developed in the last section. This is required if a stored procedure is used and called later to perform a data query. This name must be identical to the name of the stored procedure we developed; otherwise, a running error may be encountered since the stored procedure is identified by its name as the project runs.

The screenshot shows the "Output" window with the title bar "Output". The dropdown menu says "Show output from: Database Output". The output pane displays the following text:

```

Running [dbo].[WebSelectCourseSP] ( @CourseID = CSE-438 ).

course
-----
Advd Logic & Microprocessor
No rows affected.
(1 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[WebSelectCourseSP].

```

Figure 8.56. The running result of the stored procedure WebSelectCourseSP.

**WebServiceSQLInsert**  **GetSQLInsertCourse** 

```

<WebMethod()> _
Public Function GetSQLInsertCourse(ByVal CourseID As String) As SQLInsertBase
    Dim cmdString As String = "dbo.WebSelectCourseSP"
    Dim sqlConnection As New SqlConnection
    Dim GetSQLResult As New SQLInsertBase
    Dim sqlReader As SqlDataReader
    GetSQLResult.SQLInsertOK = True
    sqlConnection = SQLConn()
    If sqlConnection Is Nothing Then
        GetSQLResult.SQLInsertError = "Database connection is failed"
        ReportError(GetSQLResult)
        Return Nothing
    End If
    Dim sqlCommand = New SqlCommand(cmdString, sqlConnection)
    sqlCommand.CommandType = CommandType.StoredProcedure
    sqlCommand.Parameters.AddWithValue("@CourseID", SqlDbType.Text).Value = CourseID
    sqlReader = sqlCommand.ExecuteReader()
    If sqlReader.HasRows = True Then
        Call FillCourseDetail(GetSQLResult, sqlReader)
    Else
        GetSQLResult.SQLInsertError = "No matched course found"
        ReportError(GetSQLResult)
    End If
    sqlReader.Close()
    sqlReader = Nothing
    sqlConnection.Close()
    sqlCommand.Dispose()
    Return GetSQLResult
End Function

```

Figure 8.57. The code for the Web method GetSQLInsertCourse.

- C. Some data objects such as the Connection and the DataReader are created here. Also, a returned instance of our base class is created.
- D. The subroutine SQLConn() is called to perform the database connection. A warning message is displayed and reported using the subroutine ReportError() if an 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 use 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 the course information query in this method.
- G. The dynamic parameter @CourseID is assigned with the actual parameter CourseID that will be entered as an input by the user as the project runs. One point to note is that the nominal name of this dynamic parameter must be identical to 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 code is shown below, 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 performed.
- 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's detailed information, is returned to the calling procedure.

The code for the subroutine FillCourseDetail() is shown in Figure 8.58.

Let's have a closer look at this piece of code 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 items that contain the course information in this returned object can be returned to the calling procedure or our Web method – GetSQLInsertCourse(). From this point of view, this subroutine works like a function and our object can be returned as this subroutine is completed.
- B. The Read() method of the DataReader is executed to read the course record from the DataReader.
- C. A With...End With block is executed to assign each column of the queried course record to the associated member data item in our base class. A Convert.ToString() class method is used to convert all data to strings before this assignment.

The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar says "WebServiceSQLInsert" and the tab bar shows "FillCourseDetail". The code is written in VB.NET:

```
Protected Sub FillCourseDetail(ByRef sResult As SQLInsertBase, ByVal sReader As SqlDataReader)
    sReader.Read()
    With sResult
        .FacultyID = Convert.ToString(sReader("faculty_id"))
        .Course = Convert.ToString(sReader("course"))
        .Schedule = Convert.ToString(sReader("schedule"))
        .Classroom = Convert.ToString(sReader("classroom"))
        .Credit = Convert.ToString(sReader("credit"))
        .Enrollment = Convert.ToString(sReader("enrollment"))
    End With
End Sub
```

Three annotations are present: 'A' points to the first argument 'sResult', 'B' points to the 'Read()' call, and 'C' points to the 'With sResult' block.

Figure 8.58. The code for the subroutine FillCourseDetail.

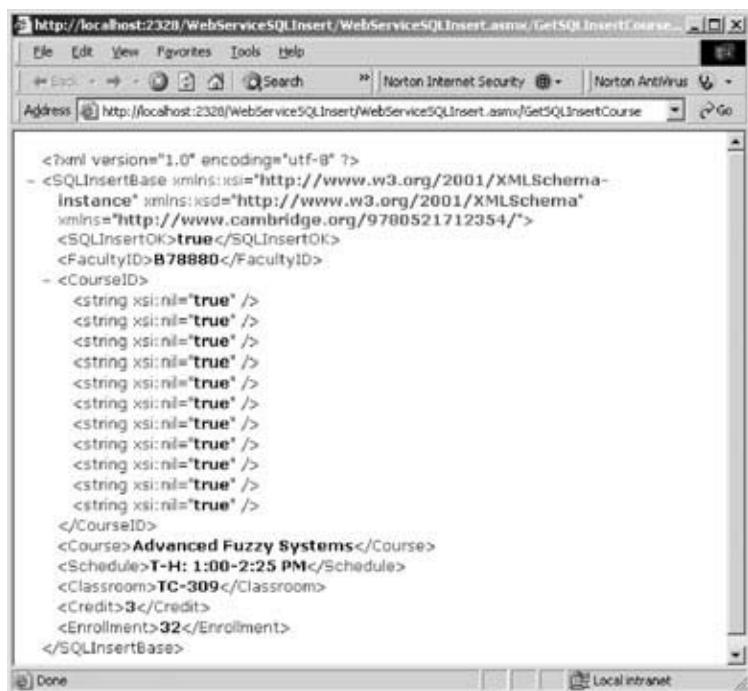


Figure 8.59. The running result of our Web method – GetSQLInsertCourse.

Now let's run our project to test this Web method. Click the Start Debugging button to run the project. Select our Web method GetSQLInsertCourse from the built-in Web interface, enter CSE-665 as the course\_id into the Value box, and then click the Invoke button to run this Web method. The running result is shown in Figure 8.59.

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, so the size of our array CourseID(10) is 11. This member data is used for our second Web method – GetSQLInsert() – that returns an array containing 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 the resulting file.

Click the Close button located at the upper right corner of this Web interface to terminate our service. The completed Web Service project WebServiceSQLInsert that contains all four Web methods can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

At this point, we have finished all developing jobs in our Web Service project in the server side. Next, we want to develop some professional Windows-based and Web-based applications with beautiful graphical user interfaces to use or to consume the Web Service application we developed in this part. Those Windows-based and Web-based applications can be considered as Web Service clients.

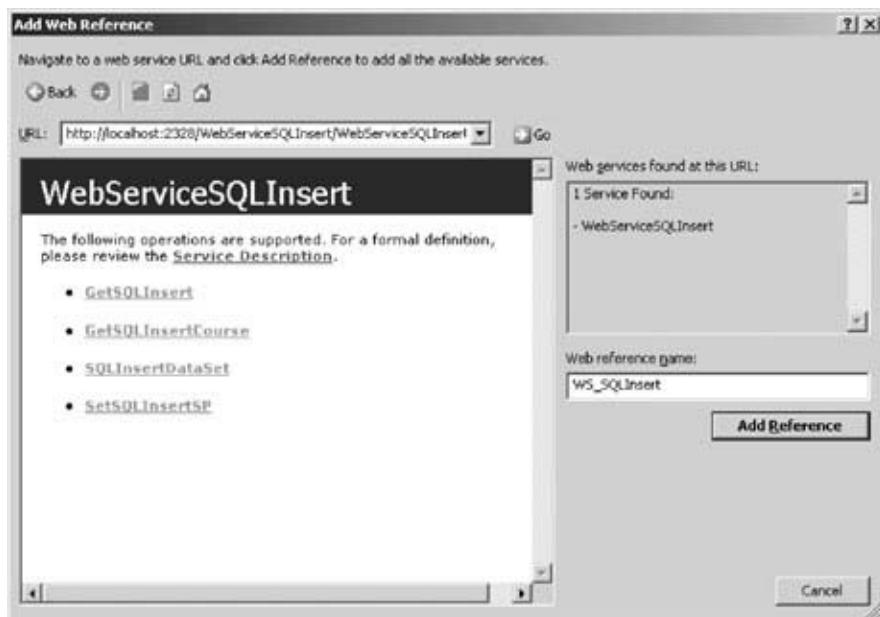


Figure 8.60. Adding a Web reference to the client project.

#### **8.4.4 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 application. 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 application.

##### **8.4.4.1 Create a Web Service Proxy Class**

Basically, adding a Web reference to our Windows-based or Web-based application is to execute a searching process. During this process, Visual Studio.NET 2005 will try to find all Web Services available to our applications.

To add a Web reference to our client project, we first need to create a client project. Open Visual Studio.NET 2005 and create a new Windows-based project, and name this project WinClientSQLInsert.

In Visual Studio.NET, right-click our new project WinClientSQLInsert from the Solution Explorer window, and select the item Add Web Reference from the popup menu to open the Add Web Reference dialog box, which is shown in Figure 8.60.

As we mentioned in Section 8.3.10.1, there are two ways to select the desired Web Service and add it as a reference to our client project; one way is to use the browser provided by Visual Studio.NET 2005 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 dialog box. The second way requires you to first run the Web Service, and then copy its URL and paste it to the URL box in this dialog box if you did not deploy that Web Service to IIS. If you did deploy that Web Service, you can directly type that URL into the URL box in this dialog box.

Because we developed our Web Service using the File System on our local computer and have not deployed our Web Service to IIS, we can use the second way to find our Web Service. Open our Web Service project WebServiceSQLInsert and click the Start Debugging button to run it. Copy the URL from the Address bar and then switch back to our client project WinClientSQLInsert, and paste that URL into the URL box in the Add Web Reference dialog box. Click the Go button to allow Visual Studio.NET 2005 to search for it.

When our Web Service is found, its name is displayed in the right pane, as shown in Figure 8.60.

Alternately, you can change the name for this Web reference from localhost to any meaningful name, such as WS\_SQLInsert in our case. Click the Add Reference button to add this Web Service as a reference to our new client project. Click the Close button from our Web Service built-in Web interface window to terminate our Web Service project.

Immediately you will find that the Web Service WS\_SQLInsert, which is under the folder Web References, has been added into the Solution Explorer window in our project. This reference is the so-called Web Service proxy class.

Next, let's develop the graphical user interface by adding useful controls to interface with our Web Service and to display the queried course information.

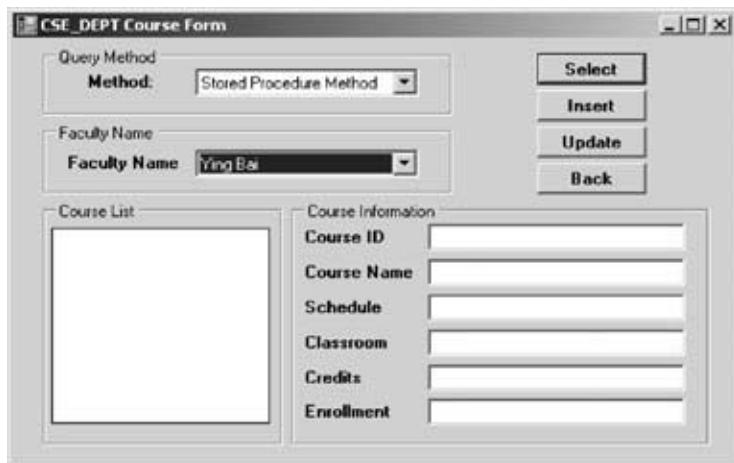
#### **8.4.4.2 Develop the Graphical User Interface for the Client Project**

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, 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 graphical user interface located in the project SQLUpdateDeleteRTOBJECT we developed in Chapter 6. Open that project from the folder **DBProjects\Chapter 6** [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354). Then open the Course form window and select all controls from that form by going to Edit|Select All, and then go to the Edit|Copy menu item to copy all controls selected from this form window.

Return to our new Windows-based Web Service client project WinClientSQLInsert, open our form window, and paste the controls we copied from the Course form in the project SQLUpdateDeleteRTOBJECT. The only modification to this form is to add one more text box control named txtCourseName and the associated label



**Figure 8.61.** The finished graphical user interface.

control since we need them to enter the input parameters for the insert query. Make the following changes to finish these modifications:

1. Add a text box control just under the Course ID text box control and name this control txtCourseName by changing its Name property from TextBox1 to txtCourseName.
2. Add an associated label control and change its Text property from Label1 to Course Name.

Your finished graphical user interface is shown in Figure 8.61.

Also set up the TabIndex values for all controls as shown in Figure 8.62.



**Figure 8.62.** The TabIndex setup of controls in the client form window.

The ComboMethod combo box control is used to select two different methods developed in our Web Service project to get our desired course information:

1. The Stored Procedure Method that uses a stored procedure to insert the course information into the database.
2. The DataSet Method that uses three separate queries to insert the course information into the database and return a DataSet that contains the detailed course information.

The Faculty Name combo box control is used to select the desired faculty name as the input parameter to the Web methods to insert and pick up the desired course information.

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 used to trigger a data validation action to confirm that data insertion, and the Back button is used to terminate our project.

The detailed functionalities of this project are as follows:

1. Insert Data Using the Stored Procedure Method: When the project runs, after this method and a faculty name as well as a new course record described by six pieces of course information stored in six text boxes have been selected or entered, the Insert button is clicked by the user. Our client project is connected to our Web Service based on the Web reference we provided, and we 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, including 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's click event procedure, which 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, including the newly inserted course\_id, from the database and store them into an instance of our base class SQLInsertBase that is developed 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 CourseList list box control 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, which 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 CourseList list box control in our client form window.

5. Get Detailed Course Information for a Specific Course: When either method is selected and a course\_id displayed in the CourseList list box control is clicked, the Web method GetSQLInsertCourse() developed 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 that is developed 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 text box controls in our client form window.

Now let's take care of the coding for this project to connect to our Web Service using the Web reference developed in the last section to call the associated Web methods to perform different data actions.

#### **8.4.4.3 Develop the Code to Consume the Web Service**

The coding job can be divided into the following four parts:

1. Coding to initialize and terminate the client project.
2. Coding to insert a new course record into the database using both methods.
3. Coding to validate the data insertion using both methods.
4. Coding to get the detailed information for a specific course using both methods.

Now let's start our coding process based on the four steps described above.

##### **8.4.4.3.1 Develop the Code to Initialize and Terminate the Client Project**

This coding includes developing code for the Form\_Load event procedure and the Back button's click event procedure and some other initialization coding such as the Imports commands and form-level variables development.

Open Visual Studio.NET 2005 and our client project WinClientSQLInsert if it is not open, and then open the code window of this client project by clicking the View Code button from the Solution Explorer window. Enter the code shown in Figure 8.63 into this code window.

Let's have a closer look at this piece of code 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

(CourseForm Events)		Load
A	<pre>Imports System.Data Imports System.Data.SqlClient</pre>	
B	<pre>Public Class CourseForm     Private dsFlag As Boolean     Private wsDataSet As DataSet     Private wsSQLResult As New WS_SQLInsert.SQLInsertBase</pre>	
C	<pre>Private Sub CourseForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.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</pre>	
D	<pre>    ComboBoxMethod.Items.Add("Stored Procedure Method")     ComboBoxMethod.Items.Add("DataSet Method")     ComboBoxMethod.SelectedIndex = 0 End Sub</pre>	
E	<pre>Private Sub cmdBack_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdBack.Click     Me.Close() End Sub End Class</pre>	

Figure 8.63. The code of the Form\_Load and Back button event procedures.

is used to receive the instance returned from calling the first Web method SetSQLInsertSP() when we perform a data insertion.

- C. In the Form\_Load event procedure, eight default faculty members are added into the ComboName 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 to perform 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, the Stored Procedure Method and the DataSet Method, are added into the ComboBoxMethod 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. The first method, the Stored Procedure Method, is selected as the default one.
- E. The coding for the Back button's click event procedure is very simple. The Close() method is called to terminate our client project.

The first coding job is done. Let's continue to perform the next coding.

#### 8.4.4.3.2 Develop the Code to Insert a New Course Record into the Database

This coding can be divided into two parts based on two methods: the Stored Procedure Method and the DataSet Method. Because of the similarity of the coding for these two methods, we will combine them.

To insert a new course record into the database via our Web Service, the following three jobs must have been completed before the Insert button can be clicked:

1. The Web method has been selected from the ComboMethod combo box control.
2. The faculty name has been selected from the ComboName combo box control.
3. Six text boxes have been filled with the associated information for a new course to be inserted.

Besides those conditions, one more important requirement for this data insertion is that any new course record can be inserted into the database only once. In other words, no duplicate record can be inserted into the database. Duplication can be identified by checking the content of the Course ID text box, or the course\_id column 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 can occur. So based on this analysis, in order to avoid duplicate insertion, the Insert button should be disabled after a new course record is inserted into the database, and this button will be kept disabled until a different or new course\_id is entered into the Course ID text box, which means that a new record is ready to be inserted into the database.

Keep this in mind, and now let's develop the code for the Insert button's click event procedure.

Double-click the Insert button from the Design View of our client project to open the Insert button's click event procedure, and enter the code shown in Figure 8.64 into this event procedure.

Let's have a closer look at this piece of code 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 the Web methods developed in our Web Service project.
- B. If the user selects 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. 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 the error 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 the calling Web method works fine. One of the member data items, SQLInsertOK, is used to store this running status. If this status is False, which means that something went wrong during the execution of this Web method, the error message is displayed using the member data item SQLInsertError that stored the error source.

```

cmdInsert Click

```

```

Private Sub cmdInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim wsSQLInsert As New WS_SQLInsert.WebServiceSQLInsert
    If ComboMethod.Text = "Stored Procedure Method" Then
        Try
            wsSQLResult = wsSQLInsert.SetSQLInsertSP(ComboName.Text, txtCourseID.Text,
                txtCourseName.Text, txtSchedule.Text, txtClassRoom.Text,
                txtCredits.Text, txtEnroll.Text)
        Catch err As Exception
            MsgBox("Web service is wrong: " & err.Message)
        End Try
        If wsSQLResult.SQLInsertOK = False Then
            MsgBox(wsSQLResult.SQLInsertError)
        End If
    Else
        dsFlag = True      'indicate the DataSet insert is performed
        Try
            wsDataSet = wsSQLInsert.SQLInsertDataSet(ComboName.Text, txtCourseID.Text,
                txtCourseName.Text, txtSchedule.Text, txtClassRoom.Text,
                txtCredits.Text, txtEnroll.Text)
        Catch err As Exception
            MsgBox("Web service is wrong: " & err.Message)
        End Try
        End If
        cmdInsert.Enabled = False
    End Sub

```

Figure 8.64. The code for the Insert button event procedure.

- E. If the user selects 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 will be False when users want to retrieve course information from the database when they click the Select button but have not called this Web method to first insert a new course record. A message is displayed to direct the user 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 performs 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 job is for the Course ID text box, that is, the `TextChanged` event procedure of the Course ID text box. As we mentioned, the Insert button should be disabled after one new course record has been inserted into the database to avoid duplicate insertion of the same data. But this button should be enabled when a new, different course record is ready to be inserted into the database. As long as the content of the Course ID text box has changed, it means that a new record is ready and the Insert button should be enabled. To do this coding, double-click the Course ID

text box from the Design View of our client project window to open its TextChanged event procedure. Enter the following code into this event procedure:

---

```
cmdInsert.Enabled = True
```

---

At this point, we have finished the coding for the data insertion process. Before we continue to perform the following coding, we will 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 online status or our Web Service project is open to the local computer. To do that, open our Web Service project WebServiceSQLInsert and click the Start Debugging button to run it. Then you can terminate our Web Service project by clicking the Close button. Now you will find that a small white icon is added into the status bar on the bottom of the screen. This small white icon means that our Web Service is in open status and any client can access and use it now. The reason we terminate our Web Service project is that we do not need to keep our Web Service project in a running status, but we do need it in the open status. After our Web Service project runs once, it will be in open status even when we close it.

Now run our Windows-based client project WinClientSQLInsert by clicking the Start Debugging button. When the CourseForm window is displayed, perform the following two insertions using two Web methods with the following selections and parameters:

1. Insert the first new course record, shown in Table 8.2, using the Stored Procedure method, and then click the Insert button to finish this data insertion.
2. Insert the second new course record, shown in Table 8.3, using the DataSet method, and then click the Insert button to finish this data insertion.

Now click the Back button to terminate our client project. To confirm these two data insertions, open Microsoft SQL Server Management Studio or Studio Express. Then open our sample database CSE.DEPT and our Course table. You can find that these two records have been added into our Course table in the last two rows.

**Table 8.2. 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 8.3.** 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

It is highly recommended to delete these two new records from our Course table after this checking since we will perform the same insertions when we confirm these data insertions in the following section.

#### 8.4.4.3.3 Develop the Code to Perform the Inserted Data Validation

To confirm or validate the data insertion, we can open our database and data table to check it. But a professional way to do it is to use codes to perform this validation. In this section, we discuss how to perform this validation by developing code for the Select button's click event procedure in our client project.

As mentioned in the previous sections, when 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 selected 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 coding similarity between these two methods, we combine them and put them into the Select button's click event procedure.

Now double-click the Select button from the Design View of our client project WinClientSQLInsert to open this event procedure, and enter the code shown in Figure 8.65 into this event procedure.

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

- A. An instance of our Web Service reference or 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.
- B. If the Stored Procedure method is 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 an error is encountered during the execution of this Web method.

```

cmdSelect Click
Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim wsSQLInsert As New WS_SQLInsert.WebServiceSQLInsert
    If ComboMethod.Text = "Stored Procedure Method" Then
        Try
            wsSQLResult = wsSQLInsert.GetSQLInsert(ComboName.Text)
        Catch err As Exception
            MsgBox("Web service is wrong: " & err.Message)
        End Try
        If wsSQLResult.SQLInsertOK = False Then
            MsgBox(wsSQLResult.SQLInsertError)
        End If
        ProcessObject(wsSQLResult)
    Else
        If dsFlag = False Then
            MsgBox("No DataSet Insert performed, do data insertion first")
            Exit Sub
        End If
        Call FillCourseDataSet(wsDataSet)
        dsFlag = False
    End If
End Sub

```

Figure 8.65. The code for the Select button event procedure.

- 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 an error occurs.
- D. If this Web method works fine, a user-defined subroutine ProcessObject(), whose code is shown below, is called to extract all course columns from the returned instance wsSQLResult.
- E. If the user selects the DataSet method, we first 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 retrieval. 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 because if nothing has been inserted, nothing can be returned in the 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.
- F. If the Web method SQLInsertDataSet() has been executed, a user-defined subroutine FillCourseDataSet(), whose code is shown below, is called to fill the list box control with all retrieved course\_id.
- G. Finally, the dsFlag is reset to False.

The code for the subroutines ProcessObject() and FillCourseListBox() is shown in Figure 8.66.

Let's have a look at the code in these two subroutines to see how it works.

CourseForm		▼	ProcessObject	▼
A	<pre>Private Sub ProcessObject(ByRef wsResult As WS_SQLInsert.SQLInsertBase)     If wsResult.SQLInsertOK = True Then         Call FillCourseListBox(wsResult)     Else         MsgBox("Course information cannot be retrieved: " &amp; wsResult.SQLInsertError)         End If     End Sub</pre>			
B	<pre>Private Sub FillCourseListBox(ByRef sqlResult As WS_SQLInsert.SQLInsertBase)     Dim index As Integer     CourseList.Items.Clear()           'clean up the course listbox     For index = 0 To sqlResult.CourseID.Length - 1         If sqlResult.CourseID(index) &lt;&gt; vbNullString Then             CourseList.Items.Add(sqlResult.CourseID(index))         End If     Next index End Sub</pre>			
C				
D				
E				

**Figure 8.66.** The code for the subroutines ProcessObject and FillCourseListBox.

- 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.
- A warning message is displayed if any error was encountered during the execution of that Web method.
- 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.
- 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 would be attached to resulting display would be the end of the original course\_id in this control and the messy.
- 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 note is the upper bound and the length of this array. The length or 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, that element or course\_id is added into the list box control by using the Add() method.

The code for the subroutine FillCourseDataSet() is shown in Figure 8.67. Let's have a look at the code in this subroutine to see how it works.

- Two objects, a DataSet 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.
- The list box control is first cleaned up to avoid a messy display of multiple course\_id.

```

CourseForm
FillCourseDataSet

Private Sub FillCourseDataSet(ByRef ds As DataSet)
    Dim CourseTable As New DataTable
    Dim CourseRow As DataRow
    CourseList.Items.Clear()           'clean up the course listbox
    CourseTable = ds.Tables("Course")
    For Each CourseRow In CourseTable.Rows
        CourseList.Items.Add(CourseRow(0))   'the 1st column is course_id
    Next
End Sub

```

Figure 8.67. The code for the subroutine FillCourseDataSet.

- 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 note is that the first column has an index value of 0, not 1, since the index starts from 0.

At this point we have finished all coding for the Select button's click event procedure. In other words, all coding related to the data validation is done.

Now let's run our client project to perform the data validation after the data insertion process. Before we run the project, make sure that the following two conditions are met:

1. Our Web Service is in open status. 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 the Web Service project and click the Start Debugging button to run it. After the Web Service starts to run, you can close it if you like, but it remains in the open status.
2. The two new course records we inserted before to test the Insert button's click event procedure have been deleted since we want to insert the same course records in the following test.

Now click the Start Debugging button to run our client project. Enter the same input parameters as shown in Table 8.2 in Section 8.4.4.3.2, and click the Insert button to finish this data insertion using the stored procedure method. Next, enter the same input parameters as shown in Table 8.3 in Section 8.4.4.3.2, and click the Insert button to finish this data insertion using the DataSet method.

To check or validate these data insertions, make sure that the Method is still the DataSet Method and the Faculty Name is Ying Bai, and then click the Select button to retrieve all course\_id from the database. It can be found that all six courses taught by the selected faculty member 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. Now we can select another faculty member from the Faculty Name combo box control and then click the Select button to pick up all course\_id taught by the selected faculty member. Reselect the default Faculty Name, Ying Bai, and then click the Select button to try to retrieve all course\_id taught by the

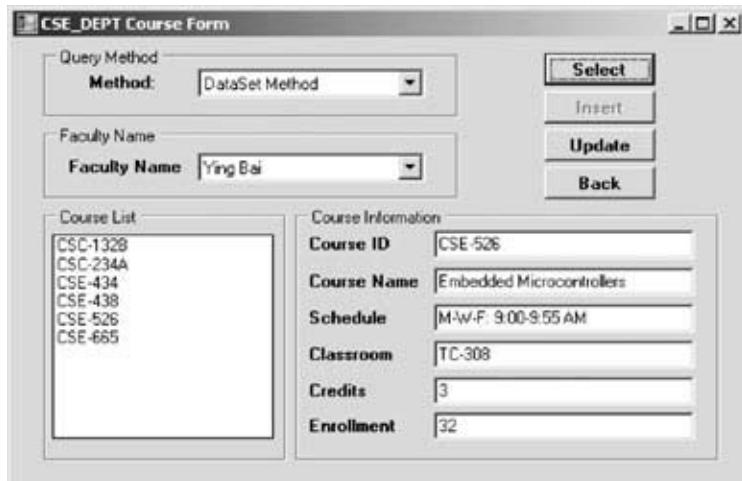


Figure 8.68. The running result of the data validation.

selected faculty member. You will find that the same results as we obtained using the DataSet Method are displayed in the list box control.

The running result of the data validation is shown in Figure 8.68. It can be seen that our two newly inserted courses, CSE-665 and CSE-526, have been added and displayed in the list box control and our data insertion is successful.

Click the Back button to terminate our project.

Next, let's concentrate on the code development to display the detailed course information for a selected course from the list box control.

#### 8.4.4.3.4 Develop the Code to Get the Detailed Information for a Specific Course

The functionality of this code is that the detailed course information such as the course name, schedule, classroom, credit, and enrollment will be displayed in the associated text box control when the user clicks and selects one course from the list box control. The main coding job is performed inside the SelectedIndexChanged event procedure of the list box control because when user clicks or selects a course 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, the Web method GetSQLInsertCourse() developed in our Web Service project WebServiceSQLInsert is called, and this method returns an instance of the base class SQLInsertBase to our client project. The detailed course information is stored in that returned instance.

Double-click the CourseList list box control from the Design View of our client project window to open the SelectedIndexChanged event procedure of the list box control, and enter the code shown in Figure 8.69 into this event procedure.

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

```

CourseList SelectedIndexChanged
Private Sub CourseList_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CourseList.SelectedIndexChanged
    Dim wsSQLInsert As New WS_SQLInsert.WebServiceSQLInsert
    Try
        wsSQLResult = wsSQLInsert.GetSQLInsertCourse(CourseList.Text)
    Catch err As Exception
        MsgBox("Web service is wrong: " & err.Message)
    End Try
    If wsSQLResult.SQLInsertOK = False Then
        MsgBox(wsSQLResult.SQLInsertError)
        Exit Sub
    End If
    Call FillCourseDetail(wsSQLResult)
End Sub

Private Sub FillCourseDetail(ByRef sqlResult As WS_SQLInsert.SQLInsertBase)
    txtCourseID.Text = CourseList.Text
    txtCourseName.Text = sqlResult.Course
    txtSchedule.Text = sqlResult.Schedule
    txtClassRoom.Text = sqlResult.Classroom
    txtCredits.Text = sqlResult.Credit
    txtEnroll.Text = sqlResult.Enrollment
End Sub

```

Figure 8.69. The code for the SelectedIndexChanged event procedure.

- 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.
- 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 retrieval. The selected course.id is stored in the Text property of the CourseList control.
- A exception message is displayed if any error is encountered during the execution of this Web method and caught by the system method Catch.
- 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 during the running of this Web method, an error message is displayed and the subroutine is exited.
- 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 text box control in our client window form.
- The coding for the subroutine FillCourseDetail() is 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 text box.

When performing this functionality to get 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 for our Windows-based client project. Now we try to run the client project to test all functionalities of this project as well as the functionalities of our Web Service project. Before we do this, make sure that the following jobs have been performed:

1. Our Web Service WebServiceSQLInsert is in open status. 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 the Web Service project and click the Start Debugging button to run it. After the Web Service starts to run, you can close it if you like, but it is still in the open status.
2. The two new course records we inserted before to test the Insert button's click event procedure have been deleted since we want to insert the same course records in this test.

Now click the Start Debugging button to run our client project. Insert two new courses by entering the parameters listed in Tables 8.2 and 8.3 and clicking the Insert button, and then perform the data validation by clicking 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 the associated text boxes, as shown in Figure 8.70.

Click the Back button to terminate our client project.

The completed Windows-based Web Service client project WinClientSQLInsert is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

#### **8.4.5 Build Web-Based Web Service Clients to Consume the Web Services**

As we did in Section 8.3.11, it can be found that there is no significant difference between developing a Web-based client application and developing a

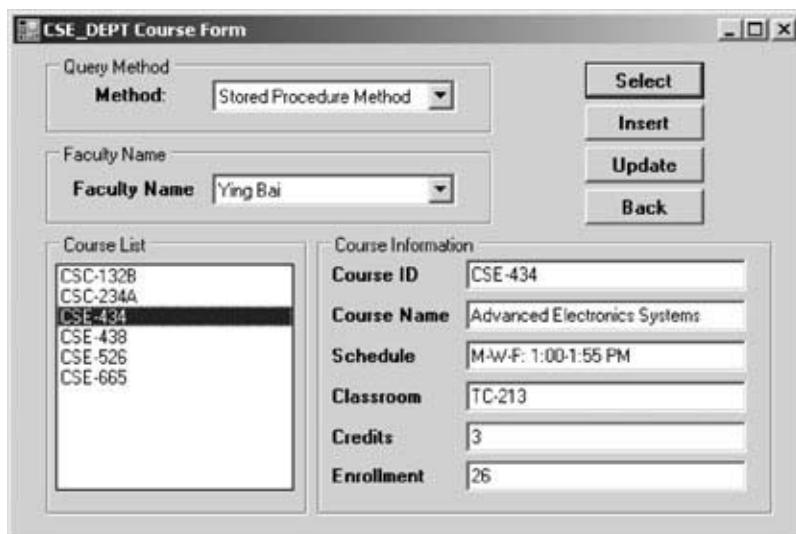


Figure 8.70. The running status of getting the detailed course information.

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 any problem. Visual Studio.NET will create the same document files, such as the Discovery Map file, the WDSL file and the DISCO file, for the client project no matter whether this Web Service is consumed by a Windows-based or Web-based client application.

To save time and space, we can modify the existing ASP.NET Web application SQLWebInsert we developed in Chapter 7 to make it into our new Web-based Web Service client project named WebClientSQLInsert. We could copy and rename that entire project as our new Web-based client project, but instead we will create a new ASP.NET Web site project and only copy and modify the Course page.

This project can be developed in the following steps:

1. Create a new ASP.NET Web site project WebClientSQLInsert, and add the existing Web 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 window of the Course.aspx to meet our data insertion requirements.
3. Modify the code 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:
  - a. Modify the code in the Page\_Load event procedure.
  - b. Develop the code for the Insert button's click event procedure.
  - c. Develop the code for the TextChanged event procedure of the Course ID text box.
  - d. Modify the code 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 to those we developed in the last Windows-based Web Service client project WinClientSQLInsert, and one can copy and paste them into our new project with a few modifications.
  - e. Modify the code in the SelectedIndexChanged event procedure.
  - f. Modify the code in the Back button's click event procedure.

Now let's start with the first step listed above.

#### **8.4.5.1 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 8\WebClientSQLInsert" into the name box that is next to the Location box, and click the OK button to create this new project.

In the opened new project window, right-click our new project icon WebClientSQLInsert from the Solution Explorer window, and select Add Existing Item from the popup menu to open the Add Existing Item dialog box. Browse to our Web project SQLWebInsert that is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the



Figure 8.71. The finished Add Web Reference dialog box.

folder **DBProjects\Chapter 7**, select it, and then click the Open button to open all existing items for this Web site project.

Select both items, Course.aspx and Course.aspx.vb, from the list and click the Add button to add these two items into our new Web site project.

#### **8.4.5.2 Add a Web Service Reference and Modify the Web Form Window**

To add a Web reference of our Web Service to this new Web site project, right-click our new project icon from the Solution Explorer window and select the item Add Web Reference from the popup menu. Now open our Web Service project Web-ServiceSQLInsert and click the Start Debugging button to run it. As the project runs, copy the URL from the Address box and paste it into the URL box in our Add Web Reference dialog box. Then click the green Go 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 will change it to WS\_SQLInsert. Your finished Add Web reference dialog box should match the one shown in Figure 8.71.

Click the Add Reference button to finish this Web reference adding process. Immediately you will 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 Course.aspx include three steps:

1. Add a Course ID text box and its associated label into this page since we need this control to insert a new course\_id in this application. Add a new text box, name it txtCourseID, and label it with the Text property “Course ID”. The position of this text box and label is above the Course text box.

Also set the AutoPostBack property of this text box to True. This is very important since when the content of this text box is changed as the project runs, a TextChanged event occurs, but this event only occurs in the client side, not the server side. Our Web-based client project is running on a Web server or on the server side, so this event cannot be responded to by the server and therefore the command inside this event procedure cannot be executed (the Insert button cannot be enabled) even if the content of the Course ID text box is changed when the project runs. To solve this problem, we must set the AutoPostBack property of this text box to True to allow it send back a TextChanged event to the client automatically when the content of this text box 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 “ComboMethod” and label it with the Text property “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 Name of the Course text box from txtCourse to txtCourseName, change the Name of the Credit text box from txtCredit to txtCredits, and change the Name of the Enrollment text box from “txtEnrollment” to “txtEnroll”. Your modified Course.aspx Web form window is shown in Figure 8.72.

Go to the File|Save All menu item to save these modifications.

#### **8.4.5.3 Modify the Code for the Related Event Procedures**

The first modification is to change the code in the Page\_Load event procedure and some global variables.



Figure 8.72. The modified Course page window.

#### 8.4.5.3.1 Modify the Code 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(4) and CourseID() since we do not need them 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 inside the Page\_Load event procedure and the associated global connection object that is stored in the Application state Application("sqlConnection").
5. Add the code to add and display two Web methods in the ComboMethod combo box control.

Your finished code for the Page\_Load event procedure should match that shown in Figure 8.73. The modifications are highlighted in bold.

The next step is to develop the code for the Insert button's click event procedure.

(Page Events)	▼	Load	▼
Imports System.Data			
Partial Class Course			
Inherits System.Web.UI.Page			
<b>Private dsFlag As Boolean</b>			
<b>Private wsDataSet As New DataSet</b>			
<b>Private wsSQLResult As New WS_SQLInsert.SQLInsertBase</b>			
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load			
If Not IsPostBack Then			
ComboName.Items.Add("Ying Bai")			
ComboName.Items.Add("Satish Bhalla")			
ComboName.Items.Add("Black Anderson")			
ComboName.Items.Add("Steve Johnson")			
ComboName.Items.Add("Jenney King")			
ComboName.Items.Add("Alice Brown")			
ComboName.Items.Add("Debby Angles")			
ComboName.Items.Add("Jeff Henry")			
<b>ComboMethod.Items.Add("Stored Procedure Method")</b>			
<b>ComboMethod.Items.Add("DataSet Method")</b>			
End If			
End Sub			

**Figure 8.73.** The modified Page\_Load event procedure.

#### 8.4.5.3.2 Develop Code for the Insert Button Event Procedure

The functionality of this piece of code is to insert a new course record that is stored in six text boxes in the Web page into the database as this Insert button is clicked. This code is basically identical to that in the same event procedure of the Windows-based client project we developed in the last section. Therefore we can copy the code from that event procedure and paste it into our current procedure with a few modifications.

Open the Windows-based client project WinClientSQLInsert from the folder **DBProjects\Chapter 8** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) 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.

The only modification to this event procedure is to add one more String variable, **errMsg**, which 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 the **MsgBox()** can only be used in the client side.

Your finished code for the Insert button's click event procedure should match that shown in Figure 8.74. The modifications are highlighted in bold.

```

Protected Sub cmdInsert_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdInsert.Click
    Dim wsSQLInsert As New WS_SQLInsert.WebServiceSQLInsert
    Dim errMsg As String

    A If ComboMethod.Text = "Stored Procedure Method" Then
        Try
            wsSQLResult = wsSQLInsert.SetSQLInsertSP(ComboName.Text, txtCourseID.Text, _
                txtCourseName.Text, txtSchedule.Text, txtClassRoom.Text, _
                txtCredits.Text, txtEnroll.Text)
        Catch err As Exception
            errMsg = "Web service is wrong: " & err.Message
            Response.Write("<script>alert('" + errMsg + "')</script>")
        End Try
        If wsSQLResult.SQLInsertOK = False Then
            Response.Write("<script>alert('" + wsSQLResult.SQLInsertError + "')</script>")
        End If
    B Else
        C     dsFlag = True          'indicate the DataSet insert is performed
        D     Application("dsFlag") = dsFlag   'reserve this flag as a global flag
        Try
            wsDataSet = wsSQLInsert.SQLInsertDataSet(ComboName.Text, txtCourseID.Text, _
                txtCourseName.Text, txtSchedule.Text, txtClassRoom.Text, _
                txtCredits.Text, txtEnroll.Text)
        Catch err As Exception
            errMsg = "Web service is wrong: " & err.Message
            Response.Write("<script>alert('" + errMsg + "')</script>")
        End Try
    E End If
    F     Application("wsDataSet") = wsDataSet  'reserve the global DataSet
    cmdInsert.Enabled = False
End Sub

```

Figure 8.74. The code for the Insert button event procedure.

Let's have a quick review of this piece of code to see how it works.

- A. If the user selects the Stored Procedure method to perform this data insertion, the Web method SetSQLInsertSP(), which is developed in the Web Service, is executed to call the associated stored procedure to insert a new course record into our sample database. Any error encountered during the execution of this Web method will be displayed and reported.
- B. If the user chooses 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.
- 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 functionalities: insert data into the database and retrieve data from the database. But in order to perform the data retrieval using this method, we must first insert data using this method. Otherwise no data can be retrieved if no data insertion is performed first 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, and therefore all global variables' values will also be refreshed when a refreshed page is sent back. But 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 reason 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 the Select button. Otherwise, the contents of this DataSet will be refreshed each time a refreshed Course page is sent back.
- F. Finally, the Insert button is disabled to avoid duplicate insertion of the same data into the database.

#### **8.4.5.3.3 Develop Code for TextChanged Event Procedure of the CourseID Text Box**

The coding for this event procedure is very simple. Open this event procedure by double-clicking the Course ID text box 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 duplicate insertion of the same record into the database. But when 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 records and a new course record, the content of the Course ID text box 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 text box has changed, which means that a new course record is ready to be inserted, the Insert button should be enabled.

Another important point is to make sure that the AutoPostBack property of this Course ID text box is set to True to allow it send back a TextChanged event to the client when its content is changed.

#### 8.4.5.3.4 Modify the Code in the Select Button's Click Event Procedure

The code in this event procedure is similar to the code we developed in the same event procedure in our Windows-based client project WinClientSQLInsert, so we can copy that code and paste it into our current Select button's click event procedure with a few modifications. Open the Select button's click event procedure from our Windows-based client project WinClientSQLInsert, copy the code, and paste it into our Select button's click event procedure. The only modifications to this piece of code are to change the Windows-based message box function MsgBox() to the Web-based message box function. The code for this event procedure is shown in Figure 8.75. The modifications are highlighted in bold.

Let's have a quick review of this piece of code to see how it works.

- A. An instance of our Web Service reference WebServiceSQLInsert is created first, and this instance works as a bridge to connect this client project with associated Web methods developed in the Web Service. Also, an errMsg

```

cmdSelect Click

Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim wsSQLInsert As New WS_SQLInsert.WebServiceSQLInsert
    Dim errMsg As String

    If ComboMethod.Text = "Stored Procedure Method" Then
        Try
            wsSQLResult = wsSQLInsert.GetSQLInsert(ComboName.Text)
        Catch err As Exception
            errMsg = "Web service is wrong: " & err.Message
            Response.Write("<script>alert('" + errMsg + "')</script>")
        End Try
        If wsSQLResult.SQLInsertOK = False Then
            Response.Write("<script>alert('" + wsSQLResult.SQLInsertError + "')</script>")
        End If
        ProcessObject(wsSQLResult)
    Else
        If Application("dsFlag") = False Then
            errMsg = "No DataSet Insert performed, do data insertion first"
            Response.Write("<script>alert('" + errMsg + "')</script>")
        Exit Sub
        End If
        Call FillCourseDataSet(Application("wsDataSet"))
        Application("dsFlag") = False
    End If
End Sub

```

Figure 8.75. The code for the Select button event procedure.

- string variable is created and 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 member based on the input faculty name. If any error occurs 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 item SQLInsertOK that is defined in the base class SQLInsertBase in our Web Service project.
  - D. If no error is detected, the user-defined subroutine ProcessObject(), whose code is shown below, 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 selects the DataSet method, we first need to check the dsFlag stored in an Application state to make sure that the Web method SQLInsertDataSet() has been performed once since our current data query needs to extract all course\_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, we do not have any returned DataSet available. A warning message is displayed and the procedure is exited if that situation occurs.
  - 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 from the last calling of the Web method 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 when calling this subroutine.
  - G. Finally, the dsFlag stored in an Application state is reset to False.

The detailed code for the subroutines ProcessObject() and FillCourseListBox() is shown in Figure 8.76.

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

- A. A local string variable errMsg is declared and it is used to hold any error message to be displayed and reported later.
- B. We need to check the member data item 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 page.
- C. A warning message is displayed if any error is encountered during the execution of that Web method.
- D. In the subroutine FillCourseListBox(), a local integer variable index is created and 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.

	Course	▼	ProcessObject	▼
--	--------	---	---------------	---

```

Private Sub ProcessObject(ByRef wsResult As WS_SQLInsert.SQLInsertBase)
A   Dim errMsg As String
B   If wsResult.SQLInsertOK = True Then
        Call FillCourseListBox(wsResult)
    Else
C       errMsg = "Course information cannot be retrieved: " & wsResult.SQLInsertError
        Response.Write("<script>alert('" + errMsg + "')</script>")
    End If
End Sub

Private Sub FillCourseListBox(ByRef sqlResult As WS_SQLInsert.SQLInsertBase)
D   Dim index As Integer
E   CourseList.Items.Clear()                               'clean up the course listbox
F   For index = 0 To sqlResult.CourseID.Length - 1
        If sqlResult.CourseID(index) <> vbNullString Then
            CourseList.Items.Add(sqlResult.CourseID(index))
        End If
    Next index
End Sub

```

Figure 8.76. The code for subroutines ProcessObject and FillCourseListBox.

- E. The course list box control is cleaned up first before any course\_id can be added into it. This process is very important when displaying all course\_id; otherwise, any new course\_id would be attached to the end of the original course\_id in this control and the resulting display would be 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 note is the upper bound and the length of this array. The length or 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, that element or course\_id is added into the list box control by using the Add() method.

The code for the subroutine FillCourseDataSet() is shown in Figure 8.77. This coding is identical to that of the same subroutine we developed in our Windows-

	Course	▼	FillCourseDataSet	▼
--	--------	---	-------------------	---

```

Private Sub FillCourseDataSet(ByRef ds As DataSet)
A   Dim CourseTable As New DataTable
B   Dim CourseRow As DataRow
C   CourseList.Items.Clear()                           'clean up the course listbox
D   CourseTable = ds.Tables("Course")
        For Each CourseRow In CourseTable.Rows
            CourseList.Items.Add(CourseRow(0))           'the 1st column is course_id
        Next
End Sub

```

Figure 8.77. The code for the subroutine FillCourseDataSet.

based client project WinClientSQLInsert. You can copy it and paste it into our current project.

Let's have a look at the code in this subroutine to see how it works.

- A. Two objects, a DataSet 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 display 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 note 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 code in the SelectedIndexChanged event procedure and add the fourth subroutine FillCourseDetail(). Before we can continue to do these jobs, we first need to delete the following procedures and subroutines from our current project:

- FillCourseReader()
- FillCourseReaderTextBox()
- MapCourseTable()

Now let's continue to the next step.

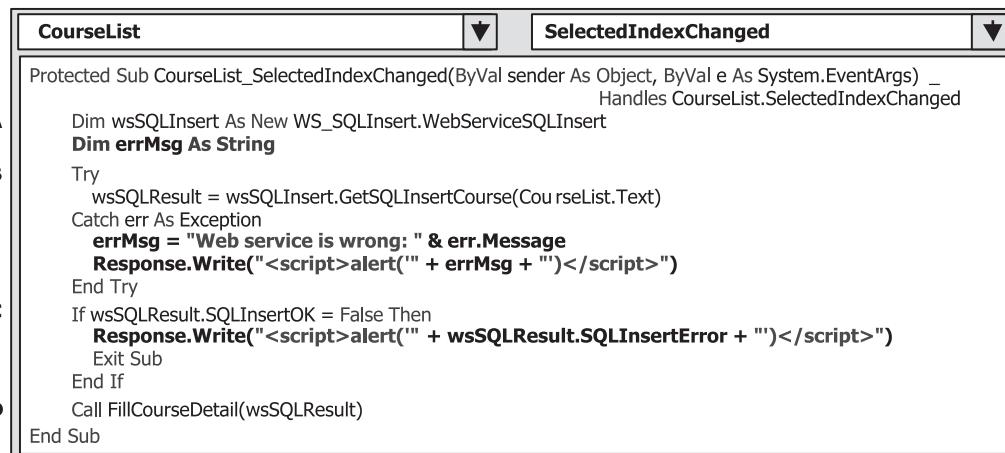
#### **8.4.5.3.5 Modify the Code in the SelectedIndexChanged Event Procedure**

The functionality of this code is that the detailed course information such as the course name, schedule, classroom, credit, and enrollment will be displayed in the associated text box controls when the user clicks and selects one course from the list box control. The main coding job is performed inside the SelectedIndexChanged event procedure of the list box control because when user clicks or selects a course 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, the Web method GetSQLInsertCourse() we developed in our Web Service project Web-ServiceSQLInsert 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 coding in this event procedure is identical to that of the same event procedure in our Windows-based client project WinClientSQLInsert, so we can copy the code from that event procedure and paste it into our current project with a few modifications.

Double-click the CourseList list box control from our client page window to open the SelectedIndexChanged event procedure of the list box control, and copy and paste the code from the same event procedure in our Windows-based project.



The screenshot shows the 'CourseList' code editor window. The title bar says 'CourseList' and the tab bar says 'SelectedIndexChanged'. The code is as follows:

```

Protected Sub CourseList_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles CourseList.SelectedIndexChanged
    Dim wsSQLInsert As New WS_SQLInsert.WebServiceSQLInsert
    Dim errMsg As String
    Try
        wsSQLResult = wsSQLInsert.GetSQLInsertCourse(CourseList.Text)
    Catch err As Exception
        errMsg = "Web service is wrong: " & err.Message
        Response.Write("<script>alert('" + errMsg + "')</script>")
    End Try
    If wsSQLResult.SQLInsertOK = False Then
        Response.Write("<script>alert('" + wsSQLResult.SQLInsertError + "')</script>")
        Exit Sub
    End If
    Call FillCourseDetail(wsSQLResult)
End Sub

```

Line A: 'Dim wsSQLInsert As New WS\_SQLInsert.WebServiceSQLInsert' and 'Dim errMsg As String' are bolded.

Line B: 'Try' and 'Catch err As Exception' are bolded.

Line C: 'errMsg = "Web service is wrong: " & err.Message' and 'Response.Write("<script>alert('" + errMsg + "')</script>")' are bolded.

Line D: 'Call FillCourseDetail(wsSQLResult)' is bolded.

**Figure 8.78.** The modified code for the SelectedIndexChanged event procedure.

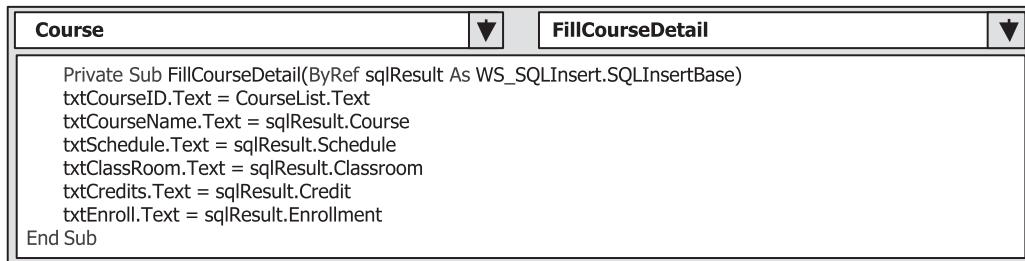
The only modifications are the `MsgBox()` methods, which should be replaced by the script message method `alert()`. Your finished SelectedIndexChanged event procedure should match the one shown in Figure 8.78. The modifications are highlighted in bold.

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

- 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. Also, a local string variable, `errMsg`, is declared here, and it is used to hold the error message to be displayed and reported later.
- 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 retrieval. The selected course.id is stored in the Text property of the `CourseList` control. An exception message is displayed if any error is encountered during the execution of this Web method and caught by the system method `Catch`.
- In addition to the error checking performed by the system, we also need to perform our exception checking by inspecting the member data `SQLInsertOK` in the base class `SQLInsertBase`. If this data value is False, which means that an application error has occurred during the running of this Web method, an error message is displayed and the subroutine is exited.
- 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 text box control in our client page form.

The code for the subroutine `FillCourseDetail()` is shown in Figure 8.79.

This piece of code is identical to that in the same subroutine in our Windows-based client project `WinClientSQLInsert`. You can copy it and paste it into this project.



```

Course
FillCourseDetail

Private Sub FillCourseDetail(ByRef sqlResult As WS_SQLInsert.SQLInsertBase)
    txtCourseID.Text = CourseList.Text
    txtCourseName.Text = sqlResult.Course
    txtSchedule.Text = sqlResult.Schedule
    txtClassRoom.Text = sqlResult.Classroom
    txtCredits.Text = sqlResult.Credit
    txtEnroll.Text = sqlResult.Enrollment
End Sub

```

**Figure 8.79.** The code for the subroutine FillCourseDetail.

The functionality of this code is straightforward and without tricks. Each piece of course information is extracted from the returned instance and assigned to the associated text box control in our client page window.

#### 8.4.5.3.6 Modify the Code in the Back Button's Click Event Procedure

The final modification is to change the code 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 code with the following code 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 run this project to test the data insertion and validation functionalities, make sure that the following jobs have been performed:

- Our main Web page Course.aspx has been set as the starting page. This can be done by right-clicking our main Web page and select the item Set As Start Page from the popup menu.
- Our Web Service WebServiceSQLInsert is in open status. This can be checked by locating a small white icon on the status bar at the bottom of the screen. If you cannot find this icon, open the Web Service project and click the Start Debugging button to run it. After the Web Service starts to run, you can close it if you like, but it remains in the open status.
- The two new course records, CSE-665 and CSE-526, that we inserted before to test the Insert button's click event procedure should have been deleted since we want to insert the same course records in this test.

Now click the Start Debugging button to run our client project. First, let's test the data insertion functionality. Select the Stored Procedure method from the Method combo box control, select the default faculty, Ying Bai, from the Faculty Name combo box control, enter the first new course's information (as shown in Table 8.2) into the associated text boxes, and then click the Insert button. Perform a similar

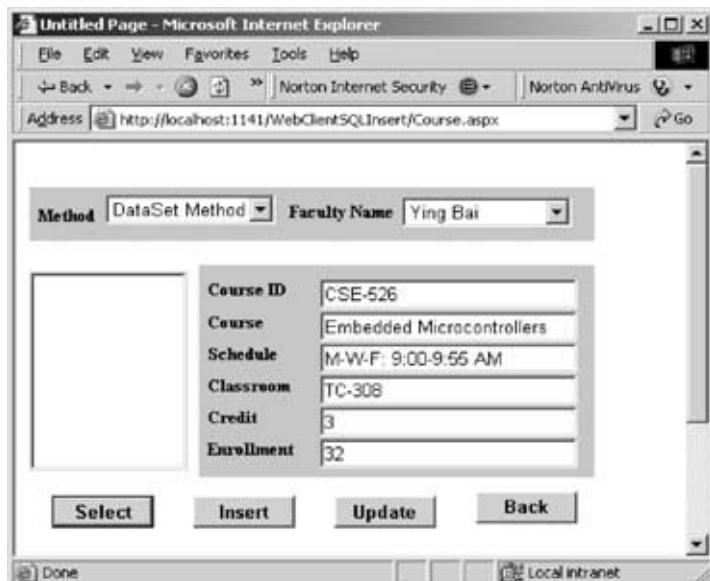


Figure 8.80. The running status of inserting new course records.

operation to insert the second new course's information (as shown in Table 8.3) with the DataSet method selected. Your running Web page is shown in Figure 8.80.

To validate these data insertions, click the Select button for the DataSet method and then the Stored Procedure method, respectively. The running result is shown in Figure 8.81.

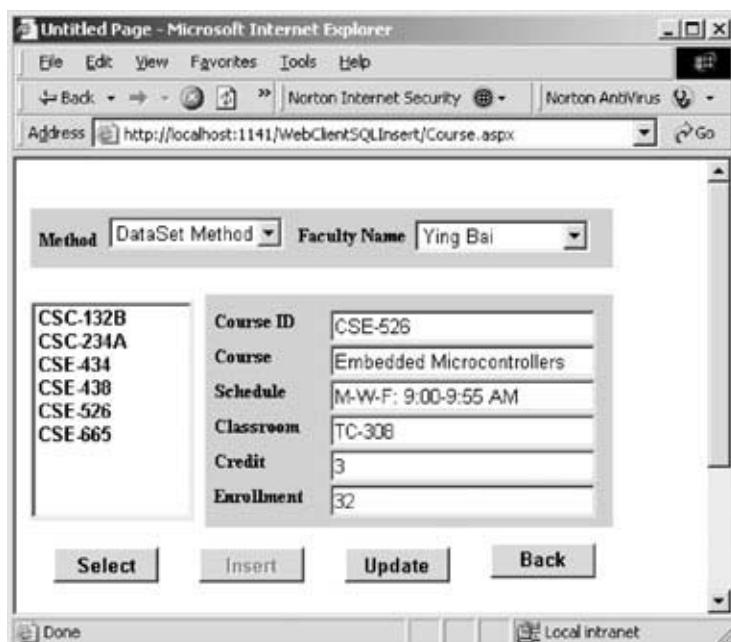


Figure 8.81. The running status of the data validation process.

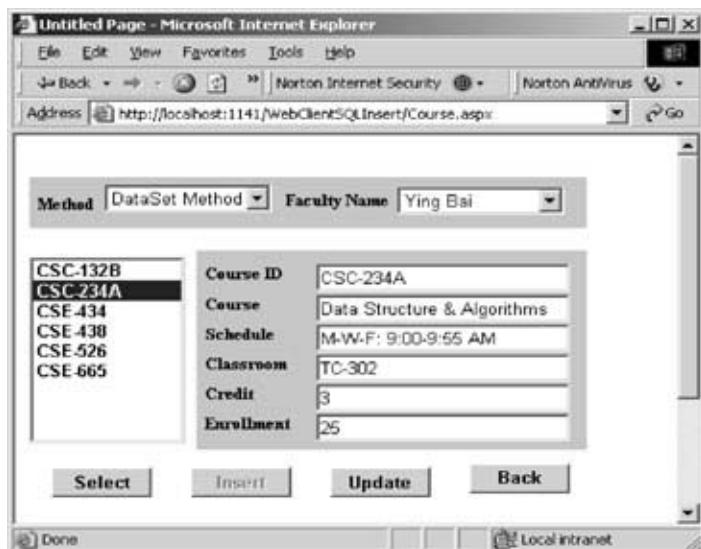


Figure 8.82. The running status of getting the detailed course information.

You can find that our two newly inserted courses, CSE-665 and CSE-526, have been added into and retrieved from our database, and are displayed in the list box control.

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 in the associated text boxes, as shown in Figure 8.82.

You can try to get the detailed information for a different course by selecting a different course\_id from the list box control via either the DataSet or the Stored Procedure method. Click the Back button to terminate our Web client project.

The completed Web-based Web Service client project WebClientSQLInsert is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

Next, we need to take care of updating and deleting data via Web Services.

## **8.5 BUILD AN ASP.NET WEB SERVICE TO UPDATE AND DELETE DATA IN AN SQL SERVER DATABASE**

In this section we discuss how to update and delete a record from the Course table in our sample database via 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 the existing Web Service project WebClientSQLInsert we developed in Section 8.4 to make it into our new Web Service project WebServiceSQLUpdateDelete.

### **8.5.1 Modify an Existing Web Service Project**

Open the Internet Explorer, browse to the folder **DBProjects\Chapter 8** at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) and select the Web Service project

WebServiceSQLInsert. Copy this project and paste it into our development folder, **C:\Chapter 8**, in our computer. Rename this project **WebServiceSQLUpdateDelete**.

Open Visual Studio.NET 2005 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 this class file and selecting the **Rename** item from the popup menu.
2. In a similar way, 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 our new Web main page **WebServiceSQLUpdateDelete.asmx** to open it, and make the following changes to the top line:
  - a. From: **CodeBehind = “~/App\_Code/WebServiceSQLInsert.vb”**  
To: **CodeBehind = “~/App\_Code/WebServiceSQLUpdateDelete.vb”**
  - b. From: **Class = “WebServiceSQLInsert”**  
To: **Class = “WebServiceSQLUpdateDelete”**
5. Double-click our new base class file **SQLBase** and perform the following modifications to the class name and the first two member data items:
  - 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 our new code-behind page **WebServiceSQLUpdateDelete.vb** from the Solution Explorer window to open it. Change the 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.

## 8.5.2 Modify Related Web Methods

These modification include:

1. Remove the Web method **SQLInsertDataSet()** from this project since we do not need this method to perform the data updating or deleting actions.
2. Modify the Web method **SetSQLInsertSP()** to make it our new Web method **SQLUpdateSP()**. This method will call a stored procedure to perform the data updating for our Course table.
3. Modify the Web method **GetSQLInsert()** to make it our new Web method **GetSQLCourse()** 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 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 named SQLDeleteSP(). This method will delete a course record based on the input course\_id.

Now let's detail these modifications starting from step 2.

### **8.5.2.1 Modify the Web Method from SetSQLInsertSP to SQLUpdateSP**

The functionality of this Web method is to call a stored procedure dbo.WebUpdateCourseSP() that will be developed in Section 8.5.3.1 to perform the data updating for a course record based on the course\_id.

Normally we do not 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 a record with a new primary key. Another reason for this is that it would be very complicated to update a primary key in a table since it may be related to many other child tables using the foreign keys, and therefore one would have to update those foreign keys in many child tables before the primary key could be updated in the parent table.

Open our new Web Service project WebServiceSQLUpdateDelete and the Web method SetSQLInsertSP(), and then perform the modifications shown in Figure 8.83 to this method. The modifications are highlighted in bold.

Let's have a closer look at the modified code to see how it works.

- 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 8.5.3.1. Keep in mind that this name must be identical to the name used for the stored procedure to be developed later.
- C. An instance of our modified base class SQLBase, SQLResult, is created, and this instance contains the running status of this Web method and will be returned to the calling procedure.
- D. A local integer variable, intUpdate, is declared here and is used to hold the value returned from calling the ExecuteNonQuery() method.
- E. We set a good running status of this Web method to the member data item SQLOK to indicate that so far our Web method is running fine.
- F. If an error is encountered during the database connection process, the error information is stored into the member data item SQLError and reported using the subroutine ReportError().
- G. The Command object is initialized with associated data objects such as connection object, command text, and command type. One point to note 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

WebServiceSQLUpdateDelete	▼	SQLUpdateSP	▼
<pre> &lt;WebMethod()&gt; Public Function <b>SQLUpdateSP</b>(ByVal FacultyName As String, ByVal CourseID As String, ByVal Course As String, _     ByVal Schedule As String, ByVal Classroom As String, ByVal Credit As Integer, ByVal Enroll As Integer) As _         <b>SQLBase</b>  <b>A</b>    Dim cmdString As String = "dbo.WebUpdateCourseSP" <b>B</b>    Dim sqlConnection As New SqlConnection <b>C</b>    Dim <b>SQLResult</b> As New <b>SQLBase</b> <b>D</b>    Dim sqlCommand As New SqlCommand <b>E</b>    Dim <b>intUpdate</b> As Integer <b>F</b>    <b>SQLResult.SQLOK</b> = True <b>G</b>    sqlConnection = SQLConn() <b>H</b>    If sqlConnection Is Nothing Then <b>I</b>        <b>SQLResult.SQLError</b> = "Database connection is failed" <b>J</b>        ReportError(<b>SQLResult</b>) <b>K</b>        Return Nothing <b>L</b>    End If <b>M</b>    sqlCommand.Connection = sqlConnection <b>N</b>    sqlCommand.CommandType = CommandType.StoredProcedure <b>O</b>    sqlCommand.CommandText = cmdString <b>P</b>    sqlCommand.Parameters.Add("@FacultyName", SqlDbType.Text).Value = FacultyName <b>Q</b>    sqlCommand.Parameters.Add("@CourseID", SqlDbType.Char).Value = CourseID <b>R</b>    sqlCommand.Parameters.Add("@Course", SqlDbType.Text).Value = Course <b>S</b>    sqlCommand.Parameters.Add("@Schedule", SqlDbType.Char).Value = Schedule <b>T</b>    sqlCommand.Parameters.Add("@Classroom", SqlDbType.Text).Value = Classroom <b>U</b>    sqlCommand.Parameters.Add("@Credit", SqlDbType.Int).Value = Credit <b>V</b>    sqlCommand.Parameters.Add("@Enroll", SqlDbType.Int).Value = Enroll <b>W</b>    <b>intUpdate</b> = sqlCommand.ExecuteNonQuery() <b>X</b>    sqlCommand.Dispose() <b>Y</b>    sqlCommand = Nothing <b>Z</b>    If Not sqlConnection Is Nothing Then sqlConnection.Close() <b>A</b>    sqlConnection = Nothing <b>B</b>    If <b>intUpdate</b> = 0 Then <b>C</b>        <b>SQLResult.SQLError</b> = "Data updating is failed" <b>D</b>        ReportError(<b>SQLResult</b>) <b>E</b>    End If <b>F</b>    Return <b>SQLResult</b> <b>G</b> End Function </pre>			

Figure 8.83. The modified code for the Web method SQLUpdateSP.

updating parameters to the associated dynamic parameter in the UPDATE command.

- H. The ExecuteNonQuery() method is executed to call the stored procedure to perform the data updating. An integer value will be returned from this method, and the value of this integer is equal to the number of rows that have been successfully updated in our Course table.
- I. A cleaning job is performed to release all objects used in this method.
- J. If the value returned from calling the ExecuteNonQuery() method is zero, which means that no row has been updated in our Course table and this data updating has failed, an error message is sent to the member data SQLError and reported using the subroutine ReportError().
- K. Finally, the instance SQLResult that contains the running status of this Web method is returned to the calling procedure.

Go to File|Save All to save these modifications.

```

WebServiceSQLUpdateDelete           GetSQLCourse

```

```

<WebMethod()>_
Public Function GetSQLCourse(ByVal FacultyName As String) As SQLBase
    Dim cmdString As String = "SELECT Course.course_id FROM Course JOIN Faculty " + _
        "ON (Course.faculty_id LIKE Faculty.faculty_id) AND (Faculty.name LIKE @name)"
    Dim sqlConnection As New SqlConnection
    Dim SQLResult As New SQLBase
    Dim sqlCommand As New SqlCommand
    Dim sqlReader As SqlDataReader
    SQLResult.SQLOK = True
    sqlConnection = SQLConn()
    If sqlConnection Is Nothing Then
        SQLResult.SQLError = "Database connection is failed"
        ReportError(SQLResult)
        Return Nothing
    End If
    sqlCommand.Connection = sqlConnection
    sqlCommand.CommandType = CommandType.Text
    sqlCommand.CommandText = cmdString
    sqlCommand.Parameters.Add("@name", SqlDbType.Text).Value = FacultyName
    sqlReader = sqlCommand.ExecuteReader
    If sqlReader.HasRows = True Then
        Call FillCourseReader(SQLResult, sqlReader)
    Else
        SQLResult.SQLError = "No matched course found"
        ReportError(SQLResult)
    End If
    If Not sqlReader Is Nothing Then sqlReader.Close()
    sqlReader = Nothing
    If Not sqlConnection Is Nothing Then sqlConnection.Close()
    sqlConnection = Nothing
    sqlCommand.Dispose()
    Return SQLResult
End Function

```

Figure 8.84. The modified code for the Web method GetSQLCourse.

### 8.5.2.2 Modify the Web Method GetSQLInsert to GetSQLCourse

The functionality of this Web method is to retrieve all course\_id, including the original and updated course\_id, and assign them to a CourseID array in our base class SQLBase that will be returned as an instance to the calling procedure. A client project will extract all course\_id from this returned instance and display them in a list box control.

Open this Web method and perform the modifications shown in Figure 8.84 to this method. The modifications are highlighted in bold.

Let's take a closer look at the modified code to see how it works.

- 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. We set 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 an error is encountered during the database connection process, the error information is stored in the member data item SQLError and reported using the 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 @name is assigned with the actual faculty name that is an input parameter to this method.
- F. After the ExecuteReader() method is called to perform this data query, we need to check the status of the property HasRows. If this property is True, which means that at least one row has been retrieved from the Course table, the 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 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 subroutine FillCourseReader() is to change the data type of the first input argument, sResult, from SQLInsertBase to SQLBase.

#### **8.5.2.3 Modify the Web Method GetSQLInsertCourse to GetSQLCourseDetail**

The functionality of this Web method is to retrieve the detailed information for a specific course\_id that works as an input parameter to this method. The stored procedure WebSelectCourseSP, which we developed in Section 8.4.3.4.1, is called to perform this data query as this Web method is executed.

Open this Web method and perform the modifications shown in Figure 8.85 to this method. The modifications are highlighted in bold.

Let's have a closer look at the modified code to see how it works.

- 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. We set 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 an error is encountered during the database connection process, the error information is stored into the member data SQLError and reported using the subroutine ReportError().

WebServiceSQLUpdateDelete		GetSQLCourseDetail	
---------------------------	--	--------------------	--

```

<WebMethod()>_
Public Function GetSQLCourseDetail(ByVal CourseID As String) As SQLBase
    Dim cmdString As String = "dbo.WebSelectCourseSP"
    Dim sqlConnection As New SqlConnection
    Dim SQLResult As New SQLBase
    Dim sqlReader As SqlDataReader
    SQLResult.SQLOK = True
    sqlConnection = SQLConn()
    If sqlConnection Is Nothing Then
        SQLResult.SQLError = "Database connection is failed"
        ReportError(SQLResult)
        Return Nothing
    End If
    Dim sqlCommand = New SqlCommand(cmdString, sqlConnection)
    sqlCommand.CommandType = CommandType.StoredProcedure
    sqlCommand.Parameters.Add("@CourseID", SqlDbType.Text).Value = CourseID
    sqlReader = sqlCommand.ExecuteReader
    If sqlReader.HasRows = True Then
        Call FillCourseDetail(SQLResult, sqlReader)
    Else
        SQLResult.SQLError = "No matched course found"
        ReportError(SQLResult)
    End If
    sqlReader.Close()
    sqlReader = Nothing
    sqlConnection.Close()
    sqlCommand.Dispose()
    Return SQLResult
End Function

```

**Figure 8.85.** The modified code for the Web method GetSQLCourseDetail.

- E. The Command object is initialized with associated data objects and properties such as connection object, command text, and command type. Also, the dynamic parameter @CourseID is assigned with the actual CourseID that is an input parameter to this method.
- F. After the ExecuteReader() method is called to perform this data query, we need to check the status of the property HasRows. If this property is True, which means that at least one row has been retrieved from the Course table, the 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. If this property is False, which means that no row has been retrieved from the Course table, an error message is displayed and reported using the 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 subroutine FillCourseDetail() is to change the data type of the first input argument, sResult, from SQLInsertBase to SQLBase.

The last modification to this Web method is to modify the 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.

#### **8.5.2.4 Add a New Web Method – SQLDeleteSP**

As discussed in Section 6.1.1 in Chapter 6, to delete a record from a relational database, one needs to follow the steps listed below:

1. Delete records that are related to the parent table using the foreign keys from child tables.
2. Delete records that are defined as primary keys from the parent table.

In other words, to delete one record from the parent table, all records that are related to that record as foreign keys and located at different child tables must be deleted first. In our case, in order to delete a record using the course\_id as the primary key from the Course table (parent table), one must first delete those records using the course\_id as a foreign key from the StudentCourse table (child table). Fortunately we have only one child table related to our parent table in our sample database. Refer to Section 2.5 and Figure 2.5 in Chapter 2 to get a clear description of the relationships among the 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 deletion queries need to be performed: the first query is used to delete the related records from the child table, the StudentCourse table, and the second query is used to delete the target record from the parent table, the Course table. To save time and space as well as for 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 as the primary key. At this moment, we just assume that we have already finished developing that stored procedure and will use it in this Web method.

Open our Web Service project and our code-behind page WebServiceSQLUpdateDelete.vb and create the Web method SQLDeleteSP(), which is shown in Figure 8.86.

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

- A. The name of this Web method is SQLDeleteSP, and the returned data type is our modified base class SQLBase.
- B. The content of the query string is equal to the name of the stored procedure we will develop soon. The point is that the name used in this query string must be identical to 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.

A	WebServiceSQLUpdateDelete	▼	SQLDeleteSP	▼
<WebMethod()> _				
A	Public Function SQLDeleteSP(ByVal CourseID As String) As SQLBase			
B	Dim cmdString As String = "dbo.WebDeleteCourseSP"			
C	Dim sqlConnection As New SqlConnection			
D	Dim SQLResult As New SQLBase			
E	Dim intDelete As Integer			
F	SQLResult.SQLOK = True			
G	sqlConnection = SQLConn()			
H	If sqlConnection Is Nothing Then			
I	SQLResult.SQLOK = False			
J	SQLResult.SQLError = "Database connection is failed"			
K	ReportError(SQLResult)			
L	Return Nothing			
M	End If			
N	Dim sqlCommand = New SqlCommand(cmdString, sqlConnection)			
O	sqlCommand.CommandType = CommandType.StoredProcedure			
P	sqlCommand.Parameters.AddWithValue("@CourseID", SqlDbType.Text).Value = CourseID			
Q	intDelete = sqlCommand.ExecuteNonQuery()			
R	If intDelete = 0 Then			
S	SQLResult.SQLError = "Data deleting is failed"			
T	ReportError(SQLResult)			
U	End If			
V	sqlConnection.Close()			
W	sqlCommand.Dispose()			
X	sqlCommand = Nothing			
Y	Return SQLResult			
Z	End Function			

**Figure 8.86.** The code for the Web method SQLDeleteSP

- C. An instance of our modified base class SQLBase, SQLResult, is created. This instance contains the running status of this Web method and will be returned to the calling procedure when this method is done. Also, a local integer variable intDelete is declared, and this variable is used to hold the value returned from calling the ExecuteNonQuery() method after this method runs.
- D. We set 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 an 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 Stored-Procedure 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 value to indicate the running status of this method, and the returned value is assigned to the local integer variable intDelete.

- I. The returned value 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 for our Web Service project. Next, let's begin to develop our two stored procedures.

### **8.5.3 Develop Two Stored Procedures – WebUpdateCourseSP and WebDeleteCourseSP**

Now it is time 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 2005 environment.

#### **8.5.3.1 Develop the Stored Procedure WebUpdateCourseSP**

Open Visual Studio.NET 2005 and the Server Explorer window, and connect and expand our sample SQL Server database CSE\_DEPT.mdf to find the Stored Procedures folder. Right-click this folder and select the item Add New Stored Procedure from the popup menu to open the Add New Stored Procedure dialog box.

Enter the code shown in Figure 8.87 into this procedure to make it into our new stored procedure.

```

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 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 8.87. The code for our new stored procedure WebUpdateCourseSP

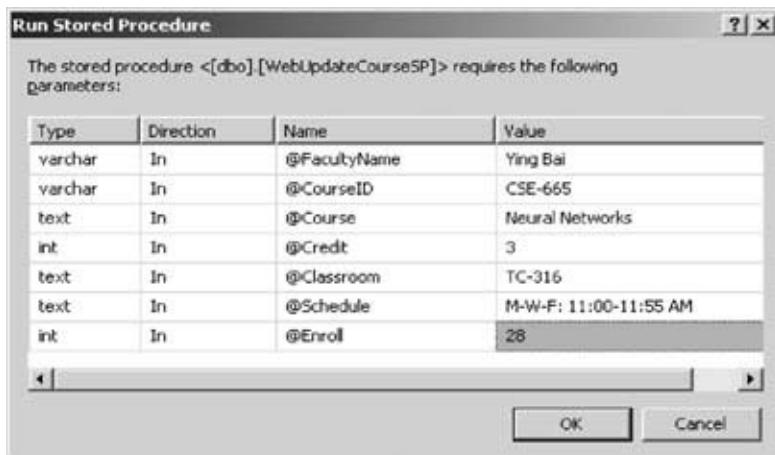


Figure 8.88. The input parameters to stored procedure WebUpdateCourseSP.

The actual name of this procedure is dbo.WebUpdateCourseSP, but generally we call this procedure WebUpdateCourseSP without the prefix dbo since this prefix is added automatically when a new SQL Server stored procedure is created.

Seven input parameters are listed in the parameter section with the related data types. Two queries are included in this procedure. The first one is used to pick up the desired faculty\_id based on the input parameter FacultyName since there is no faculty name column available in the Course table. The second query is used to perform the data updating based on another six input parameters with the course\_id as the dynamic parameter.

Go to the 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 our newly created stored procedure from the Server Explorer window and select the Execute item from the popup menu to open the Run Stored Procedure dialog box, which is shown in Figure 8.88.

Enter the parameters shown in Table 8.4 into the Value box as the input parameters (refer to Figure 8.88).

**Table 8.4. 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

The screenshot shows the 'Output' window from Visual Studio. The title bar says 'Output'. Under 'Show output from:', 'Database Output' is selected. The main pane displays the following text:

```
Running [dbo].[WebUpdateCourseSP] (@FacultyName = Ying Bai, @CourseID = CSE-665, @Course = Advanced Fuzzy Systems, @Credit = 3, @Classroom = TC-315, @Schedule = T-H: 1:00-2:25 PM, @Enrollment = 26, @FacultyID = B78880)
(1 row(s) affected)
(0 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[WebUpdateCourseSP].
```

Figure 8.89. The running result of the stored procedure WebUpdateCourseSP.

Click the OK button to run this stored procedure, and the running result is displayed in the Output window, which is shown in Figure 8.89.

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 the 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 it. Select the item Show Table Data from the popup menu to try to find this updated course record. In the Course table you will find that this record has been updated according to the parameters we entered when this procedure was executed.

It is highly recommended to restore this updated record to its original state since we will use the same input parameters later to update this record again when we test our Web Service project. You can perform this record recovering in the Course table by entering the values shown in Table 8.5.

### 8.5.3.2 Develop the Stored Procedure WebDeleteCourseSP

Open Visual Studio.NET 2005 and the Server Explorer window, and connect and expand our sample SQL Server database CSE\_DEPT.mdf to find the Stored Procedures folder. Right-click this folder and select the item Add New Stored Procedure from the popup menu to open the Add New Stored Procedure dialog box.

Enter the code shown in Figure 8.90 into this procedure to make it into our new stored procedure.

**Table 8.5. 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

The screenshot shows the Microsoft SQL Server Management Studio interface. In the center pane, the code for the stored procedure `dbo.WebDeleteCourseSP` is displayed:

```

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

```

The code consists of two `DELETE` statements and a `RETURN` statement. The first `DELETE` statement removes records from the `StudentCourse` table where the `course_id` matches the input parameter `@CourseID`. The second `DELETE` statement removes the target course from the `Course` table with the same condition.

Figure 8.90. The code for the stored procedure WebDeleteCourseSP

The actual name of this procedure is `dbo.WebDeleteCourseSP`, but generally we call this procedure `WebDeleteCourseSP` without the prefix `dbo` since this prefix is added automatically when a new SQL Server 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 `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 `@CourseID` 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 our newly created stored procedure from the Server Explorer window and select the `Execute` item from the popup menu to open the `Run Stored Procedure` dialog box, which is shown in Figure 8.91.

Enter `CSE-526` into the `Value` box as the value of the input parameter `@CourseID`, and click the `OK` button to run this stored procedure. The running result is displayed in the Output window, which is shown in Figure 8.92.



Figure 8.91. The Run Stored Procedure dialog box.



```
Output
Show output from: Database Output
Running [dbo].[WebDeleteCourseSP] ( @CourseID = CSE-526 ).

(1 row(s) affected)
(0 row(s) returned)
@RETURN_VALUE = 0
Finished running [dbo].[WebDeleteCourseSP].
```

Figure 8.92. The running result of the stored procedure WebDeleteCourseSP.

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 deletion 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 it. Select the item Show Table Data from the popup menu to try to see whether this deleted record is still in there or not. In the Course table you will find that the course having course\_id CSE-526 has been deleted from both our Course (parent) table and StudentCourse (child) table. However, since this is a newly added course and no student has taken this course yet, you cannot find this course in 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 it yet, we do not need to recover it for the StudentCourse table. So just recover it for the Course table by adding the data shown in Table 8.6.

We have finished the development of this Web Service project. Now let's run our Web Service project to test all Web methods.

Click the Start Debugging button to run our project. The built-in Web interface window is displayed, as shown in Figure 8.93.

Four Web methods are shown in this built-in interface. First, let's test the Web method SQLUpdateSP. Click this item to open the parameter-input interface, which is shown in Figure 8.94.

**Table 8.6. 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

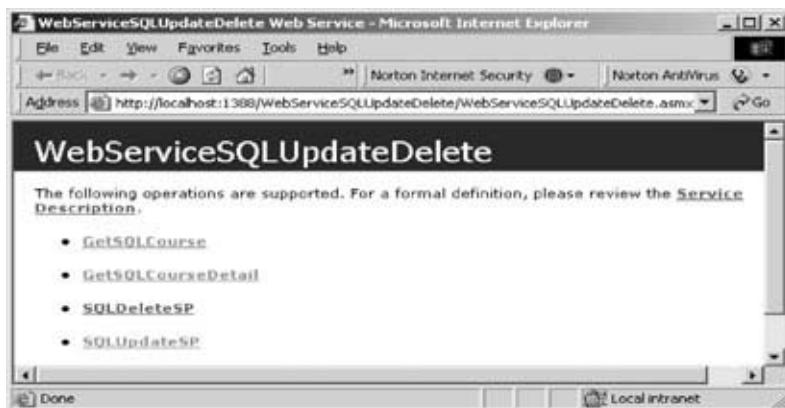


Figure 8.93. The running status of the Web Service project.

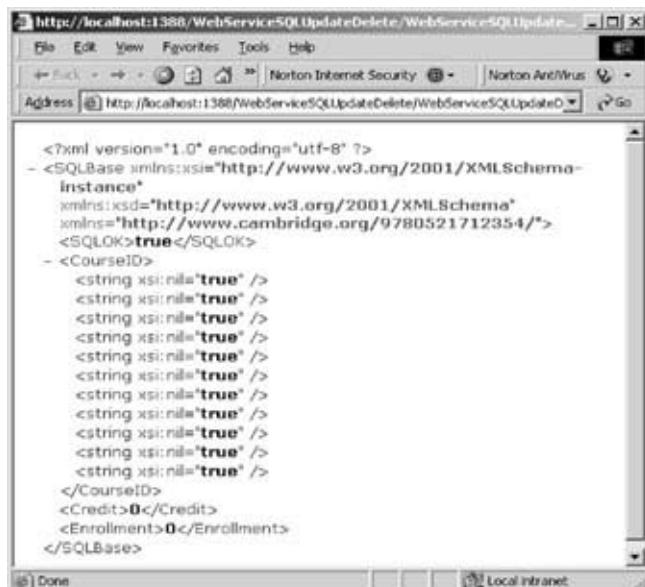
Enter the updated parameters shown in Figure 8.94 into the associated boxes to update a course with the course.id of CSE-526. Click the Invoke button to run this method.

The running result of this Web method is shown in Figure 8.95.

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 other data should be returned from the execution of this data update, all data stored in the returned instance, including

Parameter	Value
FacultyName:	Ying Bai
CourseID:	CSE-526
Course:	Advanced Control Systems
Schedule:	T-H: 9:00-10:25 AM
Classroom:	TC-330
Credit:	3
Enroll:	30

Figure 8.94. The parameter-input interface.



The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost:1388/WebServiceSQLUpdateDelete/WebServiceSQLUpdateDelete.asmx>. The page displays an XML document representing the updated data in the database. The XML structure includes a root element <SQLBase>, which contains elements for CourseID (an array of 11 elements, all set to true), Credit (0), and Enrollment (0). The XML is as follows:

```

<?xml version="1.0" encoding="utf-8" ?>
-<SQLBase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.cambridge.org/9780521712354/*">
  <SQLOK>true</SQLOK>
  -<CourseID>
    <string xsi:nil="true" />
    <string xsi:nil="true" />
  </CourseID>
  <Credit>0</Credit>
  <Enrollment>0</Enrollment>
</SQLBase>

```

Figure 8.95. The running result for the Web method SQLUpdateSP.

the CourseID() array that has 11 elements and two integers Credit and Enrollment, are either true or zero.

To confirm this data updating, now we can call other Web methods to do this job. First, we want to get back all courses (that is, all course\_id) taught by the selected faculty member. To do that, close the running result window shown in Figure 8.95 and click the Back button to return to the home page of this built-in interface. Click the Web method GetSQLCourse to run it to obtain all course\_id. Enter the faculty name Ying Bai into the Value box as the input parameter for this Web method, which is shown in Figure 8.96. Click the Invoke button to run this method.

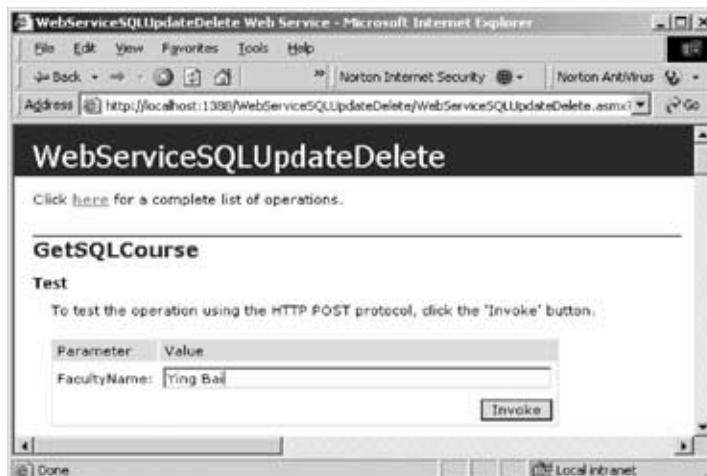


Figure 8.96. The parameter-input built-in Web interface.



Figure 8.97. The running result for the Web method GetSQLCourse.

The running result for the Web method GetSQLCourse is shown in Figure 8.97.

It can be found from Figure 8.97 that all six courses or course\_id taught by the selected faculty member are retrieved and displayed with XML tags in this built-in Web interface.

To check whether the target course CSE-526 has been updated or not, we need to run another Web method, GetSQLCourseDetail(). Close the running result window shown in Figure 8.97, and click the Back button to return to the home page of our Web Service. Click the Web method GetSQLCourseDetail to run it. Enter CSE-526 as the input parameter, as shown in Figure 8.98, to this method to pick up the detailed information for this course.

Click the Invoke button to run this method, and the running result is shown in Figure 8.99.

It can be found that the course CSE-526 has been updated based on the input parameters we entered for the Web method SQLUpdateSP() in Figure 8.94.

Next, let's test the Web method SQLDeleteSP() to try to delete a course record from the Course table. Close the current running result window shown in Figure 8.99, and click the Back button to return to the home page of the Web Service. Click the Web method SQLDeleteSP to run it, and then enter CSE-526 into the Value box for the course\_id parameter, as shown in Figure 8.100. Click the Invoke button to run this method.

The running result is shown in Figure 8.101.

It can be found that the returned running status SQLOK, which is the only returned data, is true, and this means that the data deletion is successful.

To confirm this data deletion, close the current running result interface shown in Figure 8.101 and click the Back button to return to the home page of the Web Service project. Click the Web method GetSQLCourse to run it to pick up all courses taught by the selected faculty member Ying Bai. Enter the faculty name Ying Bai

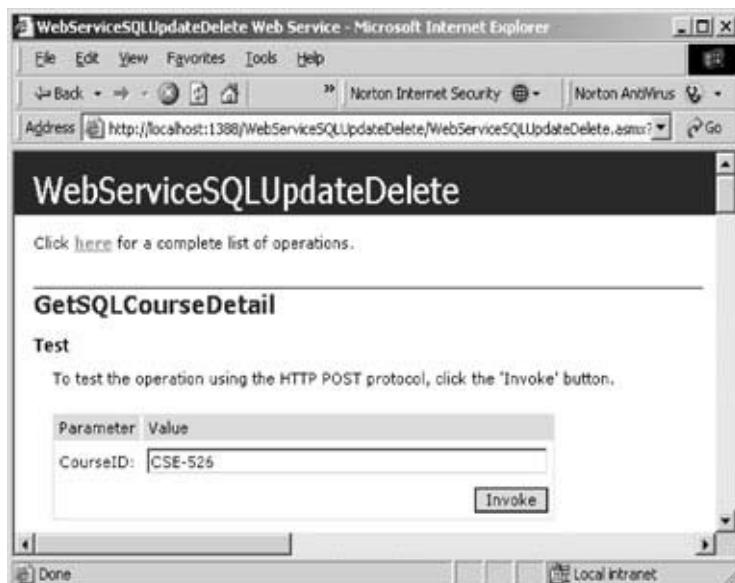


Figure 8.98. The parameter-input Web interface.

into the Value box as the input parameter for this method, and click the Invoke button to run it. The running result is shown in Figure 8.102.

From this running result shown in Figure 8.102, it can be found that the course with the course\_id as CSE-526 has been deleted from the Course table since that course is taught by the faculty member Ying Bai.

To get a clearer picture of this data deletion, let's try to run another Web method, GetSQLCourseDetail(). Close the current running result interface shown in Figure 8.102, and click the Back button to return to the home page. Select and click the Web method GetSQLCourseDetail to try to run it. Enter CSE-526 as the course\_id in the Value box as the input parameter for this method, and click the Invoke button to run it.

The running process becomes very slow. The reason is that a message box is displayed behind the top page. Move the current top page to either side of the screen

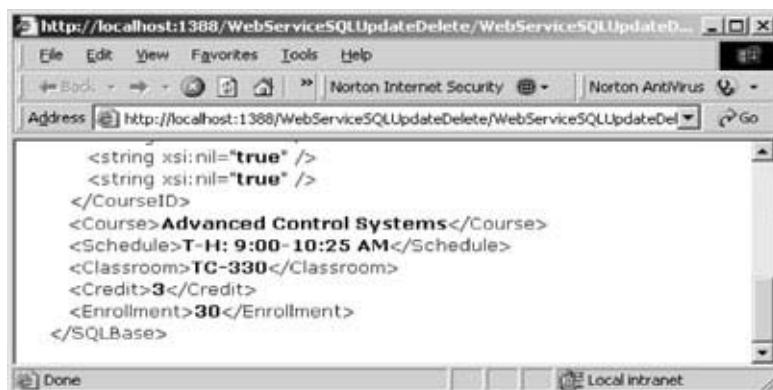


Figure 8.99. The running result of the Web method GetSQLCourseDetail.



Figure 8.100. The parameter-input interface.

and you will see a message box with the message “No matched course found.” This means that the queried course has been deleted from the Course table and cannot be found from that table again.

Click the OK button on the message box to close it, and the running result is displayed, as shown in Figure 8.103. The following returned values are displayed for two member data items:

- SQLOK: false
- SQLError: No matched course found

```

<?xml version="1.0" encoding="utf-8" ?>
- <SQLBase xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xmlns:xsdt="http://www.w3.org/2001/XMLSchema-
  type" xmlns="http://www.cambridge.org/9780521712354/">
  <SQLOK>true</SQLOK>
- <><CourseID>
  <cstring xsi:nil="true" />
  <cstring xsi:nil="true" />
</CourseID>
<Credit>0</Credit>
<Enrollment>0</Enrollment>
</SQLBase>

```

Figure 8.101. The running result of the Web method SQLDeleteSP



The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost:1388/WebServiceSQLUpdateDelete/WebServiceSQLUpdateDelete.asmx>. The page content displays an XML document representing course data:

```

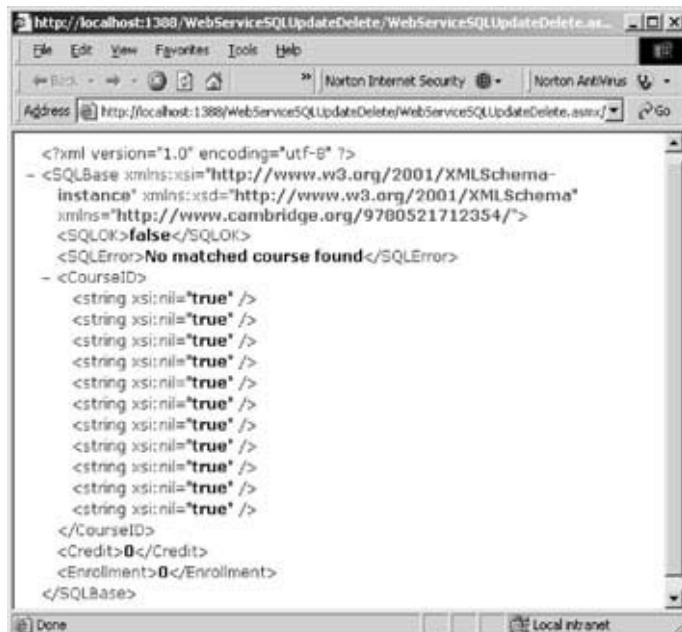
<?xml version="1.0" encoding="utf-8" ?>
-<SQLBase xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://www.cambridge.org/9780521712354/">
<SQLOK>true</SQLOK>
- <CourseID>
  <string>CSC-132B</string>
  <string>CSC-234A</string>
  <string>CSE-434</string>
  <string>CSE-438</string>
  <string>CSE-665</string>
  <string xsi:nil="true" />
  <string xsi:nil="true" />
  <string xsi:nil="true" />
  <string xsi:nil="true" />
</CourseID>
<Credit>0</Credit>
<Enrollment>0</Enrollment>
</SQLBase>

```

Figure 8.102. The running result of the Web method GetSQLCourse.

This is identical to 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.



The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost:1388/WebServiceSQLUpdateDelete/WebServiceSQLUpdateDelete.asmx>. The page content displays an XML document representing course data, including an error message:

```

<?xml version="1.0" encoding="utf-8" ?>
-<SQLBase xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://www.cambridge.org/9780521712354/">
<SQLOK>false</SQLOK>
<SQLError>No matched course found</SQLError>
- <CourseID>
  <string xsi:nil="true" />
  <string xsi:nil="true" />
</CourseID>
<Credit>0</Credit>
<Enrollment>0</Enrollment>
</SQLBase>

```

Figure 8.103. The running result of the Web method GetSQLCourseDetail.

As a reminder, it is highly recommended to recover the deleted data for 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 pieces of information shown in Table 8.6 for the deleted course CSE-526 into our Course table.

You can remove all MsgBox() message box functions from this Web Service project to speed up the execution of this Web Service if you like.

The completed Web Service project WebServiceSQLUpdateDelete can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

Next, let's take care of building some Windows-based and Web-based client projects to consume this Web Service.

## **8.6 BUILD WINDOWS-BASED WEB SERVICE CLIENTS TO CONSUME THE WEB SERVICES**

To save time and space, we do not need to create a new project and perform a full development; instead, we can copy and modify the existing Windows-based client project WinClientSQLInsert we developed in Section 8.4.4 in this chapter to make it into our new client project, WinClientSQLUpdateDelete. To do that, create a new folder, **Chapter 8**, at our root directory if you have not already done so, then go to [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in and copy this client project from the folder **DBProjects\Chapter 8** and paste it into our new folder **C:\Chapter 8**. Rename the copied project WinClientSQLUpdateDelete.

Now let's perform the necessary modifications to this project to make it into our new project. The modifications can be divided into four parts: the modifications to the file folder and the project files, the addition of a new Web reference to our new client project, the modifications to the graphical user interface or form window, and the modifications to the code in the code window. First, let's perform the modifications to the first part.

### **8.6.1 Modifications to the File Folder and Project Files**

Open the Windows Explorer and browse to our new project folder WinClientSQLUpdateDelete that is located in the folder **C:\Chapter 8**, and perform the following modifications:

1. Rename the project folder from WinClientSQLInsert to WinClientSQLUpdateDelete.
2. Rename the project file from WinClientSQLInsert.vbproj to WinClientSQLUpdateDelete.vbproj.
3. Double-click the project file WinClientSQLUpdateDelete.vbproj to open the project. In the project window, go to the Project|WinClientSQLUpdateDelete Properties menu item to open the project property dialog box. Perform the following modifications:
  - a. Change the Assembly name to WinClientSQLUpdateDelete.
  - b. Change the Root namespace to WinClientSQLUpdateDelete.

- c. Click the Assembly Information button to open the associated dialog box, change the Title to WinClientSQLUpdateDelete, and change the Product to WinClientSQLUpdateDelete. Click the OK button to close this dialog.
  - d. Click the Start Debugging button to run the project to make these modifications, and then click the Back button to terminate the project.
4. Reopen the Windows Explorer, browse to our new project folder WinClientSQLUpdateDelete|bin|Debug, and remove all old project files that have the old name WinClientSQLInsert with extensions such as .exe, .pdb, .config, and .xml.
  5. Go to the subfolder WinClientSQLUpdateDelete|obj and remove the old file WinClientSQLInsert.vbproj.FileList.txt.
  6. Go to the subfolder WinClientSQLUpdateDelete|obj|Debug and remove all old resource files with a name of WinClientSQLInsert followed by extensions such as resources and Cache.
  7. Remove the Web Reference folder from the Windows Explorer and Solution Explorer windows. To remove the Web Reference folder from Solution Explorer window, first delete the Web reference object and then delete the folder.

Go to File|Save All to save these modifications.

### 8.6.2 Add a New Web Reference to Our Client Project

To consume or use the Web Service WebServiceSQLUpdateDelete we developed in the last section, we first need to set up a Web reference to point to or connect to that Web Service with our client project. Right-click our new project WinClientSQLUpdateDelete from the Solution Explorer window, and select the item Add Web Reference from the popup menu to open the Add Web Reference dialog box, which is shown in Figure 8.104.

As we mentioned in Section 8.3.10.1, there are two ways to select the desired Web Service and add it as a reference to our client project; one way is to use the browser provided by the Visual Studio.NET 2005 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 dialog box. The second way requires that you first run the Web Service, and then copy its URL and paste it to the URL box in this dialog box if you did not deploy that Web Service to IIS. If you did deploy that Web Service, you can directly type that URL into the URL box in this dialog box.

In this application, we prefer to use the second way to find our Web Service and add it into our project. Open our Web Service project WebServiceSQLUpdateDelete and click the Start Debugging button to run it. Copy the URL from the Address bar, then switch back to our client project WinClientSQLUpdateDelete, and paste that URL into the URL box in the Add Web Reference dialog box. Click the Go button to allow Visual Studio.NET 2005 to search for it.

When our Web Service is found, its name is displayed in the right pane, as shown in Figure 8.104.

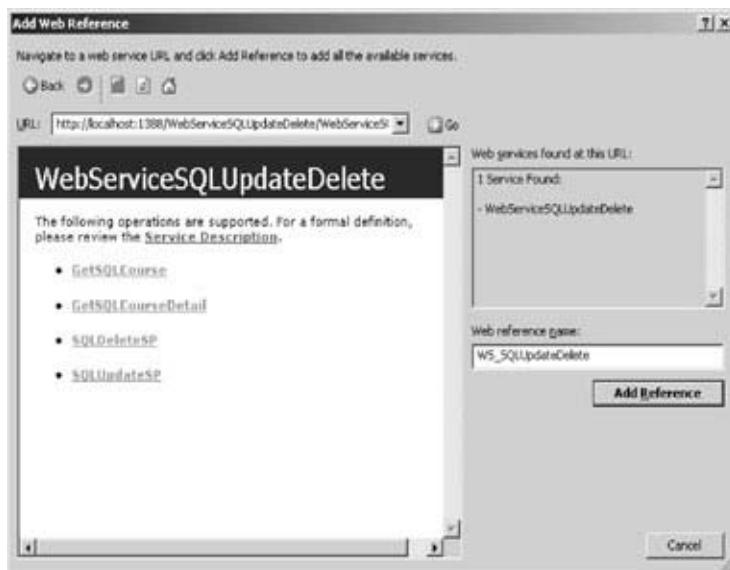


Figure 8.104. The Add Reference dialog box.

Alternately, you can change the name of this Web reference from localhost to any meaningful name, such as WS\_SQLUpdateDelete in our case. Click the Add Reference button to add this Web Service as a reference to our new client project. Click 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\_SQLUpdateDelete, which is under the folder Web References, has been added into the Solution Explorer window in our project. This reference is the so-called Web Service proxy class.

Next, let's modify the graphical user interface by changing some controls to interface with our Web Service to perform the desired data updating and deleting queries.

### 8.6.3 Modifications to the Graphical User Interface

Because we only develop one method to perform data updating and deleting in our Web Service project, we do not need the Method combo box control in this application. We can remove this control from the graphical user interface; however, it does not matter if we keep it without using it in our client project.

Another modification to this form window is to change the name and the title of the Insert button since no data insertion will be performed in this application. Perform the following modifications to this button:

1. Name property: cmdDelete
2. Text property: Delete

Also change the order of the Update and Delete buttons, that is, move the Delete button down and locate it under the Update button. One can use the Ctrl key to select these four buttons and align them either horizontally or vertically with the help of the Format menu item. Our finished graphical user interface is shown in Figure 8.105.

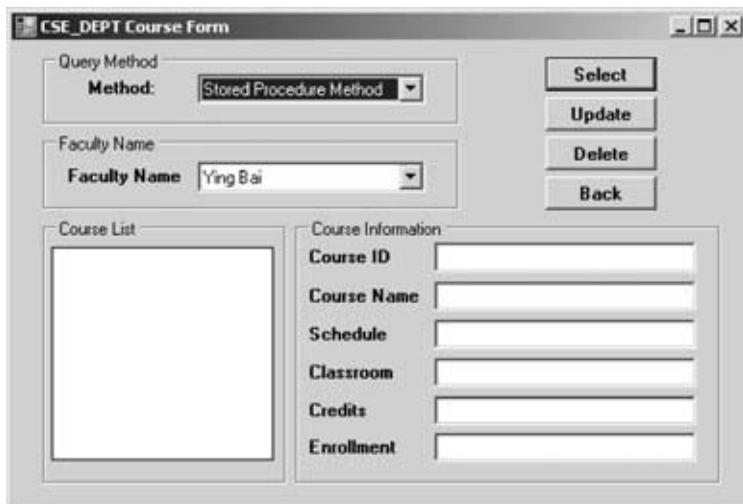


Figure 8.105. The modified graphical user interface.

#### 8.6.4 Modifications to the Code for the Different Event Procedures

The modifications to the code include the following parts:

1. Remove the Insert button's click event procedure and its code.
2. Remove the TextChanged event procedure of the Course ID text box and its code.
3. Remove the subroutine FillCourseDataSet() and its code.
4. Modify the code for the Form\_Load event procedure and form-level variables.
5. Develop the code for the Update button's click event procedure to perform the data updating actions.
6. Develop the code for the Delete button's click event procedure to perform the data deleting actions.
7. Modify the code for the Select button's click event procedure and related subroutines to perform the data validation after the data updating and deleting actions.
8. Modify the code for the SelectedIndexChanged event procedure of the Course List list box control and related subroutine to perform the confirmation for the data updating actions.

Let's start these modifications with step 4 listed above.

##### 8.6.4.1 Modify the Code for the Form\_Load Event Procedure and Form-Level Variables

Perform the following modifications to this part:

1. Remove the two form-level variables dsFlag and wsDataSet since we do not need them in this application.
2. Change the class name of the form-level instance from WS\_SQLInsert.SQLInsertBase to WS\_SQLUpdateDelete.SQLBase.

```

(CourseForm Events) ▾ Load ▾
Imports System.Data
Imports System.Data.SqlClient
Public Class CourseForm
    Private wsSQLResult As New WS_SQLUpdateDelete.SQLBase
    Private Sub CourseForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.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
    End Sub

```

Figure 8.106. The modified Form\_Load event procedure.

3. Remove the second method, DataSet Method, from the Form\_Load event procedure.

Your modified Form\_Load event procedure and form-level variables should match those shown in Figure 8.106. The modifications are highlighted in bold.

Next, let's concentrate on the development of the code for the Update button's click event procedure.

#### **8.6.4.2 Develop the Code for the Update Button Event Procedure**

The functionality of this event procedure is that when a faculty name is selected and all six pieces of updated course information are ready in the six text box controls, the updated course information will be passed to the Web method SQLUpdateSP() we developed in our Web Service project, and the stored procedure WebUpdateCourseSP() is executed to perform this course updating action when the Update button is clicked by the user. Now let's double-click the Update button to open its click event procedure, and enter the code shown in Figure 8.107 into this event procedure.

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

- A. A new instance of our Web proxy class, wsSQLUpdate, is created, and this instance is used to access the Web 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 updated course information to execute the stored procedure WebUpdateCourseSP() to perform this course updating action in our sample database.
- C. An error message will be displayed if an error is encountered during that data updating action.

```

cmdUpdate Click
Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click
    Dim wsSQLUpdate As New WS_SQLUpdateDelete.WebServiceSQLUpdateDelete
    Try
        wsSQLResult = wsSQLUpdate.SQLUpdateSP(ComboName.Text, txtCourseID.Text, _
            txtCourseName.Text, txtSchedule.Text, txtClassRoom.Text, _
            txtCredits.Text, txtEnroll.Text)
    Catch err As Exception
        MsgBox("Web service is wrong: " & err.Message)
    End Try
    If wsSQLResult.SQLOK = False Then
        MsgBox(wsSQLResult.SQLError)
    End If
End Sub

```

Figure 8.107. The code for the Update button click event procedure.

- 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 updating is application-error free. A returned False value of this member data indicates that this data updating encountered an application error, and the error source stored in the member data SQLError is displayed.

It looks like this coding is very simple. Yes, it is! As long as the Web Service is developed and is ready to be used, developing client projects to consume that Web Service is very simple and easy.

In a similar way, we can develop the code for the Delete button's click event procedure to perform data deletion in our sample database.

#### **8.6.4.3 Develop the Code for the Delete Button Event Procedure**

The functionality of this event procedure is that when a course\_id has been selected either from the list box control or from the Course ID text box control in this client form window, the selected course with a primary key that equals that course\_id will be deleted from all tables, including the child and parent tables, in our sample database.

Double-click the Delete button from our client form window to open the Delete button's click event procedure, and enter the code shown in Figure 8.108 into this event procedure.

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

- A. A new instance of our Web proxy class, wsSQLDelete, is created, and this instance is used to access the Web 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, which works as an identifier, to run the stored procedure WebDeleteCourseSP() to perform this course deleting action in our sample database.

	cmdDelete	▼	Click	▼
A	Private Sub cmdDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdDelete.Click			
	Dim wsSQLDelete As New WS_SQLUpdateDelete.WebServiceSQLUpdateDelete			
B	Try			
	wsSQLResult = wsSQLDelete.SQLDeleteSP(txtCourseID.Text)			
C	Catch err As Exception			
	MsgBox("Web service is wrong: " & Err.Message)			
D	End Try			
	If wsSQLResult.SQLOK = False Then			
	MsgBox(wsSQLResult.SQLError)			
	End If			
	End Sub			

**Figure 8.108.** The code for the Delete button's click event procedure.

- C. An error message will be displayed if an 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 value of this member data indicates that this data deleting encountered some application error, and the error source stored in the member data SQLError is displayed.

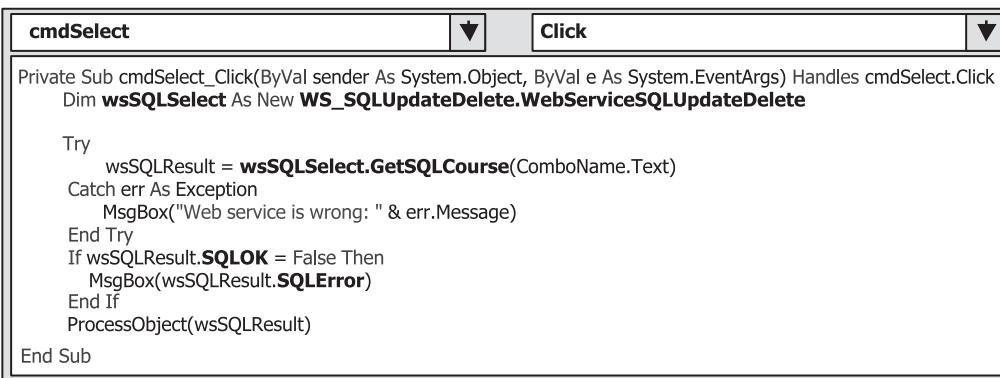
Go to File|Save All to save these modifications and developments.

#### 8.6.4.4 Modify the Code for the Select Button Event Procedure

The functionality of this event procedure is that 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 member from our sample database. To do that, a desired faculty name 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() we developed in our Web Service project, and an instance that contains all retrieved courses taught by the selected faculty member is returned from that Web method. Some subroutines are executed to extract those courses 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 shown in Figure 8.109 to this event procedure:

- 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, the stored procedure method, used in this application. Also remove all code between the Else and End If half-block since we do not have the DataSet method 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.



The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The title bar of the code editor window says "cmdSelect". The tab bar shows "Click" as the active tab. The code itself is as follows:

```

Private Sub cmdSelect_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim wsSQLSelect As New WS_SQLUpdateDelete.WebServiceSQLUpdateDelete
    Try
        wsSQLResult = wsSQLSelect.GetSQLCourse(ComboName.Text)
        Catch err As Exception
            MsgBox("Web service is wrong: " & err.Message)
        End Try
        If wsSQLResult.SQLOK = False Then
            MsgBox(wsSQLResult.SQLError)
        End If
        ProcessObject(wsSQLResult)
    End Sub

```

Annotations A through E are placed to the left of the code to highlight specific changes:

- A**: Points to the line "Handles cmdSelect.Click".
- B**: Points to the line "Dim wsSQLSelect As New WS\_SQLUpdateDelete.WebServiceSQLUpdateDelete".
- C**: Points to the line "Try".
- D**: Points to the line "If wsSQLResult.SQLOK = False Then".
- E**: Points to the line "End If".

Figure 8.109. The modified codes for the Select button event procedure.

- Change the name of the member data from SQLInsertOK to SQLOK.
- Change the name of the member data from SQLInsertError to SQL-Error.

All modifications are highlighted in bold.

Two related subroutines are associated with the Select button's click event procedure, and they are ProcessObject() and FillCourseListBox(). The modifications to these two subroutines include the following steps:

- Change the data type of passed argument wsResult from WS\_SQLInsert. SQLInsertBase to WS\_SQLUpdateDelete.SQLBase.
- Change the If block condition variable from SQLInsertOK to SQLOK.
- Change the error message member data from SQLInsertError to SQL-Error.
- Change the data type of passed argument wsResult from WS\_SQLInsert. SQLInsertBase to WS\_SQLUpdateDelete.SQLBase.

The two modified subroutines are shown in Figure 8.110, and the modifications are highlighted in bold.

#### 8.6.4.5 Modify the Code for the SelectedIndexChanged Event Procedure

Open the SelectedIndexChanged event procedure, and perform the modifications shown in Figure 8.111 to this event procedure.

Let's take a closer look at these modifications.

- Change the new instance's name to wsSQLSelect, and change the Web proxy class's name to WS\_SQLUpdateDelete.WebServiceSQLUpdateDelete.
- Change the instance name of our Web proxy class from wsSQLInsert to wsSQLSelect, and the Web method's name from GetSQLInsert to GetSQLCourseDetail.
- Change the name of the member data from SQLInsertOK to SQLOK.
- Change the name of the member data from SQLInsertError to SQL-Error.

```
CourseForm
```

---

```
Private Sub ProcessObject(ByRef wsResult As WS_SQLUpdateDelete.SQLBase)
    If wsResult.SQLOK = True Then
        Call FillCourseListBox(wsResult)
    Else
        MsgBox("Course information cannot be retrieved: " & wsResult.SQLError)
    End If
End Sub
```

---

```
Private Sub FillCourseListBox(ByRef sqlResult As WS_SQLUpdateDelete.SQLBase)
    Dim index As Integer
    CourseList.Items.Clear()           'clean up the course listbox
    For index = 0 To sqlResult.CourseID.Length - 1
        If sqlResult.CourseID(index) <> vbNullString Then
            CourseList.Items.Add(sqlResult.CourseID(index))
        End If
    Next index
End Sub
```

**Figure 8.110.** The modified subroutines ProcessObject and FillCourseListBox.

The modification to the related subroutine FillCourseDetail() is to change the data type of the passed argument from WS\_SQLInsert.SQLInsertBase to WS\_SQLUpdateDelete.SQLBase.

At this point, we have finished all modifications to this client project, and now it is time to run this project to access our Web Service to perform the data updating and deleting actions. However, before we run this project, make sure that our Web Service project `WebServiceSQLUpdateDelete` is in the open status. This can be identified by a small white icon located in the status bar on the bottom of the screen. If you cannot find this icon, open the Web Service project `WebServiceSQLUpdateDelete` and click the Start Debugging button to run it. As long as our Web Service runs once, it can be closed by clicking the Close button to terminate it, but the small white icon should remain, which means that our Web Service is open and ready to be accessed and consumed.

Now click the Start Debugging button from our client project to run it. First, let's test the data updating functionality by updating a course record, CSE-665. But

```
CourseList  
SelectedIndexChanged  
Private Sub CourseList_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) _  
    Handles CourseList.SelectedIndexChanged  
    Dim wsSQLSelect As New WS_SQLUpdateDelete.WebServiceSQLUpdateDelete  
    Try  
        wsSQLResult = wsSQLSelect.GetSQLCourseDetail(CourseList.Text)  
    Catch err As Exception  
        MsgBox("Web service is wrong: " & err.Message)  
    End Try  
    If wsSQLResult.SQLOK = False Then  
        MsgBox(wsSQLResult.SQLError)  
        Exit Sub  
    End If  
    Call FillCourseDetail(wsSQLResult)  
End Sub
```

**Figure 8.111.** The modified codes for the SelectedIndexChanged event procedure.

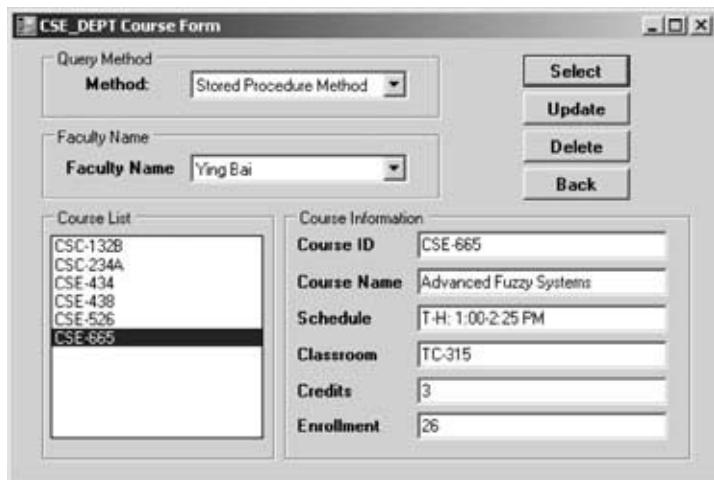


Figure 8.112. The detailed information of the course CSE-665.

before we do that, we will retrieve the current information for the course CSE-665. Click the Select button to get all courses currently taught by the selected faculty member, Ying Bai, and all courses will be retrieved and displayed in the list box control. Click the course CSE-665 from the list box control to get the detailed information for this course. Immediately the detailed information related to course CSE-665 is displayed in the associated text box controls, as shown in Figure 8.112.

Now enter the following updated information for the course CSE-665 into the associated text boxes, as shown in Figure 8.113.

Click the Update button to call the Web method SQLUpdateSP() in our Web Service project to update this course record.

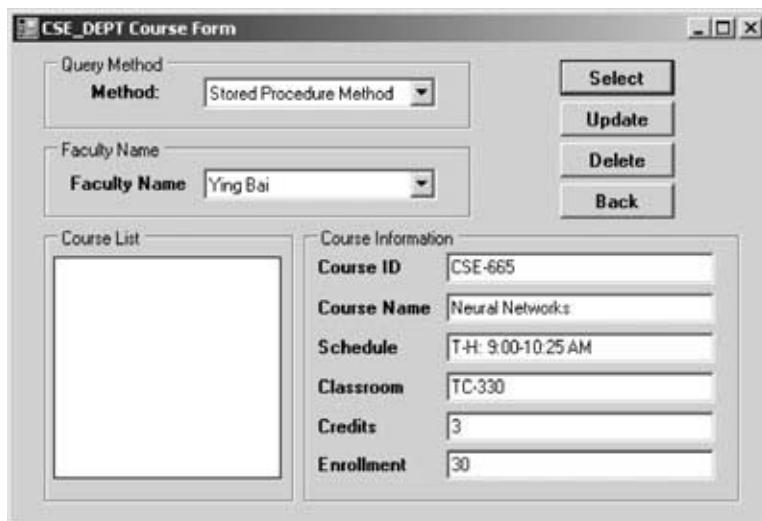
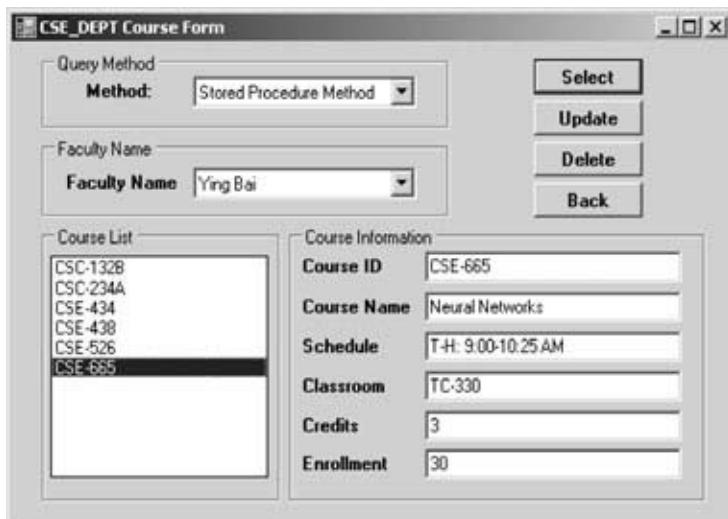


Figure 8.113. The updated information for the course CSE-665.



**Figure 8.114.** The updated course CSE-665.

To confirm this data updating, click the Select button to try to retrieve all courses taught by the selected faculty member, Ying Bai. All courses taught by that faculty member, are returned and displayed in the list box control, which is shown in Figure 8.114.

To check whether the course CSE-665 has been updated or not, first select another course from the list box, such as CSC-132B, and then click the course CSE-665 from the list box control. Immediately the detailed information about this updated course is displayed in the associated text boxes, as shown in Figure 8.114. It can be seen that this course was updated successfully.

To test the deleting functionality, keep the course CSE-665 selected from the list box, and click the Delete button to try to delete this record from the Course table. To confirm this course deleting action, click the Select button to try to retrieve all courses taught by the selected faculty member. Immediately all courses 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 in the list box now.

Click the Back button to terminate our client project. Our client project is very successful.

But 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. Use the Server Explorer window in Visual Studio.NET to open our sample database CSE\_DEPT.mdf and our Course data table.
2. Use Microsoft SQL Server Management Studio or Studio Express to open our sample database CSE\_DEPT.mdf and our Course data table.
3. Use our Web Service project WebServiceSQLInsert to insert a new course to perform this course recovery.
4. Use our Windows-based Web Service client project WinClientSQLInsert to perform this course recovery.

5. Use our Web-based Web Service client project WebClientSQLInsert to insert a new course to recover this course record.

Relatively speaking, the last three methods to recover this course information are more professional since normally no one wants to access and change the content of a database directly by opening the database to do modifications.

The completed Windows-based Web Service client project WinClientSQLUpdateDelete can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

## **8.7 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 will modify the existing Web-based client project WebClientSQLInsert we developed in the previous section to make it into our new Web-based client project WebClientSQLUpdateDelete.

We could copy and rename that entire project as our new Web-based client project, but instead we will create a new ASP.NET Web site project and then copy and modify the Course page.

This project can be developed using the following steps:

1. Create a new ASP.NET Web site project WebClientSQLUpdateDelete, and add the existing Web page Course.aspx from the project WebClientSQLInsert into our new project.
2. Add a Web Service reference to our new project, and modify the Web form window of Course.aspx to meet our data updating and deleting 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 updating and deleting. The code modifications include the following:
  - a. Remove the Insert button's click event procedure cmdInsert\_Click() since we do not need any data insertion in this application.
  - b. Remove the subroutine FillCourseDataSet() since no DataSet method will be used in this application.
  - c. Remove the TextChanged event procedure of the Course ID text box since we do not need this event and its event procedure in this application.
  - d. Modify the code in the Page\_Load event procedure.
  - e. Develop the code for the Update button's click event procedure.
  - f. Develop the code for the Delete button's click event procedure.
  - g. Modify the code in the Select button's click event procedure and the related subroutines such as ProcessObject() and FillCourseListBox().
  - h. Modify the code in the SelectedIndexChanged event procedure of the course list box control and the related subroutine FillCourseDetail().

Now let's start with the first step listed above.

### 8.7.1 Create a New Web Site Project and Add an Existing Web Page

Open Visual Studio.NET 2005 and go to the File|New Web Site menu item to create a new Web site project. Enter “C:\Chapter 8\WebClientSQLUpdateDelete” into the name box that is next to the Location box, and click OK to create this new project.

In the opened new project window, right-click the new project icon WebClientSQLUpdateDelete from the Solution Explorer window, and select Add Existing Item from the popup menu to open the Add Existing Item dialog box. Browse to the Web project WebClientSQLInsert that is located at www.cambridge/9780521712354 in the folder **DBProjects\Chapter 8**, select it, and then click the Open button to open all existing items for this Web site project.

Select both items, Course.aspx and Course.aspx.vb, from the list and click the Add button to add these two items into our new Web site project.

### 8.7.2 Add a Web Service Reference and Modify the Web Form Window

To add a Web reference of our Web Service to this new Web site project, right-click our new project icon from the Solution Explorer window and select the item Add Web Reference from the popup menu. Now open our Web Service project WebServiceSQLUpdateDelete and click the Start Debugging button to run it. As the project runs, copy the URL from the Address box and paste it into the URL box in our Add Web Reference dialog box. Then click the green Go 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 will change it to WS\_SQLUpdateDelete. Your finished Add Web reference dialog box should match the one shown in Figure 8.115.

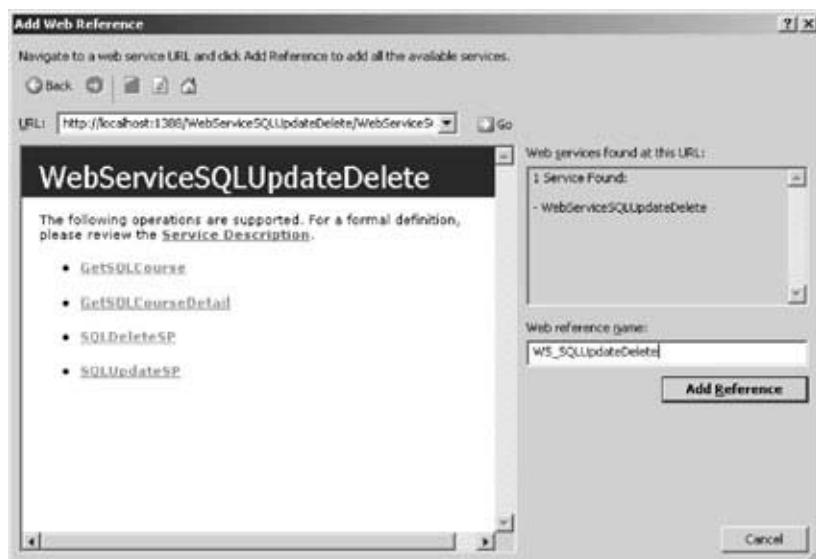


Figure 8.115. The finished Add Web Reference dialog box.

Click the Add Reference button to finish this Web reference adding process. Immediately you will find that the following three files have been created in the Solution Explorer window under the folder App\_WebReferences:

- WebServiceSQLUpdateDelete.disco
- WebServiceSQLUpdateDelete.discomap
- WebServiceSQLUpdateDelete.wsdl

The modifications to the Web page of the Course.aspx include the following steps.

First, because we develop only one method to update and delete data in our Web Service project, we do not need the Method combo box control in this application. We can remove this control from the graphical user interface; however, it does not matter if we keep it without using it in our Web client project.

The second modification to this page window is to change the name and the title of the Insert button since no data insertion will be performed in this application. Perform the following modifications to this button:

1. Name property: cmdDelete
2. Text property: Delete

Also change the order of the Update and the Delete buttons, that is, move the Delete button right and locate it after the Update button. One can use the Ctrl key to select these four buttons and align them either horizontally or vertically with the help of the Format menu item. Our finished graphical user interface is shown in Figure 8.116.

Go to the File|Save All menu item to save these modifications.

Now let's take care of modifications to the code in related event procedures and subroutines in the Course.aspx page.

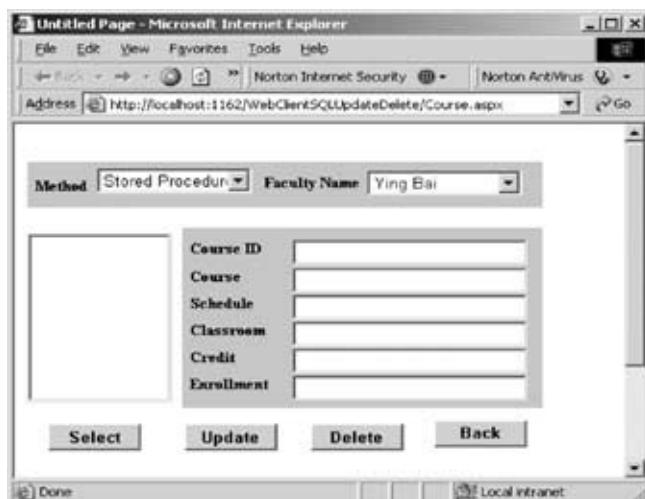


Figure 8.116. The modified graphical user interface.

```
(Page Events) ▾ Load ▾
Imports System.Data
Partial Class Course
Inherits System.Web.UI.Page
Private wsSQLResult As New WS_SQLUpdateDelete.SQLBase
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
If Not IsPostBack Then
    ComboName.Items.Add("Ying Bai")
    ComboName.Items.Add("Satish Bhalla")
    ComboName.Items.Add("Black Anderson")
    ComboName.Items.Add("Steve Johnson")
    ComboName.Items.Add("Jenney King")
    ComboName.Items.Add("Alice Brown")
    ComboName.Items.Add("Debby Angles")
    ComboName.Items.Add("Jeff Henry")
    ComboMethod.Items.Add("Stored Procedure Method")
End If
End Sub
```

Figure 8.117. The modified Page\_Load event procedure.

### 8.7.3 Modify the Code for the Related Event Procedures and Subroutines

The first modification is to change the code in the Page\_Load event procedure and some global variables.

#### 8.7.3.1 Modify the Code for the Page\_Load Event Procedure

Perform the following changes to complete this modification:

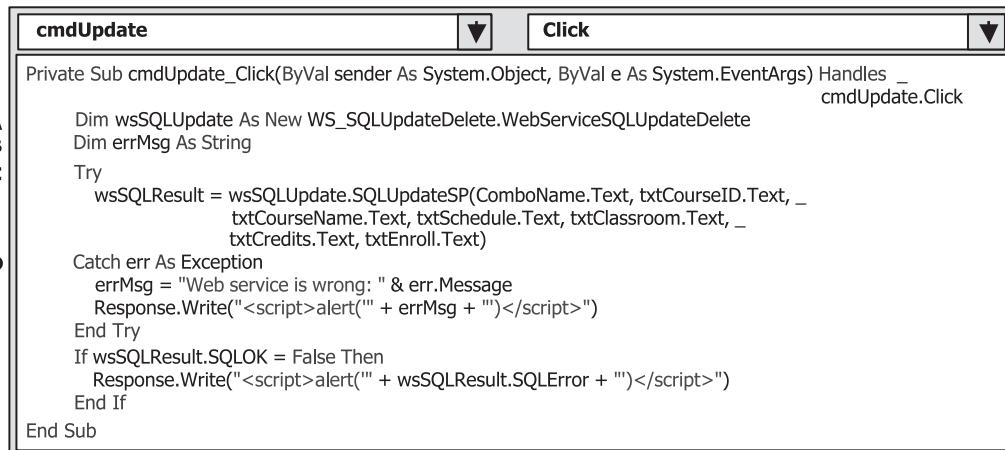
1. Remove the form level variables dsFlag and wsDataSet.
2. Change the name of the base class for the form level instance wsSQLResult from WS.SQLInsert.SQLInsertBase to WS.SQLUpdateDelete.SQLBase.
3. In the Page\_Load event procedure, remove the code that is used to add and display the second Web method, DataSet Method, in the combo box control.

Your modified code for the Page\_Load event procedure should match that shown in Figure 8.117. The modifications are highlighted in bold.

The next step is to develop the code for the Update button's click event procedure.

#### 8.7.3.2 Develop Code for the Update Button Event Procedure

The functionality of this event procedure is that when a faculty name is selected and all six pieces of updated course information are ready in the six text box controls, the updated course information will be passed to the Web method SQLUpdateSP() we developed in our Web Service project, and the stored procedure WebUpdateCourseSP() is executed to perform this course updating action when the Update button is clicked by the user. Now let's double-click the Update button to open its click event procedure, and enter the code shown in Figure 8.118 into this event procedure.



The screenshot shows a Microsoft Visual Studio IDE window. The title bar says "cmdUpdate" and the dropdown menu says "Click". The code is written in VB.NET:

```

Private Sub cmdUpdate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdUpdate.Click
    Dim wsSQLUpdate As New WS_SQLUpdateDelete.WebServiceSQLUpdateDelete
    Dim errMsg As String
    Try
        wsSQLResult = wsSQLUpdate.SQLUpdateSP(ComboName.Text, txtCourseID.Text,
                                              txtCourseName.Text, txtSchedule.Text, txtClassroom.Text,
                                              txtCredits.Text, txtEnroll.Text)
    Catch err As Exception
        errMsg = "Web service is wrong: " & err.Message
        Response.Write("<script>alert('" + errMsg + "')</script>")
    End Try
    If wsSQLResult.SQLOK = False Then
        Response.Write("<script>alert('" + wsSQLResult.SQLError + "')</script>")
    End If
End Sub

```

**Figure 8.118.** The code for the Update button click event procedure.

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

- A. A new instance of our Web proxy class, wsSQLUpdate, is created, and this instance is used to access the Web method SQLUpdateSP() we developed in our Web Service class 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 updated course information to execute the stored procedure WebUpdateCourseSP() to perform this course update in our sample database.
- D. An error message will be displayed if an error is encountered during that data updating action. A point to note is the display format of this error message. To display a string variable in a message box in the client side, one must use the Javascript 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 value of this member data indicates that this data update encountered some application error, and the error source stored in the member data SQLError is displayed using the Java script function alert().

In a similar way, we can develop the code for the Delete button's click event procedure to perform data deletion in our sample database.

#### 8.7.3.3 Develop Code for the Delete Button's Click Event Procedure

The functionality of this event procedure is that when a course\_id has been selected either from the list box control or from the Course ID text box control in this client page window, the selected course with a primary key that equals that course\_id will

```

cmdDelete          Click
Protected Sub cmdDelete_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdDelete.Click
    Dim wsSQLDelete As New WS_SQLUpdateDelete.WebServiceSQLUpdateDelete
    Dim errMsg As String
    Try
        wsSQLResult = wsSQLDelete.SQLDeleteSP(txtCourseID.Text)
    Catch err As Exception
        errMsg = "Web service is wrong: " & err.Message
        Response.Write("<script>alert('" + errMsg + "')</script>")
    End Try
    If wsSQLResult.SQLOK = False Then
        Response.Write("<script>alert('" + wsSQLResult.SQLError + "')</script>")
    End If
End Sub

```

**Figure 8.119.** The code for the Delete button click event procedure.

be deleted from all tables, including the child and parent tables, from our sample database.

Double-click the Delete button from our client page window to open the Delete button's click event procedure, and enter the code shown in Figure 8.119 into this event procedure.

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

- A. A new instance of our Web proxy class, wsSQLDelete, is created, and this instance is used to access the Web method SQLDeleteSP() we developed in our Web Service class WebServiceSQLUpdateDelete.
- 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, which works as an identifier, to run the stored procedure WebDeleteCourseSP() to perform this course deletion in our sample database.
- D. An error message will be displayed if any error is encountered during the data deletion. A point to note is the display format of this error message. To display a string variable in a message box in the client side, one must use the Javascript 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 deletion is application-error free. A returned False value of this member data indicates that this data deletion encountered some application error, and the error source stored in the member data SQLError is displayed.

Go to File|Save All to save these modifications and developments.

```

cmdSelect
    Click
Protected Sub cmdSelect_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles cmdSelect.Click
    Dim wsSQLSelect As New WS_SQLUpdateDelete.WebServiceSQLUpdateDelete
    Dim errMsg As String
    Try
        wsSQLResult = wsSQLSelect.GetSQLCourse(ComboName.Text)
    Catch err As Exception
        errMsg = "Web service is wrong: " & err.Message
        Response.Write("<script>alert('" + errMsg + "')</script>")
    End Try
    If wsSQLResult.SQLOK = False Then
        Response.Write("<script>alert('" + wsSQLResult.SQLError + "')</script>")
    End If
    ProcessObject(wsSQLResult)
End Sub

```

Figure 8.120. The modified code for the Select button event procedure.

#### 8.7.3.4 Modify Code for the Select Button Event Procedure and Related Subroutines

The functionality of this event procedure is that 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 member from our sample database. To do that, the desired faculty name 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() we developed in our Web Service project, and an instance that contains all retrieved courses taught by the selected faculty member is returned from that Web method. Some subroutines are executed to extract those courses 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 shown in Figure 8.120 to this event procedure:

- A. Change 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, which 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, the stored procedure method, used in this application. Also remove all code between the Else and End If half-block since we do not have the DataSet method 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 other 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.

The modifications are highlighted in bold.

	Course	▼	ProcessObject	▼
<b>A</b>	<pre>Private Sub ProcessObject(ByRef wsResult As WS_SQLUpdateDelete.SQLBase)     Dim errMsg As String     If wsResult.SQLOK = True Then         Call FillCourseListBox(wsResult)     Else         errMsg = "Course information cannot be retrieved: " &amp; wsResult.SQLError         Response.Write("&lt;script&gt;alert('" + errMsg + "')&lt;/script&gt;")     End If End Sub</pre>			
<b>B</b>				
<b>C</b>				
<b>D</b>	<pre>Private Sub FillCourseListBox(ByRef sqlResult As WS_SQLUpdateDelete.SQLBase)     Dim index As Integer     CourseList.Items.Clear()                                'clean up the course listbox     For index = 0 To sqlResult.CourseID.Length - 1         If sqlResult.CourseID(index) &lt;&gt; vbNullString Then             CourseList.Items.Add(sqlResult.CourseID(index))         End If     Next index End Sub</pre>			

Figure 8.121. The modified subroutines ProcessObject and FillCourseListBox.

Two related subroutines are associated with the Select button's click event procedure, and they are ProcessObject() and FillCourseListBox(). The modifications to these two subroutines include the following steps:

- Change the data type of passed argument wsResult from WS\_SQLInsert.SQLInsertBase to WS\_SQLUpdateDelete.SQLBase.
- Change the If block condition variable from SQLInsertOK to SQLOK.
- Change the error message member data from SQLInsertError to SQLError.
- Change the data type of passed argument wsResult from WS\_SQLInsert.SQLInsertBase to WS\_SQLUpdateDelete.SQLBase.

The two modified subroutines are shown in Figure 8.121, and all modifications are highlighted in bold.

Go to File|Save All to save these modifications. Next, we will perform the modifications to the last event procedure, SelectedIndexChanged, which is the event procedure of the course list box control in our page window.

### 8.7.3.5 Modify Code for the SelectedIndexChanged Event Procedure of the Course List Box Control and Related Subroutines

Open the SelectedIndexChanged event procedure, and perform the modifications shown in Figure 8.122 to this event procedure.

Let's take a closer look at these modifications.

- Change the new instance's name to wsSQLSelect, and change the Web proxy class's name to WS\_SQLUpdateDelete.WebServiceSQLUpdateDelete.
- Change the instance name of our Web proxy class from wsSQLInsert to wsSQLSelect, and the Web method's name from GetSQLInsert to GetSQLCourseDetail.

CourseList	SelectedIndexChanged
<pre> Protected Sub CourseList_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles _     CourseList.SelectedIndexChanged     Dim wsSQLSelect As New WS_SQLUpdateDelete.WebServiceSQLUpdateDelete     Dim errMsg As String     Try         wsSQLResult = wsSQLSelect.GetSQLCourseDetail(CourseList.Text)     Catch err As Exception         errMsg = "Web service is wrong: " &amp; err.Message         Response.Write("&lt;script&gt;alert('" + errMsg + "')&lt;/script&gt;")     End Try     If wsSQLResult.SQLOK = False Then         Response.Write("&lt;script&gt;alert('" + wsSQLResult.SQLError + "')&lt;/script&gt;")         Exit Sub     End If     Call FillCourseDetail(wsSQLResult) End Sub </pre>	

**Figure 8.122.** The modified code for the SelectedIndexChanged event procedure.

- C. Change the name of the member data from SQLInsertOK to SQLOK.
- D. Change the name of the member data from SQLInsertError to SQL-Error.

The modification to the related subroutine FillCourseDetail() is to change the data type of the passed argument from WS\_SQLInsert.SQLInsertBase to WS\_SQLUpdateDelete.SQLBase.

At this point, we have finished all modifications to this Web-based client project, and now it is time to run this project to access our Web Service to perform the data updating and deleting actions. However, before we run this project, make sure that our Web Service project WebServiceSQLUpdateDelete is in the open status. This can be identified by a small white icon located in the status bar at the bottom of the screen. If you cannot find this icon, open the Web Service project WebServiceSQLUpdateDelete and click the Start Debugging button to run it. As long as our Web Service runs once, it can be closed by clicking the Close button to terminate it, but the small white icon should remain, which means that our Web Service is open and ready to be accessed and consumed.

Now click the Start Debugging button from our client project to run it. First, let's test the data updating functionality by updating the course record, CSE-665. But before we do that, we will retrieve the current information for the course CSE-665. Click the Select button to get all courses currently taught by the selected faculty member, Ying Bai, and all courses will be retrieved and displayed in the list box control. Click the course CSE-665 from the list box control to get the detailed information for this course. Immediately the detailed information related to course CSE-665 is displayed in the associated text box controls, as shown in Figure 8.123.

Now enter the updated information for the course CSE-665 into the associated text boxes, as shown in Figure 8.124.

Now click the Update button to call the Web method SQLUpdateSP() in our Web Service project to update this course record.

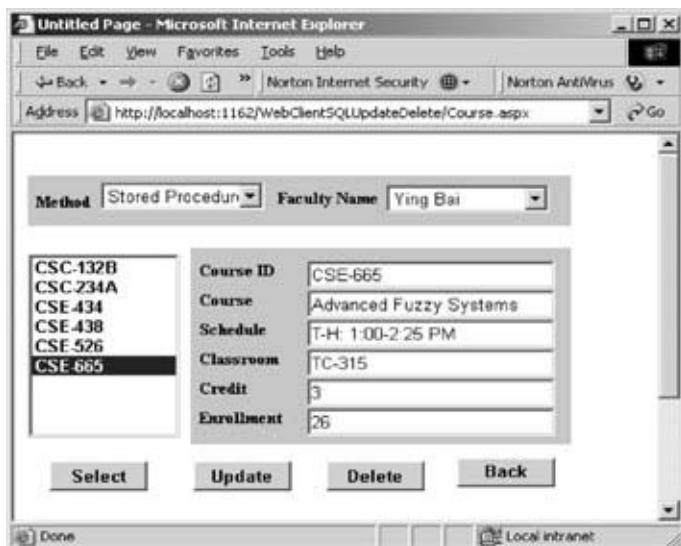


Figure 8.123. The detailed information for the course CSE-665.

To check whether the course CSE-665 has been updated or not, first select another course from the list box, such as CSC-234A, and then click the course CSE-665 from the list box control. Immediately the detailed information about this updated course is displayed in the associated text boxes, as shown in Figure 8.125. It can be found that this course has been updated successfully.

To test the deleting functionality, keep the course CSE-665 selected from the list box, and click the Delete button to try to delete this record from the Course table. To confirm this course deleting action, click the Select button to try to retrieve all



Figure 8.124. The updated information for the course CSE-665.

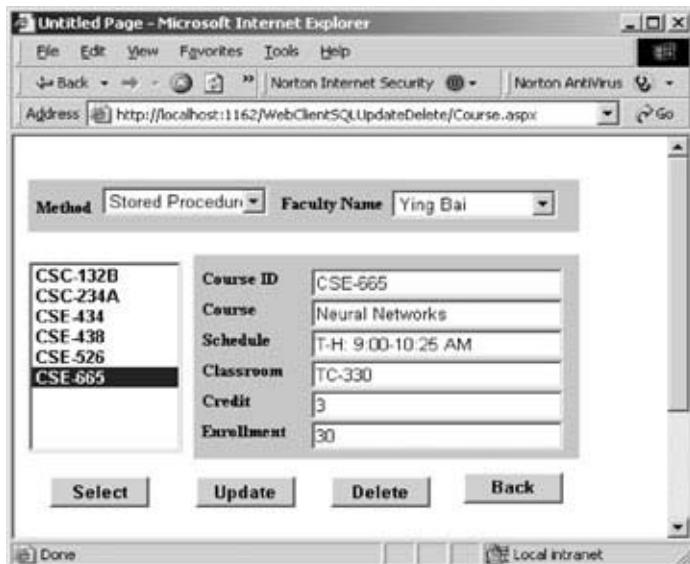


Figure 8.125. The updated course CSE-665.

courses taught by the selected faculty member. Immediately all courses 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 in the list box now.

Click the Back button to terminate our client project. Our client project is very successful.

But 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 data by using one of the following five methods:

1. Use the Server Explorer window in Visual Studio.NET to open our sample database CSE\_DEPT.mdf and our Course data table.
2. Use Microsoft SQL Server Management Studio or Studio Express to open our sample database CSE\_DEPT.mdf and our Course data table.
3. Use our Web Service project WebServiceSQLInsert to insert a new course to perform this course recovery.
4. Use our Windows-based Web Service client project WinClientSQLInsert to perform this course recovery.
5. Use our Web-based Web Service client project WebClientSQLInsert to insert a new course to recover this course record.

Relatively speaking, the last three methods to recover this course information are more professional since normally no one wants to access and change the content of a database directly by opening the database to do modifications.

The completed Web-based Web Service client project WebClientSQLUpdateDelete can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder DBProjects\Chapter 8.

At this point, we have finished the discussion about how to access and manipulate data in the SQL Server database via ASP.NET Web Services. In the next section, we will discuss how to access and manipulate data in the Oracle database via ASP.NET Web Services.

## **8.8 BUILD AN ASP.NET WEB SERVICE PROJECT TO ACCESS AN ORACLE DATABASE**

Basically, the procedure to build an ASP.NET Web Service to access an SQL Server database is very similar to the procedure to build an ASP.NET Web Service to access an Oracle database. The main differences are in the following:

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 points distinguish the procedures to build a Web Service to access these two kinds of databases. Let's have a more detailed discussion of these issues.

First, when connecting to a different database, the connection string is obviously different, which includes the protocol and security issues in that connection string. Refer to Section 4.18.1 in Chapter 4 to get a clear picture of the differences in the connection strings between these two kinds of databases.

Second, as we know, ADO.NET provides different Data Providers to allow users to access the different databases, and these Data Providers are database-dependent, which means a different Data Provider is needed to use to access a different database. For an 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 in an SQL Server database is totally different from calling a stored procedure to perform a similar data action in 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 will develop our Web Service projects to access the Oracle database by modifying some existing Web Service projects in the following sections. In this section, we concentrate on the modifications to the Web Service project WebServiceSQLSelect to make it into our new Web Service project, WebServiceOracleSelect.

### 8.8.1 Build a Web Service Project – WebServiceOracleSelect

In this section, we modify the existing Web Service project WebServiceSQLSelect to make it into our new Web Service project WebServiceOracleSelect and allow it to access the Oracle database to perform data selection queries.

Open the Windows Explorer and create a new folder, **Chapter 8**, under the root directory if you have not done that already. Open Internet Explorer and browse to our desired source Web Service project, WebServiceSQLSelect which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**, and copy and paste it into our new folder **Chapter 8**. Rename it WebServiceOracleSelect. Perform the following modifications to this project:

1. Change the main Web Service page from WebServiceSQLSelect.asmx to WebServiceOracleSelect.asmx.
2. Open the App\_Code folder and change the name of our base class file from SQLSelectBase.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 2005 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.asmx. Open this page by double-clicking 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 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

### 8.8.2 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 = SYSTEM;Password = reback;" />
```

---

The Oracle database server XE is used for the server name, the User ID is the default value SYSTEM, and the password is determined by the user when installing the Oracle Database 10g Express Edition in the local computer.

### 8.8.3 Modify the Namespace Directories

First, we need to add an Oracle Data Provider reference to our Web Service project. To do that, right-click our new project icon WebServiceOracleSelect from the Solution Explorer window, and then select the item Add Reference from the popup menu to open the Add Reference dialog box. Browse down the list until you find the item System.Data.OracleClient, click to select it, and then click the OK button to add it into our project.

Now double-click our code-behind page WebServiceOracleSelect.vb to open it. On the page, change the last namespace line from

---

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

---

to

---

```
Imports System.Data.OracleClient
```

---

Also change the name of our Web Service class, which is located after the accesssing mode Public Class, from WebServiceSQLSelect to WebServiceOracleSelect.

Next, we will perform the necessary modifications to three Web methods combined with those five differences listed above.

### 8.8.4 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.

- A. Rename this Web method GetOracleSelect, and change the name of returned class to OracleSelectResult.
- B. Modify the query string by replacing the LIKE @ symbol before the dynamic parameter facultyName with the symbol =:, which is an assignment operator used in the Oracle database.
- 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 OracleType.VarChar.
- H. Change the name of the returned instance from SQLResult to OracleResult, and change the prefix from sql to ora for all data objects.

Your modified Web method GetOracleSelect() should match the one shown in Figure 8.126. The modifications are highlighted in bold.

Now let's perform the modifications to three related subroutines. Perform the following modifications to the subroutine SQLConn():

- A. Change the name of this subroutine from SQLConn to OracleConn, and the return class name from SqlConnection to OracleConnection. Also change the connection string from sql\_conn to ora\_conn.
- B. Change the data type of the returned connection object to OracleConnection.

Perform the following modifications to the subroutine FillFacultyReader():

- C. Change the data type of the first passed argument from SQLSelectResult to OracleSelectResult. Also change the data type of the second passed argument from SqlDataReader to OracleDataReader.

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() should match those shown in Figure 8.127. All modifications are highlighted in bold.

```

WebServiceOracleSelect           GetOracleSelect
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Data
Imports System.Data.OracleClient
<WebService(Namespace:="http://www.cambridge.org/9780521712354")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class WebServiceOracleSelect
    Inherits System.Web.Services.WebService
    <WebMethod()> _
    Public Function GetOracleSelect(ByVal FacultyName As String) As OracleSelectResult
        Dim cmdString As String = "SELECT faculty_id, office, phone, college, title, email FROM Faculty " + _
            "WHERE name =: facultyName"
        Dim oraConnection As New OracleConnection
        Dim OracleResult As New OracleSelectResult()
        Dim oraCommand As New OracleCommand
        Dim oraReader As OracleDataReader
        OracleResult.OracleRequestOK = True
        oraConnection = OracleConn()
        If oraConnection Is Nothing Then
            OracleResult.OracleRequestError = "Database connection is failed"
            ReportError(OracleResult)
            Return Nothing
        End If
        oraCommand.Connection = oraConnection
        oraCommand.CommandType = CommandType.Text
        oraCommand.CommandText = cmdString
        oraCommand.Parameters.Add("facultyName", OracleType.VarChar).Value = FacultyName
        oraReader = oraCommand.ExecuteReader
        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 8.126. The modified Web method GetOracleSelect.

### 8.8.5 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 in our sample database. The modifications to this Web method include the following two parts:

1. Modifications to the stored procedure since the prototype of a stored procedure in the SQL Server is different from that of a stored procedure in the Oracle database.
2. Modifications to the code in this Web method.

Now let's perform the modifications to the stored procedure first.

```

WebServiceOracleSelect OracleConn
Protected Function OracleConn() As OracleConnection
    Dim cmdString As String = ConfigurationManager.ConnectionStrings("ora_conn").ConnectionString
    Dim conn As New OracleConnection
    conn.ConnectionString = cmdString
    conn.Open()
    If conn.State <> ConnectionState.Open Then
        MsgBox("Database Open is failed")
        conn = Nothing
    End If
    Return conn
End Function

Protected Sub FillFacultyReader(ByRef sResult As OracleSelectResult, ByVal sReader As OracleDataReader)
    If sReader.Read() = True Then
        With sResult
            .FacultyID = Convert.ToString(sReader("faculty_id"))
            .FacultyOffice = Convert.ToString(sReader("office"))
            .FacultyPhone = Convert.ToString(sReader("phone"))
            .FacultyCollege = Convert.ToString(sReader("college"))
            .FacultyTitle = Convert.ToString(sReader("title"))
            .FacultyEmail = Convert.ToString(sReader("email"))
        End With
    End If
End Sub

Protected Sub ReportError(ByVal ErrSource As OracleSelectResult)
    ErrSource.OracleRequestOK = False
    MsgBox(ErrSource.OracleRequestError)
End Sub

```

Figure 8.127. Modified subroutines OracleConn, FillFacultyReader, and ReportError.

### 8.8.5.1 Modifications to the Stored Procedure WebSelectFacultySP

Basically, the modifications to this stored procedure are to develop a similar procedure in the Oracle database environment. To develop a stored procedure that returns data in an Oracle database, we have to build a package since a stored procedure developed in an Oracle database will not return any data. Refer to Section 4.19.7 in Chapter 4 to get more detailed information for building and developing a package in an Oracle database.

Many different methods can be used to build a package in an Oracle database. In this section we want to use the Object Browser page in Oracle Database 10g Express Edition (XE) to build a package.

Open the Oracle Database 10g XE home page by going to Start>All Programs|Oracle Database 10g Express Edition|Go To Database Home Page. Enter the correct user ID and password to complete the login process. Then click the Object Browser and select Create|Package to open the Create Package window, which is shown in Figure 8.128.

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 clicking the Next button to open the Name page, which is shown in Figure 8.129.

Enter the package name WebSelectFaculty into the name box, and click the Next button to go to the specification page, which is shown in Figure 8.130.

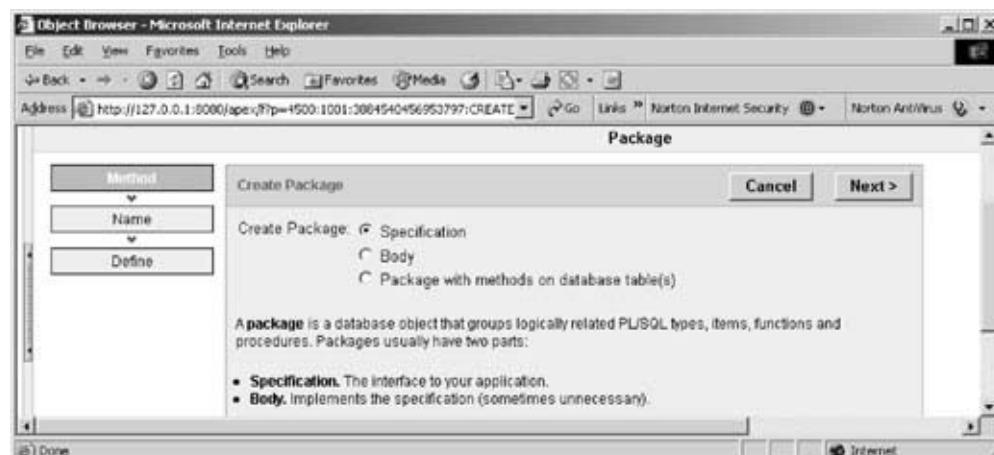


Figure 8.128. The Create Package window.

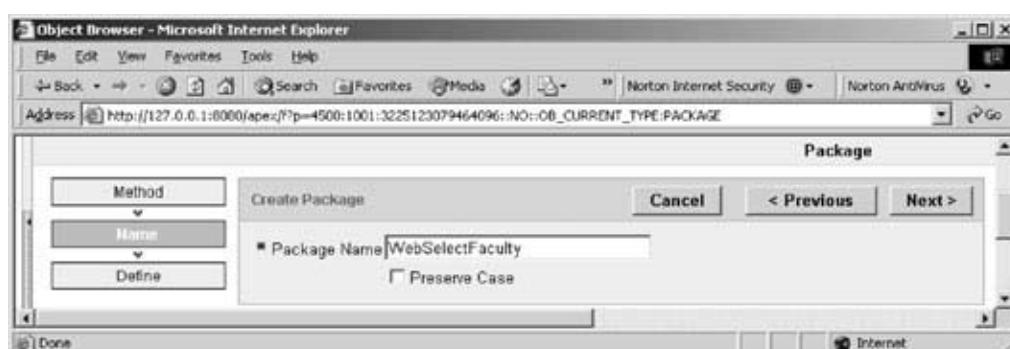


Figure 8.129. The Name page of the Package window.

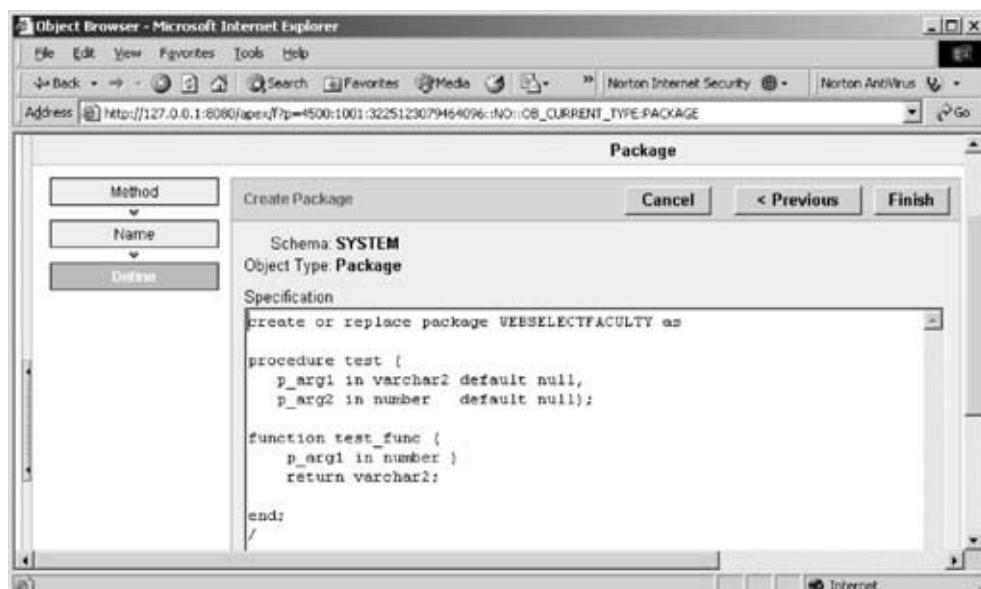


Figure 8.130. The Specification page.

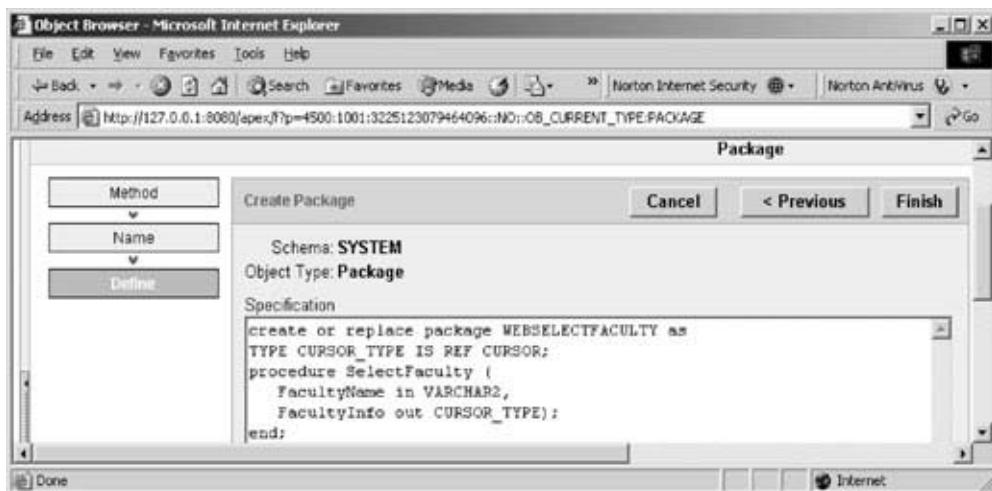


Figure 8.131. The code for the Specification page.

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 test to our procedure name – SelectFaculty. Your code for the specification page should match that shown in Figure 8.131.

The coding language we used in this section is called Procedural Language Extension for SQL or PL/SQL, which is a popular language that is widely used in Oracle database programming.

In line 2, we defined the returned data type as a CURSOR\_TYPE by using

#### **TYPE CURSOR\_TYPE IS REF CURSOR;**

since 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 the keyword **out**. Each PL/SQL statement must end in a semicolon, and this rule also applies to the **end** statement.

You can click the Compile button to compile this specification block if you like. Next we need to create the body block of this package. Click the Finish button to complete this step.

Click the Body tab to open the Body page, which is shown in Figure 8.132.

Click the Edit button to begin to create the body part. Enter the following PL/SQL code into this body, as shown in Figure 8.133.

The procedure prototype is redeclared in line 2. Starting from the **begin**, our real SQL statements are included in lines 6 and 7. The OPEN FacultyInfo FOR

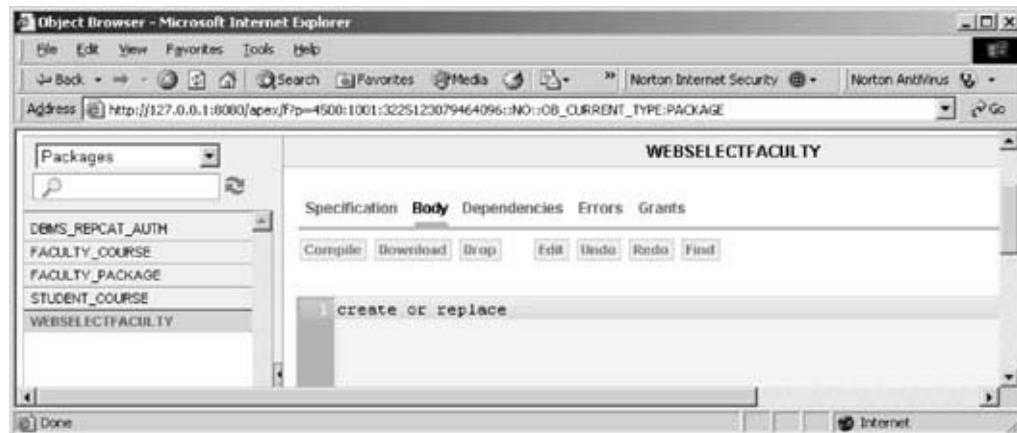


Figure 8.132. The Body page of the package.

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 4.18.9 in Chapter 4. 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 ends the package.

Now let's compile our package by clicking the Compile button. A successful compiling message

**PL/SQL code successfully compiled (10:56:11)**

is displayed if this package is bug-free, which is shown in Figure 8.134.

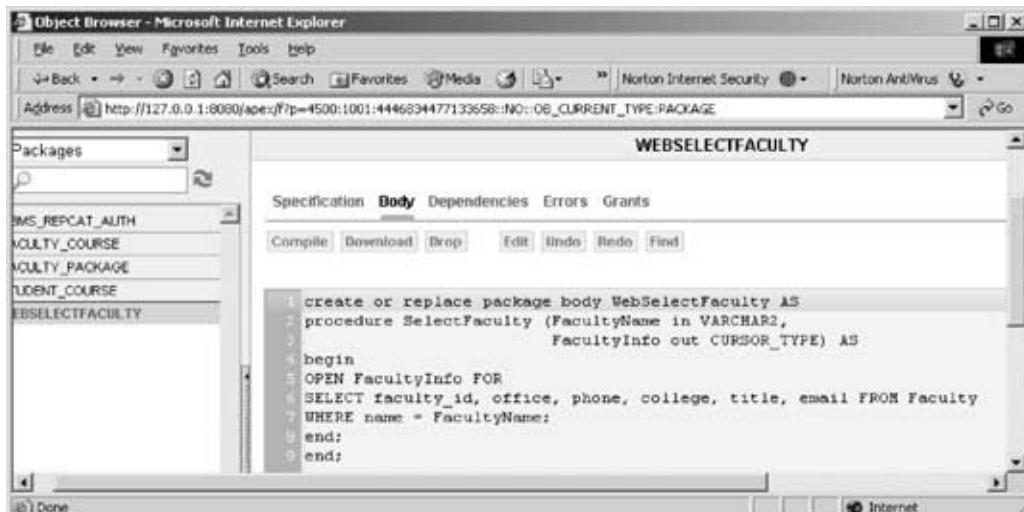


Figure 8.133. The code for the Body part of the package.

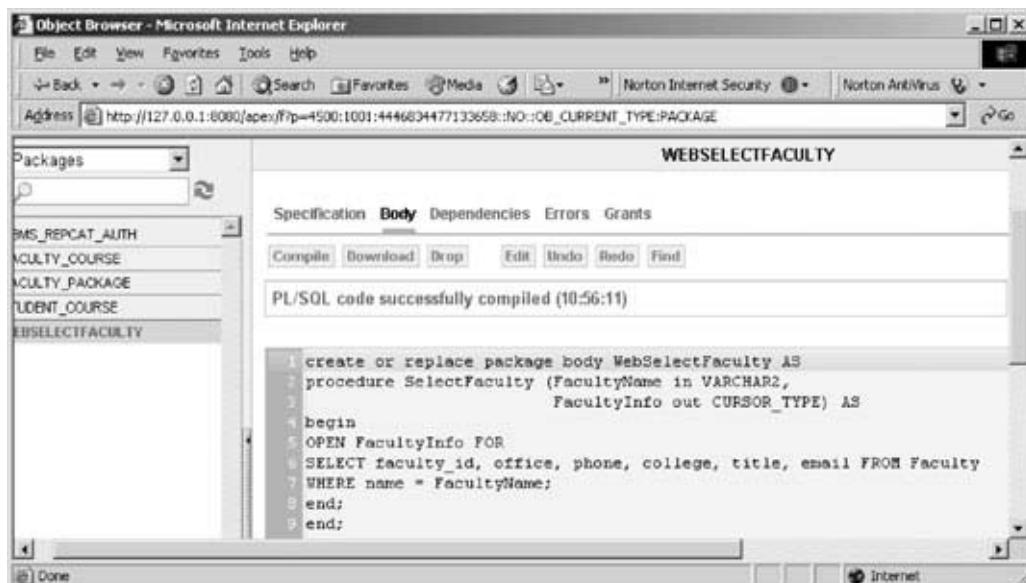


Figure 8.134. The compiled code for the body part of the package.

The development of our Oracle package is complete. Now let's go to Visual Studio.NET to call this package to perform our faculty data query for our Web Service project.

#### **8.8.5.2 Modifications to the Code for 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 these modifications step by step according to this sequence.

- A. Rename this Web method to GetOracleSelectSP, and change the name of the returned class to OracleSelectResult.
- B. Modify the query string by replacing the content of this string with the name of the package we developed in the Oracle database in the last section. The point is that both the package's name (WebSelectFaculty) and the stored procedure's name (SelectFaculty) must be used together with the dot operator to tell the Web Service that an Oracle stored procedure embedded in an Oracle package will be called to perform this faculty data query as the project runs.
- 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. Two Oracle Parameter objects are created here and 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 for this parameter are needed.
  - E. Change the name of the returned instance from SQLResult to OracleResult, and change the 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 the second parameter, as we mentioned in step **D**, is an output parameter with a data type of Cursor. This makes our initialization more complicated. Two points must be noted for this initialization: First, the data type of this parameter must be assigned to OracleType.Cursor, which is identical with the data type we defined in our stored procedure SelectFaculty() in Section 8.8.5.1. Second, the direction of this parameter must be assigned to the **Output** that is a value of the ParameterDirection property.
  - H. This assignment for the parameter direction property is very important; the stored procedure would not be executed correctly if this property were not set up correctly.
  - I. Change the prefix from sql to ora for all data objects.
  - J. The Add() method is executed to add two initialized parameter objects to the Command object and make the latter ready to be called.
  - K. Change the name of the returned instance from SQLResult to OracleResult, and change the prefix from sql to ora for all data objects.

Your modified Web method GetOracleSelectSP() should match the one shown in Figure 8.135, and all modifications are highlighted in bold.

### **8.8.6 Modify the Web Method GetSQLSelectDataSet**

The functionality 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()

```

A <WebMethod()> _
B Public Function GetOracleSelectSP(ByVal FacultyName As String) As OracleSelectResult
C     Dim cmdString As String = "WebSelectFaculty.SelectFaculty"
D     Dim oraConnection As New OracleConnection
E     Dim OracleResult As New OracleSelectResult()
F     Dim oraCommand As New OracleCommand
G     Dim oraReader As OracleDataReader
H     Dim paramFacultyName As New OracleParameter
I     Dim paramFacultyInfo As New OracleParameter
J     OracleResult.OracleRequestOK = True
K     oraConnection = OracleConn()
L     If oraConnection Is Nothing Then
M         OracleResult.OracleRequestError = "Database connection is failed"
N         ReportError(OracleResult)
O         Return Nothing
P     End If
Q     paramFacultyName.ParameterName = "FacultyName"
R     paramFacultyName.OracleType = OracleType.VarChar
S     paramFacultyName.Value = FacultyName
T     paramFacultyInfo.ParameterName = "FacultyInfo"
U     paramFacultyInfo.OracleType = OracleType.Cursor
V     paramFacultyInfo.Direction = ParameterDirection.Output      'this is very important
W     oraCommand.Connection = oraConnection
X     oraCommand.CommandType = CommandType.StoredProcedure
Y     oraCommand.CommandText = cmdString
Z     oraCommand.Parameters.Add(paramFacultyName)
A     oraCommand.Parameters.Add(paramFacultyInfo)
B     oraReader = oraCommand.ExecuteReader
C     If oraReader.HasRows = True Then
D         Call FillFacultyReader(OracleResult, oraReader)
E     Else
F         OracleResult.OracleRequestError = "No matched faculty found"
G         ReportError(OracleResult)
H     End If
I     If Not oraReader Is Nothing Then oraReader.Close()
J     oraReader = Nothing
K     If Not oraConnection Is Nothing Then oraConnection.Close()
L     oraConnection = Nothing
M     Return OracleResult
N End Function

```

Figure 8.135. The modified Web method – GetOracleSelectSP.

should match the one shown in Figure 8.136. The modifications are highlighted in bold.

Let's take a closer look at this modified Web method to see how it works.

- Rename this Web method to GetOracleSelectDataSet.
- Modify the query string by replacing the LIKE @ symbol before the dynamic parameter facultyName with the symbol =:, which is an assignment operator used in the Oracle database.
- 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.
- Change the name of the returned instance from SQLResult to OracleResult, and change the member data from SQLRequestOK to OracleRequestOK.

```

WebServiceOracleSelect      GetOracleSelectDataSet

```

```

<WebMethod()>_
Public Function GetOracleSelectDataSet(ByVal FacultyName As String) As DataSet
    Dim cmdString As String = "SELECT faculty_id, office, phone, college, title, email FROM Faculty " + _
        "WHERE name =: facultyName"
    Dim oraConnection As New OracleConnection
    Dim OracleResult As New OracleSelectResult()
    Dim oraCommand As New OracleCommand
    Dim FacultyAdapter As New OracleDataAdapter
    Dim dsFaculty As New DataSet
    Dim intResult As Integer
    OracleResult.OracleRequestOK = True
    oraConnection = OracleConn()
    If oraConnection Is Nothing Then
        OracleResult.OracleRequestError = "Database connection is failed"
        ReportError(OracleResult)
        Return Nothing
    End If
    oraCommand.Connection = oraConnection
    oraCommand.CommandType = CommandType.Text
    oraCommand.CommandText = cmdString
    oraCommand.Parameters.Add("facultyName", OracleType.VarChar).Value = FacultyName
    FacultyAdapter.SelectCommand = oraCommand
    intResult = FacultyAdapter.Fill(dsFaculty, "Faculty")
    If intResult = 0 Then
        OracleResult.OracleRequestError = "No matched faculty found"
        ReportError(OracleResult)
    End If
    If FacultyAdapter IsNot Nothing Then FacultyAdapter.Dispose()
    FacultyAdapter = Nothing
    If oraConnection IsNot Nothing Then oraConnection.Close()
    oraConnection = Nothing
    Return dsFaculty
End Function

```

**Figure 8.136.** The modified Web method – GetOracleSelectDataSet.

- 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 OracleType.VarChar.
- H. Change the name of the returned instance from SQLResult to OracleResult, and change the prefix from sql to ora for all data objects.

At this point, we have finished all modifications to our new Web Service project. It is time for us to run our project to test the data query functionalities. Click the Start Debugging button to run our Web Service project. Click Yes to the message box to allow the project to run in the Debug mode. The running status of the Web Service project is shown in Figure 8.137.

First, let's test the functionality of the Web method GetOracleSelect() to pick up the detailed information for the selected faculty member. Click this method to open the parameter input page that is shown in Figure 8.138, and enter the selected faculty name Ying Bai into the Value box. Then click the Invoke button to execute this method.

The running result of this Web method is shown in Figure 8.139.



Figure 8.137. The running status of the Web Service project.

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 8.139.

Now let's test the next Web method, GetOracleSelectSP(). To do that, close the running result page by clicking the Close button located at the upper right corner of this page, and click the Back button to return the initial page. Then click the Web method GetOracleSelectSP to open its parameter-input page. Enter the selected faculty name, such as Satish Bhalla, into the Value box, and click the Invoke button to execute this Web method.

This Web method will call the Oracle package WebSelectFaculty that contains the stored procedure SelectFaculty() we developed in the previous section to access

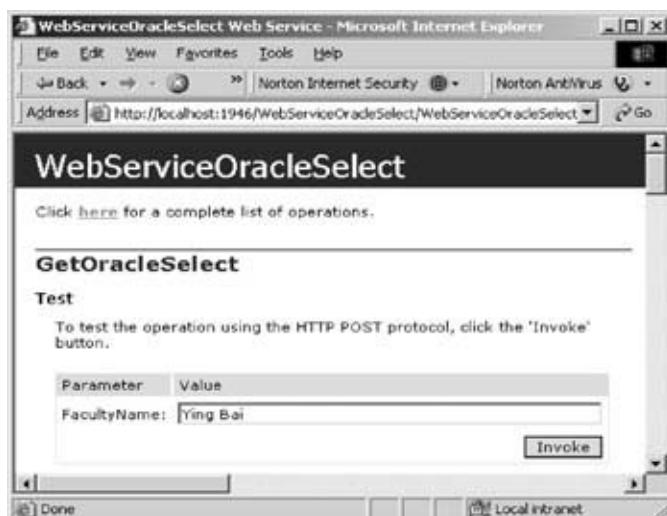


Figure 8.138. The parameter-input page.



The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost:1946/WebServiceOracleSelect/WebServiceOracleSel...>. The page content displays an XML document representing the selected faculty member's information:

```

<?xml version="1.0" encoding="utf-8" ?>
- <OracleSelectResult
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.cambridge.org/9780521712354/">
  <OracleRequestOK>true</OracleRequestOK>
  <FacultyID>B78880</FacultyID>
  <FacultyOffice>MTC-211</FacultyOffice>
  <FacultyPhone>750-378-1148</FacultyPhone>
  <FacultyCollege>Florida Atlantic
    University</FacultyCollege>
  <FacultyTitle>Assistant Professor</FacultyTitle>
  <FacultyEmail>ybai@college.edu</FacultyEmail>
</OracleSelectResult>

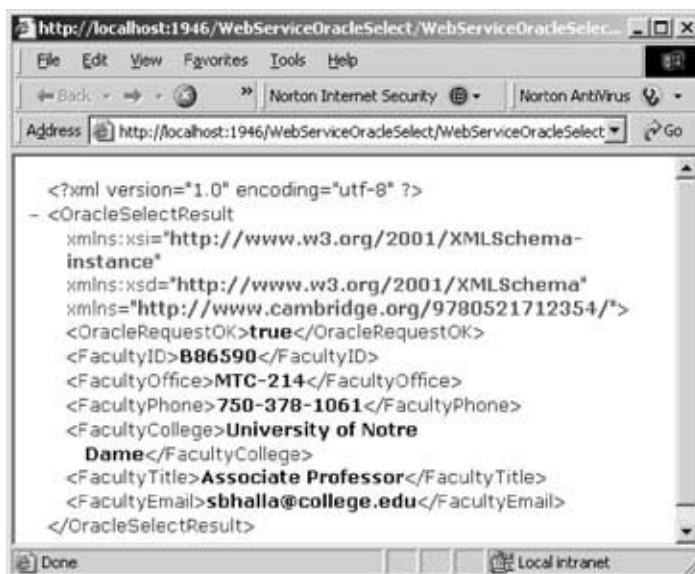
```

Figure 8.139. The running result of the Web method GetOracleSelect.

our sample Oracle database to retrieve the detailed information for the selected faculty member. The running result is shown in Figure 8.140.

The detailed information for the selected faculty member 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 the Back button to return to the initial page. Click the Web method GetOracleSelectDataSet() to open its built-in parameter-input Web interface. Enter the selected faculty name Ying Bai, and click the Invoke



The screenshot shows a Microsoft Internet Explorer window with the URL <http://localhost:1946/WebServiceOracleSelect/WebServiceOracleSel...>. The page content displays an XML document representing the selected faculty member's information:

```

<?xml version="1.0" encoding="utf-8" ?>
- <OracleSelectResult
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.cambridge.org/9780521712354/">
  <OracleRequestOK>true</OracleRequestOK>
  <FacultyID>B86590</FacultyID>
  <FacultyOffice>MTC-214</FacultyOffice>
  <FacultyPhone>750-378-1061</FacultyPhone>
  <FacultyCollege>University of Notre
    Dame</FacultyCollege>
  <FacultyTitle>Associate Professor</FacultyTitle>
  <FacultyEmail>sbhalla@college.edu</FacultyEmail>
</OracleSelectResult>

```

Figure 8.140. The running result of the Web method GetOracleSelectSP.

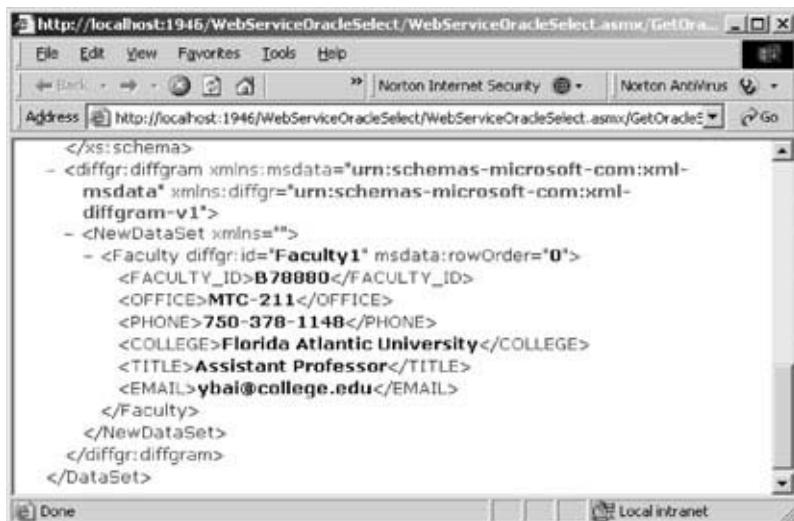


Figure 8.141. The running result of the Web method GetOracleSelectDataSet.

button to run this method. The running result of this Web method is shown in Figure 8.141.

As shown in Figure 8.141, the detailed information that is contained in a DataSet for the selected faculty member Ying Bai is retrieved and displayed in the XML tag format in this built-in Web interface. Our Web Service project is very successful. Click the Close button for both built-in Web interfaces to terminate our Web Service project.

The completed Web Service project WebServiceOracleSelect is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

## 8.9 BUILD WEB SERVICE CLIENTS TO CONSUME THE WEB SERVICE WEBSERVICEORACLESELECT

To consume this Web Service project, one can develop either a Windows-based or a Web-based Web Service client project. There is no significant difference between building a client project to consume a Web Service to access an SQL Server database and building a client project to consume a Web Service to access an Oracle database. For example, you can use any client projects, such as either WinClientSQLSelect or WebClientSQLSelect, which we developed in the previous sections, to consume the 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 dialog, run the desired Web Service project, copy the URL from that running Web Service

- project, and paste it to the URL box in the Add Web Reference dialog box in the client project.
3. Change the Web reference name for all data components used in the client project.
  4. Change the names of the base class and derived class located in the Web reference.
  5. Change the names of all Web methods located in the Web reference.

Two completed client projects, WinClientOracleSelect, which is Windows-based, and WebClientOracleSelect, which is Web-based, can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

Next, let's develop a Web Service project to perform the data insertion for the Oracle database.

## **8.10 BUILD AN ASP.NET WEB SERVICE PROJECT TO INSERT DATA INTO AN ORACLE DATABASE**

Basically, the procedure to build an ASP.NET Web Service to insert data into the SQL Server database is very similar to the procedure to build an ASP.NET Web Service to insert data into the Oracle database. The main differences are listed below:

1. The connection string defined in the Web configuration file Web.config
2. The namespace directories listed at the top of each Web Service page
3. The stored procedures used by each Web Service page
4. The protocol of the data query string used by each Web Service page
5. The nominal names of dynamic parameters for the Parameters collection object

These five distinguishing points exist between the procedures to build a Web Service to insert data into these two databases.

Based on the discussion and analysis we made in Section 8.8 as well as the similarity between the SQL Server and Oracle databases, we will 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 to make it into our new Web Service project WebServiceOracleInsert.

### **8.10.1 Build a Web Service Project – WebServiceOracleInsert**

In this section, we try to modify an existing Web Service project WebServiceSQLInsert to make it into 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 8** under the root directory if you have not already done so. Open Internet Explorer and browse to our desired source Web Service project WebServiceSQLInsert, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**, and copy and paste it into our new folder, **Chapter 8**. Rename it WebServiceOracleInsert, and 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 and 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 derived class file from SQLInsertResult.vb to OracleInsertResult.vb.
4. 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 2005 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 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 SQLRequestOK to OracleRequestOK
- Change the name of the second member data from SQLRequestError to OracleRequestError

Go to the File|Save All menu item to save these modifications.

### 8.10.2 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 = SYSTEM;Password = reback;" />
```

---

The Oracle database server XE is used for the server name, the user ID is the default value SYSTEM, and the password is determined by the user when installing the Oracle Database 10g Express Edition in the local computer.

### 8.10.3 Modify the Namespace Directories

First, we need to add an Oracle Data Provider Reference to our Web Service project. To do that, right-click our new project icon WebServiceOracleInsert from the Solution Explorer window, and then select the item Add Reference from the popup menu to open the Add Reference dialog box. Browse down the list until you

find the item System.Data.OracleClient, click to select it, and then click the OK button to add it into our project.

Now double-click our code-behind page WebServiceOracleInsert.vb to open it. On the page, change the last namespace line from

---

**Imports System.Data.SqlClient**

---

to

---

---

**Imports System.Data.OracleClient**

---

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 developed in this Web Service project combined with those five differences listed above.

#### **8.10.4 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 subroutines SQLConn() and ReportError()
5. The names of the dynamic parameters

Let's perform those modifications step by step according to this sequence.

- A. Rename this Web method to SetOracleInsertSP, and change the name of the returned class to OracleInsertBase.
- 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 change the 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 OracleType.VarChar. Change the data type for the last two input parameters from SqlDbType.Text to OracleType.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.

Your modified Web method SetOracleInsertSP() should match the one shown in Figure 8.142. All modifications are highlighted in bold.

```

WebServiceOracleInsert                                     SetOracleInsertSP
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Data
Imports System.Data.OracleClient
<WebService(Namespace:="http://www.cambridge.org/9780521712354")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class WebServiceOracleInsert
    Inherits System.Web.Services.WebService
    <WebMethod()>
    Public Function SetOracleInsertSP(ByVal FacultyName As String, ByVal CourseID As String, ByVal Course As _
        String, ByVal Schedule As String, ByVal Classroom As String, ByVal Credit As Integer, ByVal Enroll As Integer) As OracleInsertBase
        Dim cmdString As String = "InsertFacultyCourse"
        Dim oraConnection As New OracleConnection
        Dim SetOracleResult As New OracleInsertBase
        Dim oraCommand As New OracleCommand
        Dim intInsert As Integer
        SetOracleResult.OracleInsertOK = True
        oraConnection = OracleConn()
        If oraConnection Is Nothing Then
            SetOracleResult.OracleInsertError = "Database connection is failed"
            ReportError(SetOracleResult)
            Return Nothing
        End If
        oraCommand.Connection = oraConnection
        oraCommand.CommandType = CommandType.StoredProcedure
        oraCommand.CommandText = cmdString
        oraCommand.Parameters.Add("FacultyName", OracleType.VarChar).Value = FacultyName
        oraCommand.Parameters.Add("CourseID", OracleType.VarChar).Value = CourseID
        oraCommand.Parameters.Add("Course", OracleType.VarChar).Value = Course
        oraCommand.Parameters.Add("Schedule", OracleType.VarChar).Value = Schedule
        oraCommand.Parameters.Add("Classroom", OracleType.VarChar).Value = Classroom
        oraCommand.Parameters.Add("Credit", OracleType.Number).Value = Credit
        oraCommand.Parameters.Add("Enroll", OracleType.Number).Value = Enroll
        intInsert = oraCommand.ExecuteNonQuery()
        oraCommand.Dispose()
        oraCommand = Nothing
        If Not oraConnection Is Nothing Then oraConnection.Close()
        If intInsert = 0 Then
            SetOracleResult.OracleInsertError = "Data insertion is failed"
            ReportError(SetOracleResult)
        End If
        Return SetOracleResult
    End Function

```

Figure 8.142. The modified Web method SetOracleInsertSP.

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 change the return class name from SqlConnection to OracleConnection. Also change the connection string from sql\_conn to ora\_conn.
- B. Change the data type of the returned connection object to OracleConnection.

Perform the following modifications to the subroutine ReportError():

- A. Change the data type of the passed argument from SQLInsertBase to OracleInsertBase.
- B. Change the name of the first member data from SQLInsertOK to OracleInsertOK.
- C. Change the name of the second member data from SQLInsertError to OracleInsertError.

Your modified subroutines OracleConn() and ReportError() should match the ones shown in Figure 8.143. The modifications are highlighted in bold.

For the stored procedure InsertFacultyCourse, we do not need to perform any modification to this since we successfully developed this stored procedure in Section 5.7.2.1 in Chapter 5. Therefore we can directly use this procedure without any problem. Refer to Section 5.7.2.1 for more detailed information and discussion about the development issues for this stored procedure.

### 8.10.5 Modify the Web Method GetSQLInsert and Related Subroutines

The functionality of this Web method is to retrieve all course\_id, including 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 retrieve and display all course\_id in a list box control in the client project.

A	WebServiceOracleInsert	▼	▼
	<pre>Protected Function<b>OracleConn()</b> As<b>OracleConnection</b>     Dim cmdString As String = ConfigurationManager.ConnectionStrings("ora_conn").ConnectionString     Dim conn As New <b>OracleConnection</b>     conn.ConnectionString = cmdString     conn.Open()     If conn.State &lt;&gt; ConnectionState.Open Then         MsgBox("Database Open is failed")         conn = Nothing     End If     Return conn End Function</pre>	▼	▼
B			
C	<pre>Protected Sub ReportError(ByVal ErrSource As <b>OracleInsertBase</b>)     ErrSource.<b>OracleInsertOK</b> = False     MsgBox(ErrSource.<b>OracleInsertError</b>) End Sub</pre>		
D			
E			

Figure 8.143. Modified subroutines OracleConn and ReportError.

A	WebServiceOracleInsert
B	<pre>&lt;WebMethod()&gt;_ Public Function GetOracleInsert(ByVal FacultyName AsString) As OracleInsertBase     Dim cmdString As String = "SELECT Course.course_id FROM Course, Faculty " +         "WHERE (Course.faculty_id = Faculty.faculty_id) AND (Faculty.name =: fname)" </pre>
C	<pre>    Dim oraConnection As New OracleConnection     Dim GetOracleResult As New OracleInsertBase     Dim oraCommand As New OracleCommand     Dim oraReader As OracleDataReader</pre>
D	<pre>    GetOracleResult.OracleInsertOK = True     oraConnection = OracleConn()</pre>
E	<pre>If oraConnection Is Nothing Then     GetOracleResult.OracleInsertError = "Database connection is failed"     ReportError(GetOracleResult)     Return Nothing End If</pre>
F	<pre>oraCommand.Connection = oraConnection oraCommand.CommandType = CommandType.Text oraCommand.CommandText = cmdString oraCommand.Parameters.Add("fname", OracleType.VarChar).Value = FacultyName oraReader = oraCommand.ExecuteReader</pre>
G	<pre>If oraReader.HasRows = True Then     Call FillCourseReader(GetOracleResult, oraReader) Else</pre>
H	<pre>    GetOracleResult.OracleInsertError = "No matched course Found"     ReportError(GetOracleResult) End If</pre>
	<pre>If Not oraReader Is Nothing Then oraReader.Close() If Not oraConnection Is Nothing Then oraConnection.Close() oraCommand.Dispose() Return GetOracleResult End Function</pre>

Figure 8.144. The modified Web method – GetOracleInsert.

Recall that in Section 4.18.6 in Chapter 4, 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 that there is no faculty name column available in the Course table, and each course or course\_id is related only 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 joined-table query is a desired method to complete this functionality.

Open this Web method and perform the modifications shown in Figure 8.144 to this method. The modifications are highlighted in bold.

Let's take a closer look at these modifications to see how they work.

- A. Change the name of this Web method from GetSQLInsert to GetOracleInsert. Also change the name of the returned instance from SQLInsertBase to OracleInsertBase.
- B. Modify the query string to match the ANSI 89 standard. Recall that we developed a joined-table query string for the SQL Server database using the ANSI 92 standard in Section 4.18.6 in Chapter 4. Since the ANSI 89 standard is still being used in the Oracle database, we need to modify this joined-table query string by using that standard.

- C. Change the prefix 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 OracleType.VarChar.
- G. Change the names of two passed arguments 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 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 8.145.

### 8.10.6 Modify the Web Method SQLInsertDataSet

The functionality of this Web method is to use an insert query to perform the course insertion and then retrieve the newly inserted data and store it in a DataSet that will be returned to the calling procedure. Perform the modifications shown in Figure 8.146 to this Web method:

- A. Change the name of this method from SQLInsertDataSet to OracleInsertDataSet.

WebServiceOracleInsert	▼	FillCourseReader	▼
A Protected Sub FillCourseReader(ByRef sResult As <b>Oracle</b> InsertBase, ByVal sReader As <b>Oracle</b> DataReader) Dim pos As Integer While sReader.Read() sResult.CourseID(pos) = Convert.ToString(sReader. <b>GetOracleString(0)</b> )     'the 1st column is course_id pos = pos + 1 End While End Sub			

Figure 8.145. The modified subroutine FillCourseReader.

	WebServiceOracleInsert	▼	OracleInsert DataSet	▼
A	<WebMethod()> _			
B	Public Function <b>OracleInsertDataSet</b> (ByVal FacultyName As String, ByVal CourseID As String, _ ByVal Course As String, ByVal Schedule As String, ByVal Classroom As String, ByVal Credit As Integer, _ ByVal Enroll As Integer) As DataSet			
C	Dim cmdString As String = "INSERT INTO Course VALUES " + _ "(:course_id,:course,:credit,:classroom,:schedule,:enrollment,:faculty_id)"			
D	Dim oraConnection As New <b>OracleConnection</b> Dim SetOracleResult As New <b>OracleInsertBase</b> Dim oraCommand As New <b>OracleCommand</b> Dim CourseAdapter As New <b>OracleDataAdapter</b> Dim dsCourse As New <b>DataSet</b> Dim intResult As Integer Dim FacultyID As String  SetOracleResult. <b>OracleInsertOK</b> = True <b>oraConnection</b> = <b>OracleConn()</b>			
E	If <b>oraConnection</b> Is Nothing Then			
F	SetOracleResult. <b>OracleInsertError</b> = "Database connection is failed" ReportError(SetOracleResult)			
G	Return Nothing			
H	End If			
I	<b>oraCommand.Connection</b> = <b>oraConnection</b> <b>oraCommand.CommandType</b> = <b> CommandType.Text</b>			
J	<b>oraCommand.CommandText</b> = "SELECT faculty_id FROM Faculty WHERE name =: Name" <b>oraCommand.Parameters.Add("Name", OracleType.VarChar).Value</b> = FacultyName FacultyID = <b>oraCommand.ExecuteScalar()</b>			
K	<b>oraCommand.Parameters.Clear()</b> 'very important			
L	<b>oraCommand.CommandText</b> = cmdString			
M	<b>oraCommand.Parameters.Add("faculty_id", OracleType.VarChar).Value</b> = FacultyID <b>oraCommand.Parameters.Add("course_id", OracleType.VarChar).Value</b> = CourseID <b>oraCommand.Parameters.Add("course", OracleType.VarChar).Value</b> = Course <b>oraCommand.Parameters.Add("schedule", OracleType.VarChar).Value</b> = Schedule <b>oraCommand.Parameters.Add("classroom", OracleType.VarChar).Value</b> = Classroom <b>oraCommand.Parameters.Add("credit", OracleType.Number).Value</b> = Credit <b>oraCommand.Parameters.Add("enrollment", OracleType.Number).Value</b> = Enroll			
N	CourseAdapter.InsertCommand = <b>oraCommand</b> intResult = CourseAdapter.InsertCommand.ExecuteNonQuery() If intResult = 0 Then			
O	SetOracleResult. <b>OracleInsertError</b> = "No matched course found" ReportError(SetOracleResult)			
P	End If			
Q	<b>oraCommand.Parameters.Clear()</b> 'very important			
R	<b>oraCommand.CommandText</b> = "SELECT * FROM Course WHERE faculty_id =: FacultyID" <b>oraCommand.Parameters.Add("FacultyID", OracleType.VarChar).Value</b> = FacultyID			
S	CourseAdapter.SelectCommand = <b>oraCommand</b> CourseAdapter.Fill(dsCourse, "Course") CourseAdapter.Dispose() CourseAdapter = Nothing			
T	If <b>oraConnection</b> IsNot Nothing Then <b>oraConnection.Close()</b> <b>oraCommand.Dispose()</b> Return dsCourse			
U	End Function			

Figure 8.146. The modified Web method OracleInsertDataSet.

- B. Change the query string by replacing the @ symbol before each input parameter with the colon operator :, which 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 assignment operator LIKE @ before the nominal name Name with the Oracle assignment operator =:.
- G. Modify the nominal name of the input parameter by removing the @ symbol before the nominal name Name. Also change the data type of this input parameter from SqlDbType.Text to OracleType.VarChar.
- H. Since the next query needs to use different parameters, the Parameters collection must be cleaned up using the Clear() method to remove the previous parameter objects. This is very important. An Oracle database exception will be encountered if this cleaning job is not performed.
- I. Modify the nominal names for all seven input parameters by removing the @ symbol before each nominal name. Also change the data type of the top five input parameters from SqlDbType.Text to OracleType.VarChar, and change the data type of the last two input parameters from SqlDbType.Text to OracleType.Number.
- J. Change the prefix from sql to ora for all the following data objects. Also change the name of the returned instance from SetSQLResult to SetOracleResult. Change the second member data from SQLInsertError to OracleInsertError.
- K. This step has same function as the step H. This cleaning job is unnecessary in SQL Server database operations.
- L. Modify the dynamic name for the input parameter by replacing the SQL assignment operator LIKE @ before the nominal name FacultyID with the Oracle assignment operator =:.
- M. Modify the nominal name of the input parameter by removing the @ symbol before the nominal name FacultyID. Also change the data type of this input parameter from SqlDbType.Text to OracleType.VarChar.
- N. Change the prefix from sql to ora for all data objects used in this method.

### **8.10.7 Modify the Web Method GetSQLInsertCourse and Related Subroutines**

The functionality 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 create a new package that contains a stored procedure.

Open this Web method and perform the following modifications 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. WebSelectCourseSP is a package, and SelectCourse is a stored procedure.
- C. Change the prefix from Sql to Oracle for all data classes, and from sql to ora for all data objects used in this method. Also change the name of the returned instance from GetSQLResult to GetOracleResult. Change the first member data from SQLInsertOK to OracleInsertOK.
- D. Add two OracleParameter objects, paramCourseID and paramCourseInfo. Because some differences exist between the SQL Server and Oracle databases, we need to use 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. Note an important point for the second parameter, paramCourseInfo. The data type of this parameter is Cursor, and the Direction is Output. Both values are very important to this parameter and must be assigned exactly as we did here. Otherwise a running exception will be encountered when the project runs.
- G. Add two statements to add two OracleParameter objects to the Command object.
- H. Change the names of two passed arguments to the 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.

Your modified Web method `GetOracleInsertCourse()` should match the one shown in Figure 8.147, and all related modifications are highlighted in bold.

The modifications to the related subroutine `FillCourseDetail()` are simple, and the only modifications are to change the data type of the two passed arguments `sResult` and `sReader`. Change the data type of the first argument from `SQLInsertBase` to `OracleInsertBase`, and change the data type for the second argument from `SqlDataReader` to `OracleDataReader`.

Now let's create an Oracle package `WebSelectCourseSP` that contains the stored procedure `SelectCourse` to perform this course detailed information query. In this section we use the Object Browser page in Oracle Database 10g XE to build this package.

Open the Oracle Database 10g XE home page by going to Start>All Programs|Oracle Database 10g Express Edition|Go To Database Home Page. Enter the correct user ID and password to complete the login process. Then click the Object Browser and select Create|Package to open the Create Package window.

WebServiceOracleInsert	▼	GetOracleInsertCourse	▼
<pre> &lt;WebMethod()&gt; Public Function GetOracleInsertCourse(ByVal CourseID As String) As OracleInsertBase     Dim cmdString As String = "WebSelectCourseSP.SelectCourse"     Dim oraConnection As New OracleConnection     Dim GetOracleResult As New OracleInsertBase     Dim oraReader As OracleDataReader     Dim paramCourseID As New OracleParameter     Dim paramCourseInfo As New OracleParameter     Get OracleResult.OracleInsertOK = True     oraConnection = OracleConn()     If oraConnection Is Nothing Then         Get OracleResult.OracleInsertOK = False         GetOracleResult.OracleInsertError = "Database connection is failed"         ReportError(GetOracleResult)         Return Nothing     End If     paramCourseID.ParameterName = "CourseID"     paramCourseID.OracleType = OracleType.VarChar     paramCourseID.Value = CourseID     paramCourseInfo.ParameterName = "CourseInfo"     paramCourseInfo.OracleType = OracleType.Cursor     paramCourseInfo.Direction = ParameterDirection.Output      'this is very important     Dim oraCommand = New OracleCommand(cmdString, oraConnection)     oraCommand.CommandType = CommandType.StoredProcedure     oraCommand.Parameters.Add(paramCourseID)     oraCommand.Parameters.Add(paramCourseInfo)     oraReader = oraCommand.ExecuteReader     If oraReader.HasRows = True Then         Call FillCourseDetail(GetOracleResult, oraReader)     Else         GetOracleResult.OracleInsertError = "No matched course found"         ReportError(GetOracleResult)     End If     oraReader.Close()     oraReader = Nothing     oraConnection.Close()     oraCommand.Dispose()     Return GetOracleResult End Function </pre>			

Figure 8.147. The modified Web method GetOracleInsertCourse.

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 clicking the Next button to open the Name page, which is shown in Figure 8.148.

Enter the package name WebSelectCourseSP into the name box, and click the Next button to go to the specification page.

A default package specification prototype, which includes a procedure and a function, is provided in this page, and you need to use your real specifications to replace those default items. Since we don't need any function for our application, remove the default function prototype, and change the default procedure name from test to our procedure name – SelectCourse. Your code for the specification page should match that shown in Figure 8.149.

In line 2, we defined the returned data type as a CURSOR\_TYPE by using

**TYPE CURSOR\_TYPE IS REF CURSOR;**

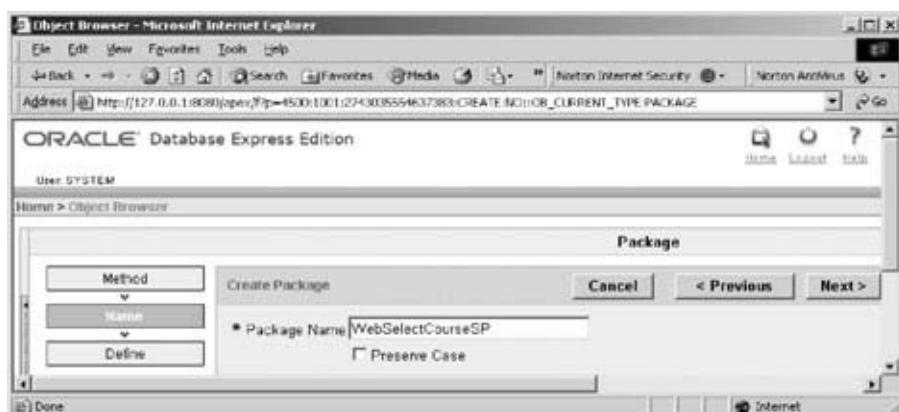


Figure 8.148. The Name page of the Package window.

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 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 the keyword **out**. Each PL/SQL statement must end in a semicolon, and this rule also applies to the **end** statement.

You can click the Compile button to compile this specification block if you like. Next we need to create the body block of this package. Click the Finish button to complete this step. Click the Body tab to open the Body page, which is shown in Figure 8.150.

Click the Edit button to begin to create our body part. Enter the PL/SQL code shown in Figure 8.151 into this body.

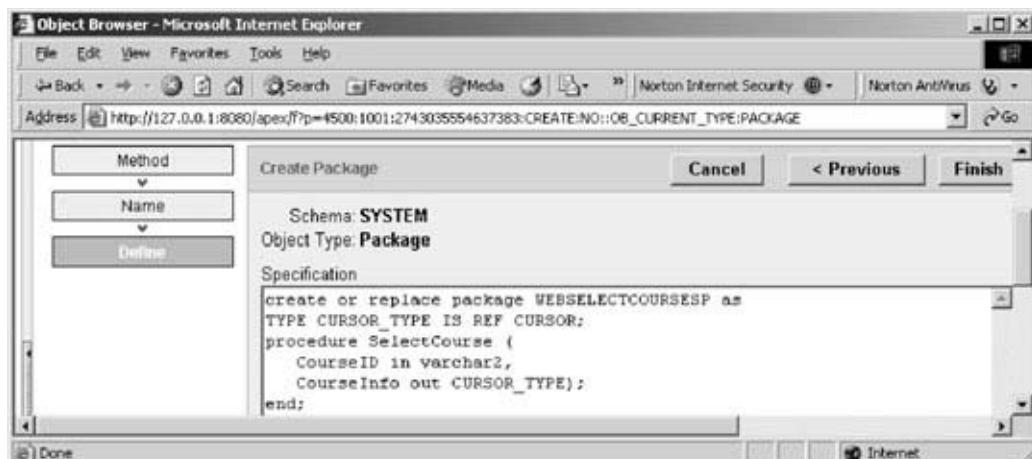


Figure 8.149. The code for the Specification page.



Figure 8.150. The Body page of the package.

The procedure prototype is redeclared in line 2. Starting from the **begin**, our real SQL statements are included in lines 6 and 7. The **OPEN 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 4.18.9 in Chapter 4. There are two **end** commands applied at the end of this package. The first one is used to end the stored procedure, and the second one is for the package.

Now let's compile our package by clicking the Compile button. A successful compiling information

#### **PL/SQL code successfully compiled (11:27:33)**

is displayed if this package is bug-free, as shown in Figure 8.152.

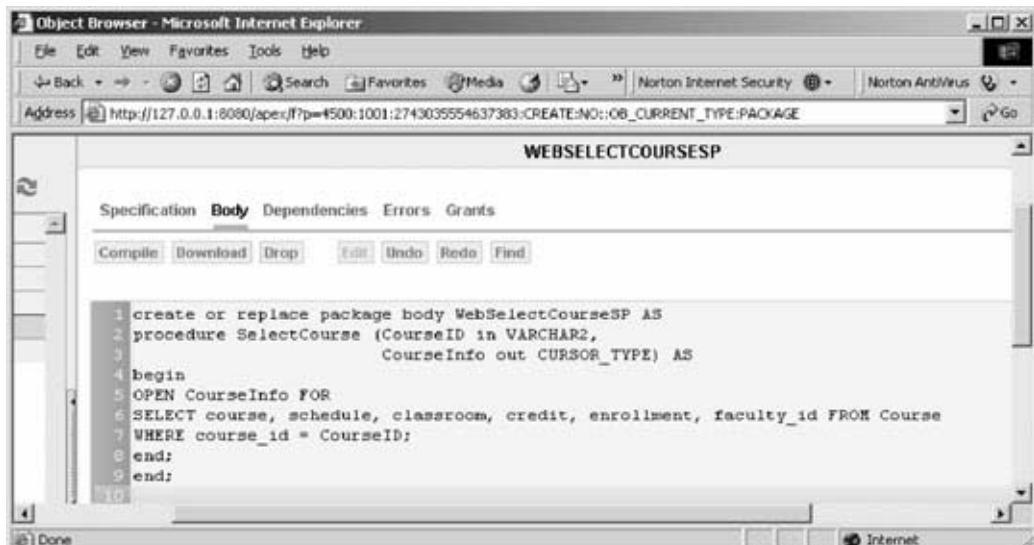


Figure 8.151. The code for the Body part of the package.

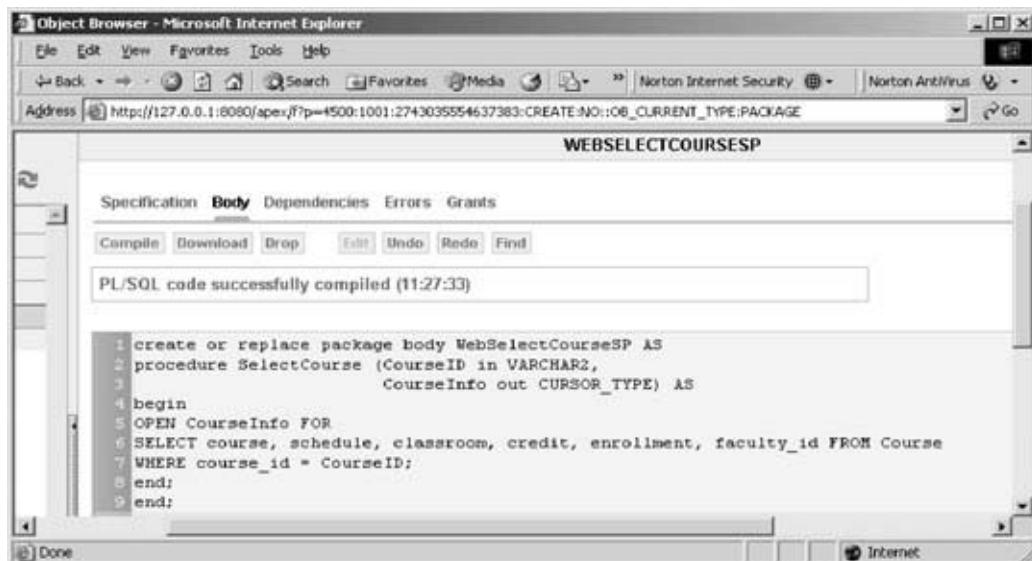


Figure 8.152. The compiled code for the body part of the package.

At this point, we have finished the development of our Oracle package and all modifications to our new Web Service project. Close Oracle Database 10g XE by clicking 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 functionality. Click the Start Debugging button to run our Web Service.

First, let's test the Web method SetOracleInsertSP() by clicking this method. Enter the input parameters shown in Figure 8.153 into the associated Value boxes in the opened parameter-input page, and click the Invoke button to run this method.

The running result of this Web method is shown in Figure 8.154.

It can be found that the running result of this Web method is fine, and this can be confirmed by the returned status variable OracleInsertOK whose value is true. Another way to confirm this data insertion is to open our sample database CSE\_DEPT from Oracle Database 10g XE and open the Course table using the Browser Object to check this newly inserted course. An example of the Course table is shown in Figure 8.155, and you can see that the newly inserted course CSE-575 is located at the bottom.

Now let's test the second Web method, GetOracleInsert(), by clicking it. On the opened parameter-input page, enter the faculty name Ying Bai into the Value box, and click the Invoke button to run this method. The running result is shown in Figure 8.156.

It can be seen 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 button to return to our Web start page. Click the Web method OracleInsertDataSet() to test this method. Enter the parameters shown in Figure 8.157 as the new course information in the

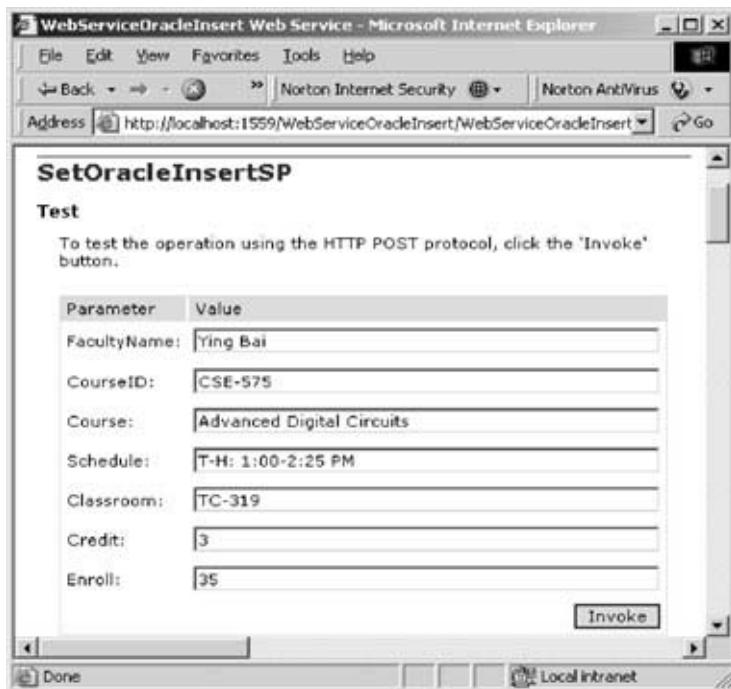


Figure 8.153. The built-in parameter-input Web interface.

associated Value boxes in the opened parameter-input page and click the Invoke button to run this method.

The running result of this Web method is shown in Figure 8.158.

It can be seen in Figure 8.158 that the newly inserted course CSC-532 is indeed added into the Course table in our sample Oracle database.

```

<instance>
  <xsi:nil></xsi:nil>
  <@xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <@xmlns='http://www.cambridge.org/9780521712354'>
  <OracleInsertOK>true</OracleInsertOK>
  - <CourseID>
    <string xsi:nil="true" />
    <string xsi:nil="true" />
  </CourseID>
  <Credit>0</Credit>
  <Enrollment>0</Enrollment>
</OracleInsertBase>

```

Figure 8.154. The running result of the Web method SetOracleInsertSP.

COURSE							
	Table	Data	Indexes	Model	Constraints	Grants	Statistics
	Query	Count Rows	Insert Row				UI Defaults
EDIT	COURSE_ID	COURSE	CREDIT	CLASSROOM	SCHEDULE	ENROLLMENT	FACULTY_ID
	CSE-433	Digital Signal Processing	3	TC-206	T-H 2:00-3:25 PM	18	H99118
	CSC-132B	Introduction to Programming	3	TC-302	T-H 1:00-2:25 PM	21	B78880
	CSE-436	Automatic Control and Design	3	TC-305	M-W-F, 10:00-10:55 AM	29	J33486
	CSE-437	Operating Systems	3	TC-303	T-H 1:00-2:25 PM	17	A77587
	CSC-234A	Data Structure & Algorithms	3	TC-302	M-W-F, 9:00-9:55 AM	25	D78000
	CSE-439	Special Topics in CSE	3	TC-206	M-W-F, 10:00-10:55 AM	22	J33486
	CSE-575	Advanced Digital Circuits	3	TC-319	T-H 1:00-2:25 PM	35	B78880

Figure 8.155. The Course table.

Finally, let's test the Web method GetOracleInsertCourse(). Close the current running result page window, click the Back button to return to our initial Web page, and click 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 the Invoke button to run this method. The running result is shown in Figure 8.159.

```

<OracleInsertOK>true</OracleInsertOK>
- <CourseID>
  <string>CSE-434</string>
  <string>CSE-438</string>
  <string>CSC-132B</string>
  <string>CSC-234A</string>
  <string>CSE-575</string>
  <string xsi:nil="true" />
  <string xsi:nil="true" />
  <string xsi:nil="true" />
  <string xsi:nil="true" />
  <string xsi:nil="true" />
</CourseID>
<Credit>0</Credit>
<Enrollment>0</Enrollment>
</OracleInsertBase>

```

Figure 8.156. The running result of the Web method GetOracleInsert.

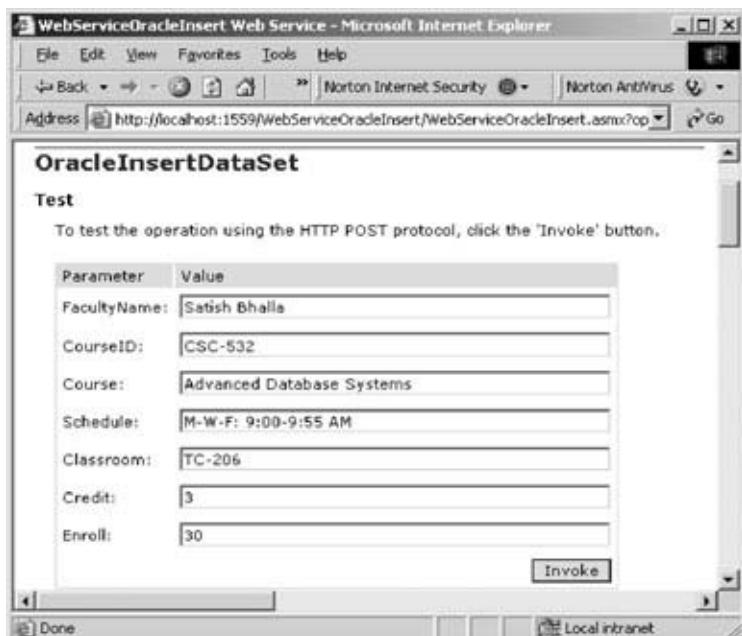


Figure 8.157. The parameter-input page.

It can be found that the detailed information about the course CSC-532 such as the course name, classroom, schedule, credit, and enrollment has been retrieved and displayed in this built-in Web interface in XML tag format.

At this point, we have finished testing all Web methods developed in this Web Service project. The completed Web Service project WebServiceOracleInsert can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

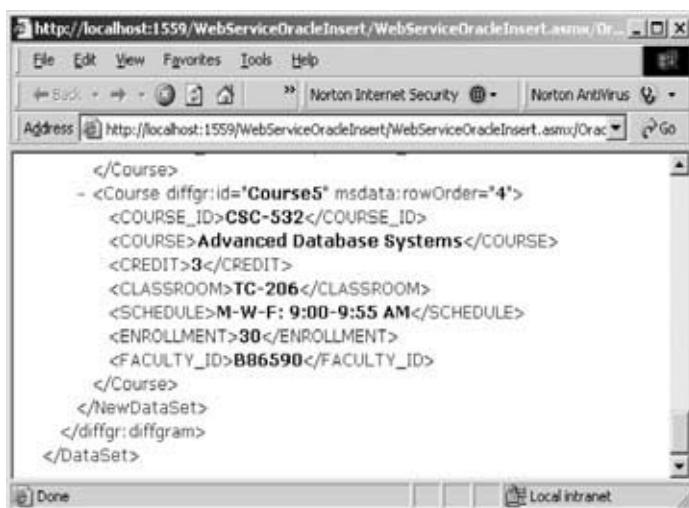


Figure 8.158. The running result of the Web method OracleInsertDataSet.



Figure 8.159. The running result of the method GetOracleInsertCourse.

## 8.11 BUILD WEB SERVICE CLIENTS TO CONSUME THE WEB SERVICE WEBSERVICEORACLEINSERT

To consume this Web Service project, one can develop either a Windows-based or a Web-based Web Service client project. There is no significant difference between building a client project to consume a Web Service to access an SQL Server database and building a client project to consume a Web Service to access an Oracle database. For example, you can use any client projects, 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 dialog, run the desired Web Service project, copy the URL from that running Web Service project, and paste it to the URL box in the Add Web Reference dialog box in the client project.
3. Change the Web reference name for all data components used in the client project.
4. Change the names of the base class and derived class located in the Web reference.
5. Change the names of all Web methods located in the Web reference.

Two completed client projects, WinClientOracleInsert, which is Windows-based, and WebClientOracleInsert, which is Web-based, can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

## **8.12 BUILD AN ASP.NET WEB SERVICE TO UPDATE AND DELETE DATA IN AN ORACLE DATABASE**

Basically, the procedure to build an ASP.NET Web Service to update and delete data in the SQL Server database is very similar to the procedure to build an ASP.NET Web Service to update and delete data in the Oracle database. The main differences are listed below:

1. The connection string defined in the Web configuration file Web.config
2. The namespace directories listed at the top of each Web Service page
3. The stored procedures used by each Web Service page
4. The protocol of the data query string used by each Web Service page
5. The nominal names of dynamic parameters for the Parameters collection object

These five distinguishing points exist between the procedures to build a Web Service to update and delete data in these two kinds of databases.

Based on the discussion and analysis in Section 8.8 as well as the similarity between the SQL Server and Oracle databases, we will develop our Web Service projects to update and delete data in the Oracle database by modifying some existing Web Service projects. In this section, we concentrate on the modifications to the Web Service project WebServiceSQLUpdateDelete to make it into our new Web Service project, WebServiceOracleUpdateDelete.

### **8.12.1 Build a Web Service Project – WebServiceOracleUpdateDelete**

In this section, we will modify an existing Web Service project WebServiceSQLUpdateDelete to make it into our new Web Service project WebServiceOracleUpdateDelete and allow it to update and delete data in the Oracle database.

Open the Windows Explorer and create a new folder, **Chapter 8**, under the root directory if you have not already done so. Open Internet Explorer and then browse to our desired source Web Service project WebServiceSQLUpdateDelete, which is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**, and copy and paste it into our new folder **Chapter 8**. Rename this project WebServiceOracleUpdateDelete, and 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 2005 and our new Web Service project WebServiceOracleUpdateDelete to perform the associated modifications to the contents of the files we renamed above. First, let's perform the modifications to our main Web Service page, WebServiceOracleUpdateDelete.asmx. Open this page by

double-clicking 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 = “WebService-OracleUpdateDelete”**

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 SQLError to OracleError

Go to the File|Save All menu item to save these modifications.

### 8.12.2 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 = SYSTEM;Password = reback;" />
```

---

The Oracle database server XE is used for the server name, the user ID is the default value SYSTEM, and the password is determined by the user when installing the Oracle Database 10g Express Edition in the local computer.

### 8.12.3 Modify the Namespace Directories

First, we need to add an Oracle Data Provider Reference to our Web Service project. To do that, right-click our new project icon WebServiceOracleUpdateDelete from the Solution Explorer window, and then select the item Add Reference from the popup menu to open the Add Reference dialog box. Browse down the list until you find the item System.Data.OracleClient, click to select it, and then click the OK button to add it into our project.

Now double-click our code-behind page WebServiceOracleUpdateDelete.vb to open it. On the opened page, change the last namespace line from Imports System.Data.SqlClient to Imports System.Data.OracleClient.

Also change the name of our Web Service class, which is located after the accessing mode Public Class, from WebServiceSQLUpdateDelete to WebServiceOracleUpdateDelete.

Next, we will perform the necessary modifications to four Web methods developed in this Web Service project combined with the five differences listed above.

### **8.12.4 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.

- A. Rename this Web method OracleUpdateSP, and change the name of the returned class to OracleBase.
- B. Modify the content of the query string by changing the name of the stored procedure from dbo.WebUpdateCourseSP to UpdateCourse\_SP. The former is an SQL Server stored procedure, and the latter is an Oracle stored procedure that will be developed in the next section.
- 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 change the member data from SQLOK to OracleOK.
- E. Change the name of the subroutine from SQLConn to OracleConn.
- F. Change the prefix from sql to ora for all data objects.
- G. Modify the nominal names for all seven input parameters to the stored procedure by removing the @ symbol before each nominal name. Also change the data type of the top five input parameters from SqlDbType.Text to OracleType.VarChar. Change the data type for the last two input parameters from SqlDbType.Text to OracleType.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.

Your modified Web method OracleUpdateSP() should match the one shown in Figure 8.160, and all related modifications are highlighted in bold.

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 change the return class name from SqlConnection to OracleConnection. Also change the connection string from sql\_conn to ora\_conn.

WebServiceOracleUpdateDelete	▼	OracleUpdateSP	▼
Imports System.Web Imports System.Web.Services Imports System.Web.Services.Protocols Imports System.Data Imports System.Data.OracleClient  <WebService(Namespace:="http://www.cambridge.org/9780521712354")> _ <WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _ <Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _ Public Class WebService <b>OracleUpdateDelete</b> Inherits System.Web.Services.WebService  <WebMethod()> _ Public Function <b>OracleUpdateSP</b> (ByVal FacultyName As String, ByVal CourseID As String, ByVal Course As String, _ ByVal Schedule As String, ByVal Classroom As String, ByVal Credit As Integer, ByVal Enroll As Integer) As OracleBase <b>A</b> Dim cmdString As String = " <b>UpdateCourse_SP</b> " <b>B</b> Dim oraConnection As New <b>OracleConnection</b> <b>C</b> Dim OracleResult As New <b>OracleBase</b> Dim OracleCommand As New <b>OracleCommand</b> Dim intUpdate As Integer <b>D</b> OracleResult.OracleOK = True <b>E</b> oraConnection = <b>OracleConn()</b> <b>F</b> If oraConnection Is Nothing Then <b>G</b> <b>OracleResult.OracleError</b> = "Database connection is failed" ReportError( <b>OracleResult</b> ) Return Nothing End If <b>H</b> oraCommand.Connection = oraConnection oraCommand.CommandType = CommandType.StoredProcedure oraCommand.CommandText = cmdString  <b>I</b> oraCommand.Parameters.Add("FacultyName", <b>OracleType.VarChar</b> ).Value = FacultyName oraCommand.Parameters.Add("inCourseID", <b>OracleType.VarChar</b> ).Value = CourseID oraCommand.Parameters.Add("inCourse", <b>OracleType.VarChar</b> ).Value = Course oraCommand.Parameters.Add("inSchedule", <b>OracleType.VarChar</b> ).Value = Schedule oraCommand.Parameters.Add("inClassroom", <b>OracleType.VarChar</b> ).Value = Classroom oraCommand.Parameters.Add("inCredit", <b>OracleType.Number</b> ).Value = Credit oraCommand.Parameters.Add("inEnroll", <b>OracleType.Number</b> ).Value = Enroll intUpdate = oraCommand.ExecuteNonQuery() oraCommand.Dispose() If Not oraConnection Is Nothing Then oraConnection.Close() oraConnection = Nothing If intUpdate = 0 Then <b>I</b> <b>OracleResult.OracleError</b> = "Data updating is failed" ReportError( <b>OracleResult</b> ) End If Return <b>OracleResult</b> End Function			

Figure 8.160. The modified Web method – OracleUpdateSP.

- B. Change the data type of the returned connection object to OracleConnection.

Perform the following modifications to the subroutine ReportError():

- A. Change the data type of the passed argument from SQLBase to OracleBase.
- B. Change the name of the first member data from SQLOK to OracleOK.
- C. Change the name of the second member data from SQLError to OracleError.

The screenshot shows a Microsoft Visual Studio code editor window. The title bar says "WebServiceOracleUpdateDelete". There are two tabs: "OracleConn" and another tab that is partially visible. The code is divided into sections A through E.

```

Protected Function OracleConn() As OracleConnection
    Dim cmdString As String = ConfigurationManager.ConnectionStrings("ora_conn").ConnectionString
    Dim conn As New OracleConnection
    conn.ConnectionString = cmdString
    conn.Open()
    If conn.State <> ConnectionState.Open Then
        MsgBox("Database Open is failed")
        conn = Nothing
    End If
    Return conn
End Function

Protected Sub ReportError(ByVal ErrSource As OracleBase)
    ErrSource.OracleOK = False
    MsgBox(ErrSource.OracleError)
End Sub

```

**A** Protected Function **OracleConn()** As **OracleConnection**  
**B** Dim cmdString As String = ConfigurationManager.ConnectionStrings("ora\_conn").ConnectionString  
**B** Dim conn As New **OracleConnection**  
**B** conn.ConnectionString = cmdString  
**B** conn.Open()  
**B** If conn.State <> ConnectionState.Open Then  
**B**     MsgBox("Database Open is failed")  
**B**     conn = Nothing  
**B** End If  
**B** Return conn  
**B** End Function

**C** Protected Sub ReportError(ByVal ErrSource As **OracleBase**)  
**D**     ErrSource.**OracleOK** = False  
**E**     MsgBox(ErrSource.**OracleError**)  
**E** End Sub

Figure 8.161. Modified subroutines OracleConn and ReportError.

Your modified subroutines OracleConn() and ReportError() should match the one shown in Figure 8.161. The modifications are highlighted in bold.

Next, let's develop the stored procedure UpdateCourseSP to perform the course updating functionality.

#### 8.12.4.1 Develop the Stored Procedure **UpdateCourse.SP**

A very detailed discussion of creating and manipulating packages and stored procedures in Oracle database is provided in Section 4.19.7 in Chapter 4. Refer to that section to get more detailed information on creating Oracle stored procedures.

The topic we discuss in this section is how to update and delete data in the database, so 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 discussed in Section 4.19.7 in Chapter 4, different methods can be used to create Oracle stored procedures. In this section, we will use the Object Browser page provided by Oracle Database 10g XE to create our stored procedures.

Open the Oracle Database 10g XE home page by going to Start>All Programs|Oracle Database 10g Express Edition|Go To Database Home Page. Finish the login process by entering the correct user ID and password. Click the Object Browser and select Create|Procedures to open the Create Procedure window. Click the Create button and select the Procedure icon from the list to open this window.

Enter “UpdateCourse.SP” into the Procedure Name box, keep the Include Argument check box checked, and click the Next button to go to the next page.

The next window is used to enter all input parameters. For this stored procedure we need to perform two queries, 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 selected by the user, and the second query is to update a course record that contains six pieces of information related to a current course in the Course table based on the faculty\_id that is obtained from the first query. The seven input parameters are Faculty Name, Course ID, Course Title, Credit, Classroom, Schedule, and Enrollment.

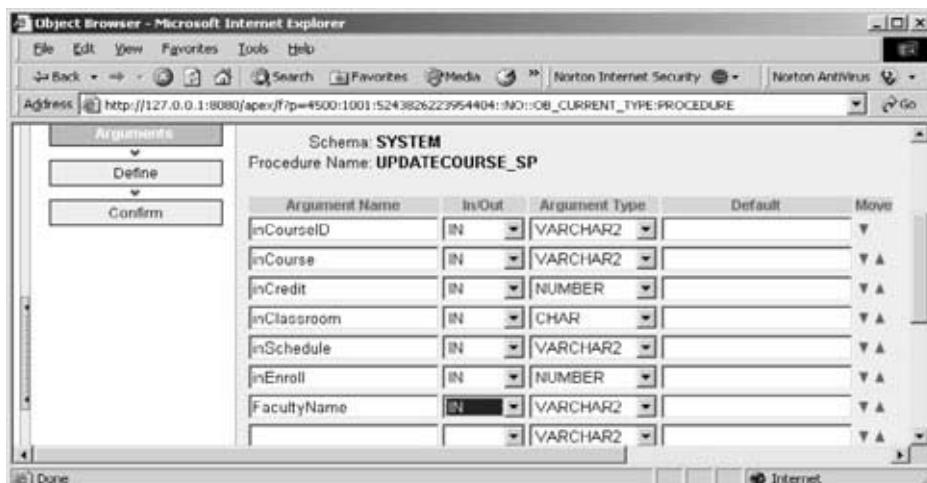


Figure 8.162. The finished argument list.

The first input parameter, FacultyName, is used by the first query, and the following six input parameters are used by the second query.

Enter those input parameters one by one into the argument box. The data type of each input parameter must be identical to the data type of each data column used in the Course table. Refer to Section 2.11.3 in Chapter 2 to get a detailed list of data types used for those data columns in the Course data table.

For the Input/Output 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 8.162.

Click the Next button to go to the procedure-defining page. Enter the code shown in Figure 8.163 into this new procedure as the body of the procedure using the Procedural Language Extension for SQL, or PL/SQL. Then click the Next and Finish buttons to confirm creating this procedure. Your finished stored procedure should match the one shown in Figure 8.164.

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 must be identical to 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 the local variable facultyID declared later.

Two queries are included in this procedure. The first query is used to get the faculty\_id from the Faculty table based on the input parameter FacultyName, and the second query is to update a course record based on six input parameters in the Course table. A semicolon must be used 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 8.164; this code has been

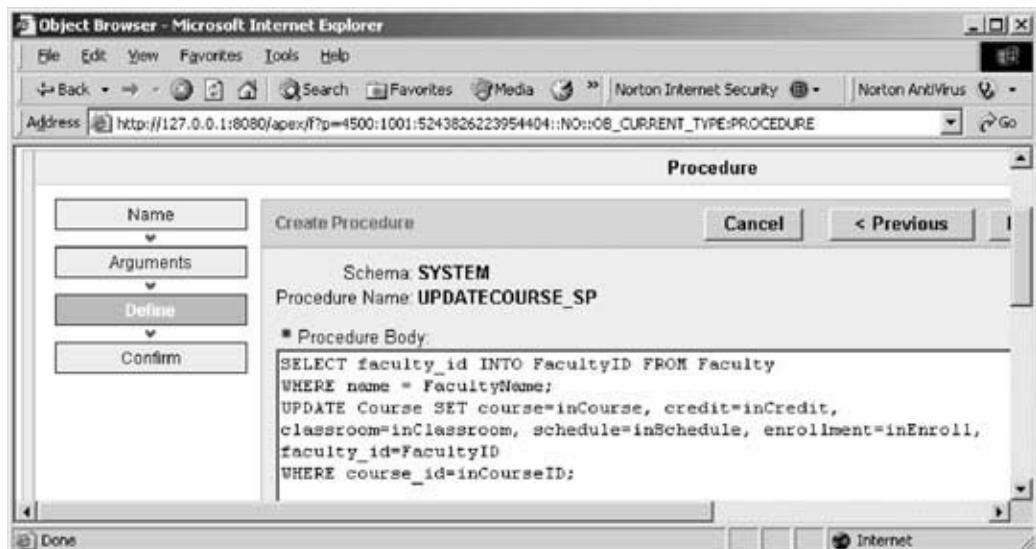


Figure 8.163. The stored procedure body.

highlighted. Click the Edit button to add this local variable. This local variable is used to hold the faculty\_id returned from executing the first query.

Another important issue regarding the input parameters or arguments in the UPDATE command is that the order of those parameters or arguments must be identical to 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 the input parameters

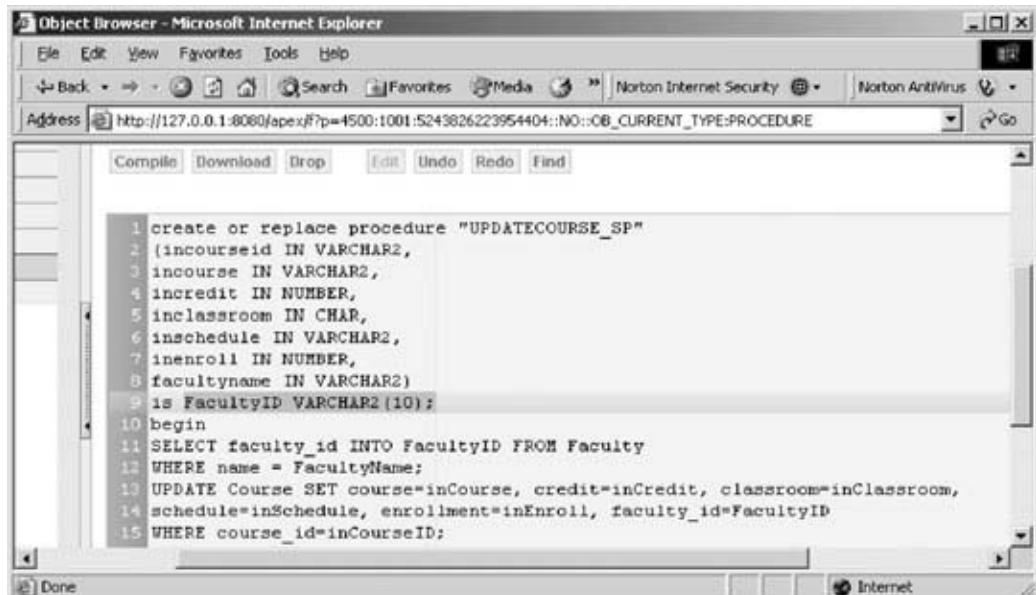


Figure 8.164. The completed stored procedure.

in the UPDATE argument list must be identical to the data columns' order displayed above.

To make sure that this procedure works properly, we need to compile it first. Click the Compile button to compile and check our procedure. A successful compilation message should be displayed if our procedure is a bug-free stored procedure.

Close the Oracle Database 10g Express Edition by clicking the Close button.

### 8.12.5 Modify the Web Method GetSQLCourse and Related Subroutines

The functionality of this Web method is to retrieve all course\_id, including 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 4.18.6 in Chapter 4, 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 this is that there is no faculty name column in the Course table, and each course or course\_id is related only 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 joined-table query is a desired method to complete this functionality.

Open this Web method and perform the modifications shown in Figure 8.165 to this method. All related modifications are 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 GetSQLCourse to GetOracleCourse. Also change the name of the returned instance from SQLBase to OracleBase.
- B. Modify the query string to match the ANSI 89 standard. Recall that we developed a joined-table query string for the SQL Server database using the ANSI 92 standard in Section 4.18.6 in Chapter 4. Since the ANSI 89 standard is still being used in the Oracle database, we need to modify this joined-table query string by using that standard.
- C. Change the prefix 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 OracleType.VarChar.
- G. Change the names of two passed arguments to the subroutine FillCourseReader() from SQLResult to OracleResult, and from sqlReader to oraReader.

WebServiceOracleUpdateDelete		▼	GetOracleCourse		▼
A	<WebMethod()> _				
B	Public Function GetOracleCourse(ByName FacultyName As String) As OracleBase				
	Dim cmdString As String = "SELECT Course.course_id FROM Course, Faculty " + _				
	"WHERE (Course.faculty_id = Faculty.faculty_id) AND (Faculty.name =: fname)"				
C	Dim oraConnection As New OracleConnection				
	Dim OracleResult As New OracleBase				
	Dim oraCommand As New OracleCommand				
	Dim oraReader As OracleDataReader				
D	OracleResult.OracleOK = True				
E	oraConnection = OracleConn()				
	If oraConnection Is Nothing Then				
	OracleResult.OracleError = "Database connection is failed!"				
	ReportError(OracleResult)				
	Return Nothing				
	End If				
F	oraCommand.Connection = oraConnection				
	oraCommand.CommandType = CommandType.Text				
	oraCommand.CommandText = cmdString				
	oraCommand.Parameters.Add("fname", OracleType.VarChar).Value = FacultyName				
G	oraReader = oraCommand.ExecuteReader				
	If oraReader.HasRows = True Then				
	Call FillCourseReader(OracleResult, oraReader)				
H	Else				
	OracleResult.OracleError = "No matched course Found"				
	ReportError(OracleResult)				
	End If				
	If Not oraReader Is Nothing Then oraReader.Close()				
	If Not oraConnection Is Nothing Then oraConnection.Close()				
	oraCommand.Dispose()				
	Return OracleResult				
	End Function				

**Figure 8.165.** The modified Web method GetOracleCourse.

- 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:

- 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.
- Change the method from GetSQLString(0) to GetOracleString(0).

Your modified subroutine FillCourseReader() should match the one shown in Figure 8.166.

### 8.12.6 Modify the Web Method GetSQLCourseDetail and Related Subroutines

The functionality 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.

WebServiceOracleUpdateDelete	▼	FillCourseReader	▼
<b>A</b>		Protected Sub FillCourseReader(ByRef sResult As OracleBase, ByVal sReader As OracleDataReader)	
		Dim pos As Integer	
<b>B</b>		While sReader.Read()	
		sResult.CourseID(pos) = Convert.ToString(sReader.GetOracleString(0))     'the 1st column is course_id	
		pos = pos + 1	
		End While	
		End Sub	

Figure 8.166. The modified subroutine FillCourseReader.

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 we developed in Section 8.10.7.

Open this Web method and perform the modifications shown in Figure 8.167 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 GetSQLCourseDetail 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 the Section 8.10.7. Change this query string from dbo.WebSelectCourseSP to WebSelectCourseSP.SelectCourse. WebSelectCourseSP is a package and SelectCourse is a stored procedure.
- C. Change the prefix from Sql to Oracle for all data classes, and from sql to ora for all data objects used in this method. Also change the name of the returned instance from SQLResult to OracleResult. Change the first member data from SQLOK to OracleOK.
- D. Add two OracleParameter objects, paramCourseID and paramCourseInfo. Because some differences exist between the SQL Server and Oracle databases, we need to use 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. Note an important point for the second parameter, paramCourseInfo. The data type of this parameter is Cursor, and the Direction is Output. Both values are very important to this parameter and must be assigned exactly as we did here. Otherwise a running exception will be encountered when the project runs.
- G. Add two statements to add two OracleParameter objects to the Command object.
- H. Change the names of two passed arguments to the subroutine FillCourseDetail() from SQLResult to OracleResult and from sqlReader to oraReader.

	WebServiceOracleUpdateDelete	▼	GetOracleCourseDetail	▼
--	------------------------------	---	-----------------------	---

```

<WebMethod()> _
Public Function GetOracleCourseDetail(ByVal CourseID As String) As OracleBase
    Dim cmdString As String = "WebSelectCourseSP.SelectCourse"
    Dim oraConnection As New OracleConnection
    Dim OracleResult As New OracleBase
    Dim oraReader As OracleDataReader
    Dim paramCourseID As New OracleParameter
    Dim paramCourseInfo As New OracleParameter

    OracleResult.OracleOK = True
    oraConnection = OracleConn()
    If oraConnection Is Nothing Then
        OracleResult.OracleOK = False
        OracleResult.OracleError = "Database connection is failed"
        ReportError(OracleResult)
        Return Nothing
    End If
    paramCourseID.ParameterName = "CourseID"
    paramCourseID.OracleType = OracleType.VarChar
    paramCourseID.Value = CourseID
    paramCourseInfo.ParameterName = "CourseInfo"
    paramCourseInfo.OracleType = OracleType.Cursor
    paramCourseInfo.Direction = ParameterDirection.Output      'this is very important
    Dim oraCommand = New OracleCommand(cmdString, oraConnection)
    oraCommand.CommandType = CommandType.StoredProcedure
    oraCommand.Parameters.Add(paramCourseID)
    oraCommand.Parameters.Add(paramCourseInfo)
    oraReader = oraCommand.ExecuteReader
    If oraReader.HasRows = True Then
        Call FillCourseDetail(OracleResult, oraReader)
    Else
        OracleResult.OracleError = "No matched course found"
        ReportError(OracleResult)
    End If
    oraReader.Close()
    oraReader = Nothing
    oraConnection.Close()
    oraCommand.Dispose()
    Return OracleResult
End Function

```

Figure 8.167. The modified Web method GetOracleCourseDetail.

- I. Change the name of the returned instance from SQLResult to OracleResult and change the second member data from SQLError to OracleError.

The modifications to the related subroutine FillCourseDetail() are simple, and the only modifications are to change the data type of two passed arguments sResult and sReader. Change the data type of the first argument from SQLBase to OracleBase and change the data type for the second argument from SqlDataReader to OracleDataReader.

### 8.12.7 Modify the Web Method SQLDeleteSP

As discussed in Section 6.1.1 in Chapter 6, 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 in different child tables must be deleted first. In our case, in order to delete a record using the course\_id as the primary key from the Course table (parent table), one must first delete those records using the course\_id as a foreign key from the StudentCourse table (child table). Fortunately we have only one child table related to our parent table in our sample database. Refer to Section 2.5.1 and Figure 2.5 in Chapter 2 to get a clear description of the relationships among different data tables in our sample database.

From this discussion, it can be seen 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, the StudentCourse table, and the second query is used to delete the target record from the parent table, the Course table. To save time and space as well as efficiency, we place these two queries into a stored procedure named WebDeleteCourseSP() that we will develop in the following section. A single input parameter, course\_id, is passed into this stored procedure as the primary key.

Open this Web method and perform the modifications shown in Figure 8.168. The modifications are highlighted in bold. Let's take a closer look at this piece of modified code to see how it works.

```
WebServiceOracleUpdateDelete OracleDeleteSP

<WebMethod()> _
Public Function OracleDeleteSP(ByVal CourseID As String) As OracleBase
    Dim cmdString As String = "WebDeleteCourseSP"
    Dim oraConnection As New OracleConnection
    Dim OracleResult As New OracleBase
    Dim intDelete As Integer

    Oracle.Result.OracleOK = True
    oraConnection = OracleConn()
    If oraConnection Is Nothing Then
        OracleResult.OracleOK = False
        OracleResult.OracleError = "Database connection is failed"
        ReportError(OracleResult)
        Return Nothing
    End If
    Dim oraCommand = New OracleCommand(cmdString, oraConnection)
    oraCommand.CommandType = CommandType.StoredProcedure
    oraCommand.Parameters.Add("CourseID", OracleType.VarChar).Value = CourseID
    intDelete = oraCommand.ExecuteNonQuery()

    If intDelete = 0 Then
        OracleResult.OracleError = "Data deleting is failed"
        ReportError(OracleResult)
    End If
    oraConnection.Close()
    oraCommand.Dispose()
    oraCommand = Nothing
    Return OracleResult
End Function
```

Figure 8.168. The modified Web method OracleDeleteSP.

- 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.
- 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 OracleType.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.

#### **8.12.7.1 Develop the Stored Procedure WebDeleteCourseSP**

The topic discussed in this section is how to update and delete data in the database, so no returned data is needed for these two data actions. Therefore, we only need to create stored procedures, not packages, in the Oracle database to perform the data deleting functionality.

As discussed in Section 4.19.7 in Chapter 4, different methods can be used to create Oracle stored procedures. In this section, we will use the Object Browser page provided by Oracle Database 10g XE to create our stored procedures.

Open the Oracle Database 10g XE home page by going to Start>All Programs|Oracle Database 10g Express Edition|Go To Database Home Page. Finish the login process by entering the correct user ID and password. Click the Object Browser and select Create|Procedures to open the Create Procedure window. Click the Create button and select the Procedure icon from the list to open this window.

Enter “WebDeleteCourseSP” into the Procedure Name box, keep the Include Argument check box checked, and click the Next button to go to the next page.

The next window is used 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, the StudentCourse table, and second, we can delete the target course record from the parent table, the Course table, based on the input course\_id. Only one input parameter, CourseID, is needed for this stored procedure.

Enter this input parameter into the argument box. The data type of this input parameter must be identical to the data type of the data column (course\_id) used in the Course table. Refer to Section 2.11.3 in Chapter 2 to get a detailed list of data types used for those data columns in the Course data table.

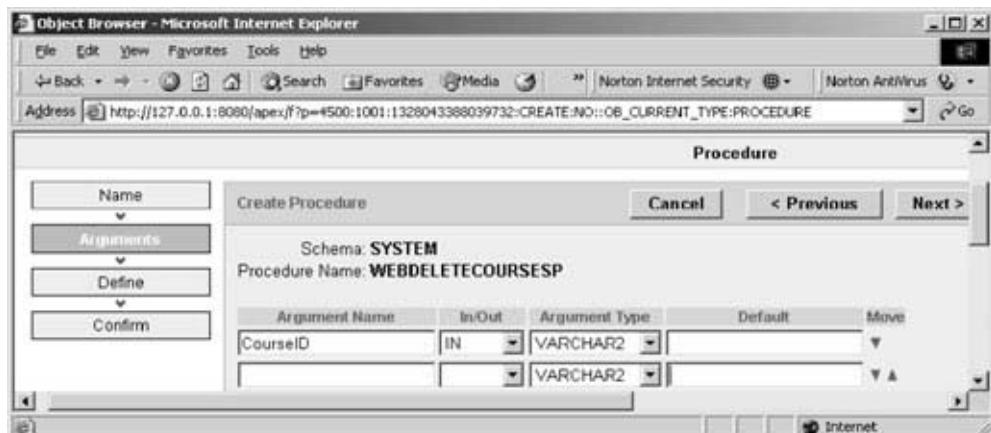


Figure 8.169. The argument list.

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 the one shown in Figure 8.169.

Click the Next button to go to the procedure-defining page. Enter the code shown in Figure 8.170 into this new procedure as the body of the procedure using the Procedural Language Extension for SQL, or PL/SQL. Then click the Next and Finish buttons to confirm creating this procedure. Your finished stored procedure should match the one shown in Figure 8.171.

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 used to delete the target course record from the parent table (Course table). A semicolon must be used after each PL/SQL statement.

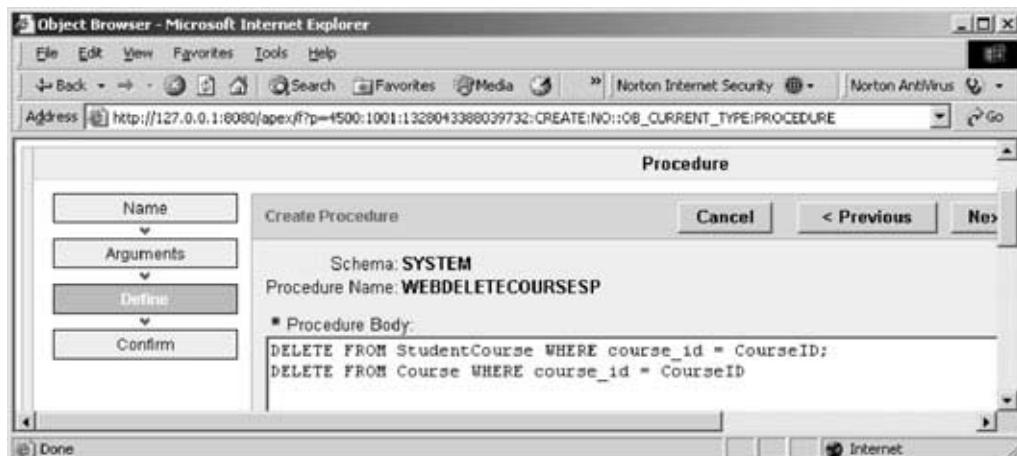


Figure 8.170. The code body of the stored procedure.



Figure 8.171. The completed code body of the stored procedure.

To make sure that this procedure works properly, we need to compile it first. Click the Compile button to compile and check our procedure. A successful compilation message should be displayed if our procedure is a bug-free stored procedure.

Close the Oracle Database 10g Express Edition by clicking the Close button.

At this point, we have finished all modifications to our new Web Service project WebServiceOracleUpdateDelete. Now it is time to run this project to test the data updating and deleting functionalities.

Click the Start Debugging button to run the project. First, test the Web method OracleUpdateSP() to update the course record CSE-575 in 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 it from the built-in Web interface. On the opened parameter-input page, enter CSE-575 into the Value box as the course\_id, and click the Invoke button to retrieve the detailed information for this course. The running result of this method is shown in Figure 8.172.

Keep in mind the detailed information for this course, and let's now try to update this course by running the Web method OracleUpdateSP(). To do that, close the



Figure 8.172. The running result of the Web method GetOracleCourseDetail.

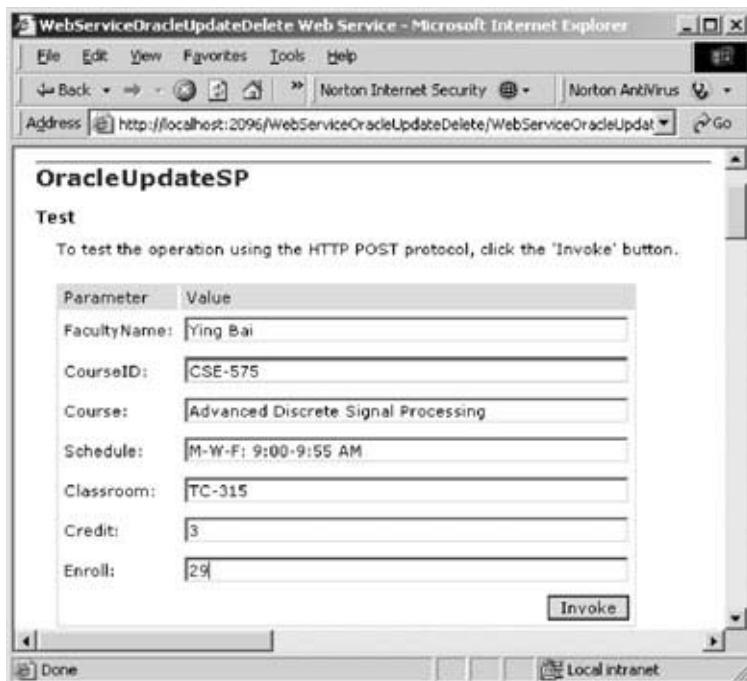


Figure 8.173. The parameter-input page.

current running result page, and click the Back button to return to the initial Web page. Click the Web method OracleUpdateSP to open its parameter-input page. Enter the updated course information shown in Figure 8.173 into the associated Value boxes.

Click the Invoke button to run this method. From the running result window, it can be seen that the member data OracleOK is true, which means that this data updating is successful. Close the running result window.

To confirm this course updating, click the Back button to return to the initial page, and click the Web method GetOracleCourseDetail to try to get back the detailed information for that updated course to validate this data updating. Enter CSE-575 into the CourseID box, and click the Invoke button to run this method. The running result of this method is shown in Figure 8.174.

Compare this running result with the one shown in Figure 8.172. It can be seen that this course has been updated.

Close the current running result window, and click the Back button to return to the initial page. Next, let's test the Web method OracleDeleteSP().

Click this method and enter a course\_id for which you want to delete the associated course, such as CSE-575, into the CourseID Value box, and click the Invoke button to perform this data deletion. It can be found from the running result that the member data OracleOK is true, which means that this data deletion is successful. Close the running result window.

To confirm this data deletion, let's run the Web method GetOracleCourse() to retrieve all courses taught by the selected faculty member. Recall that the course CSE-575 was taught by the faculty member Ying Bai. In the initial Web page, click

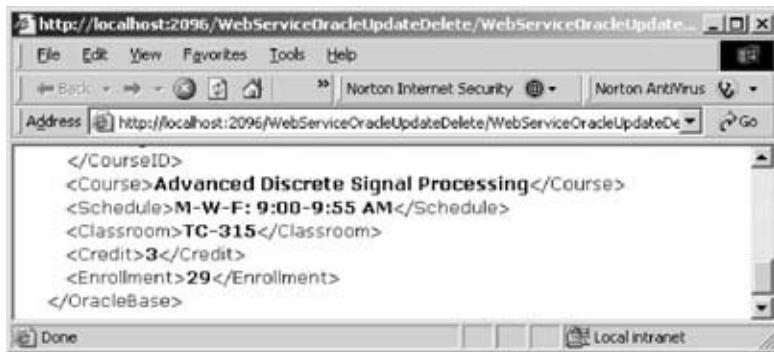


Figure 8.174. The running result of the Web method GetOracleCourseDetail.

this method to run it. Enter the faculty name Ying Bai in the FacultyName box, and click the Invoke button to run this Web method. The running result is shown in Figure 8.175.

From this running result, it can be found that the course CSE-575 has been deleted from the Course table successfully.

At this point, we have finished testing all Web methods developed in this Web Service project. The completed Web Service project WebServiceOracleUpdateDelete can be found at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder DBProjects\Chapter 8.

## 8.13 BUILD WEB SERVICE CLIENTS TO CONSUME THE WEB SERVICE WEBSERVICEORACLEUPDATEDELETE

To consume this Web Service project, one can develop either a Windows-based or a Web-based Web Service client project. There is no significant difference between building a client project to consume a Web Service to access an SQL Server database

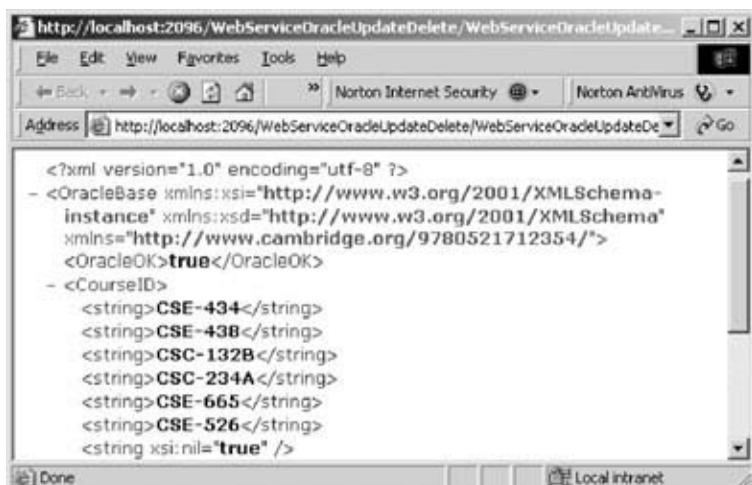


Figure 8.175. The running result of the Web method GetOracleCourse.

and building a client project to consume a Web Service to access an Oracle 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 new developed Web Service WebServiceOracleUpdateDelete.

Follow the modification steps below to complete these changes.

1. Remove the old Web reference from the Windows-based or Web-based client project. You need to first delete the Web reference object, and then you can delete the Web\_Reference folder from the current project.
2. Add a new Web reference using the Add Web Reference dialog box, run the desired Web Service project, copy the URL from that running Web Service project, and paste it to the URL box in the Add Web Reference dialog box in the client project.
3. Change the Web reference name for all data components used in the client project.
4. Remove the namespace for 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 at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**.

## 8.14 CHAPTER SUMMARY

A detailed discussion and analysis of the structure and components of Web Services was provided in this chapter. Unlike 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, 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 ASP.NET Web applications and ASP.NET Web Services is that the latter do not provide any graphical user interfaces (GUIs) and users need to create those GUIs themselves to access 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 the following:

- 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 were also developed in this chapter. Both Windows-based and Web-based Web Service client projects were discussed and built for each kind of Web Service listed above. A total of eighteen projects, including the Web Service projects and the associated Web Service client projects, were developed in this chapter. All projects were debugged and tested and can be run in any Windows-compatible operating systems such as Windows 95, 98, 2000, XP, and Vista.

## **8.15 HOMEWORK**

### **I. True/False Selections**

- \_\_\_\_\_ 1. Web Services can be considered 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 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 code related to building a Web Service are located in this page and major coding development is 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. 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. XML is used to \_\_\_\_\_ the data to be transferred between applications.
- a. Tag
  - b. Rebuild
  - c. Receive
  - d. Interpret
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 to \_\_\_\_\_ 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 the 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. Graphical 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, a 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 graphical 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 an ASP.NET Web Service?
  - b. What are main components of the ASP.NET Web Service?
  - c. How is an ASP.NET Web Service 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 code for that method to perform the data query for the Student via our sample SQL Server database CSE\_DEPT (the project file is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**).
5. Develop a Windows-based Web Service client project WinClientSQLStudent to consume the Web Service developed in question 4, to consume the new Web method GetSQLStudent().

**Hints:** Refer to the project WinClientSQLSelect that is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**. You can add some controls to the main form window and code 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, to consume the new Web method GetSQLStudent().

**Hints:** Refer to the project WebClientSQLSelect that is located at [www.cambridge.org/9780521712354](http://www.cambridge.org/9780521712354) in the folder **DBProjects\Chapter 8**. You can add some controls to the main page window and code to the associated event procedures in that project to complete this job.



# Index

- .NET Framework, 4, 488
- .NET Framework collection class, 308
- .NET Framework Data Provider, 92, 93
- .NET Framework model, 489
  - advantages of using, 488, 489
  - supports user interfaces, 488
- .NET SDK, 630
- Accept Condition, 122
- AcceptButton property, 146, 147, 150
- AccessSelectRTOObject, 191, 192, 193, 195, 197, 199, 201, 203, 205, 207, 209, 211, 213, 215, 217, 219, 221, 223, 225
- Active Server Page.NET, 487
- Active Service Method file, 585
- ActiveX Data Object (ADO), 87
- Add Column, 66
- Add Connection dialog, 139, 152, 153, 249, 250
- Add Existing Item, 229, 230, 545, 624, 669, 714
- Add New Data Source, 137, 151
- Add New Item dialog, 155, 306, 356, 387, 514, 534, 596
- Add New Stored Procedure, 250, 252, 258, 388, 465, 561, 607, 648, 691, 693
- Add Procedure dialog, 258, 388
- Add Query, 168, 314, 418, 419
- Add Reference, 268, 380, 449, 526, 614, 625, 654, 670, 704, 715, 726, 741, 759
- Add Table dialog, 168, 182
- Add Web Reference, 613, 614, 624, 653, 654, 670, 703, 714, 739, 757, 775
- Add Web Reference dialog, 613, 614, 624, 653, 654, 670, 703, 714, 739, 757, 775
- Add Windows Form, 147, 148, 149, 155, 259, 306, 331, 356, 387
- Add() method, 101, 103, 196, 201, 213, 214, 233, 255, 290, 309, 311, 313, 334, 364, 366, 500, 504, 518, 521, 538, 570, 617, 640, 658, 664, 677, 734
- Address bar, 614, 654, 703
- AddWithValue() method, 102
  - and protocols for, 103, 104
- ADO.NET, 87, 301
  - applications and components, 88
  - architecture of, 3, 89, 90
  - and classes in, 191
  - and components of, 190
  - DataSet component of, 113
  - embedded data tables in DataSet, 89
  - nonembedded data tables in DataSet, 90
- classes in, 88, 190
- and components, 5
- development of, 124
- introduction to, 3, 88
- major components, 88, 91
  - Data Provider/data driver, 91
- overview of, 88, 89
- ADO.NET Common Language Runtime, 87
- ADO.NET Framework, 88
- AllCells, 155
- ALTER command, 79, 252, 281
- alternate keys, 19
- ANSI 89 standard, 241, 531, 745, 765
- ANSI 92 standard, 241, 531, 745, 765
- Application state
  - for global connection object storing, 521
  - structure of, 487
  - used for, 509
- Application User Interface, 145
- Application.Exit(), 199
- AS, 248, 281, 285
- ASMX page, 585
- ASP.NET, 489
- ASP.NET runtime, 592, 594, 599, 605, 609, 613
- ASP.NET technology, 490

- ASP.NET Web application, 6, 487, 489, 491, 494, 498, 500, 511, 578, 623, 669
  - core components of, 490
    - caching, 491
    - code-behind pages, 490
    - configuration files, 490
    - database connectivity, 490
    - Global.asax file, 490
    - Web forms, 490
    - Web services, 490
    - XML Web service links, 490
  - creation of, 494
    - data actions in Oracle databases
      - modifications for data updating, 567
    - development and building process of, 4
    - execution, events involved in, 492
    - and execution model sequential events, 491, 492
    - for data insertion into Oracle databases, 544, 545, 547, 549
      - code modification for Faculty page, 548, 549, 550
    - Insert Web page creation and coding for, 545, 546, 547
  - for data selection from SQL Server databases, 494, 495, 497, 499, 501, 503, 505, 507, 509, 511, 513, 515, 517, 519, 521, 523
  - for data updating and deletion in Oracle databases, 567
    - modification for data updating, 568
    - stored procedure for deletion, 570, 571, 572, 573, 574, 575, 577
  - for data updating and deletion in SQL Server databases, 551, 552
    - application user interfaces, 552
    - coding for delete button event, 557, 558, 561, 564
    - coding modification for faculty page, 552
    - coding for update button event, 553, 554, 555, 557
  - file structure, 491
    - folders and file structure, 491
  - implementations of, 4, 5
  - requirements to test and run Web project, 493
    - structure of, 490
  - ASP.NET Web Form, 489
  - ASP.NET Web service, 6, 584
    - access to different databases
      - listed differences, 723, 724
    - to access SQL Server database, 588
    - for data insertion in databases, 740
      - differences in, 740
    - for data insertion in SQL server database, 633, 635, 637
      - code-behind page development, 635, 636, 637, 638, 640, 641, 643, 646, 647, 648, 649, 651, 652
    - development procedure used, 635
      - modification in project, 633, 634
    - implementations of, 4
  - procedures involved in building of, 587
  - ASP.NET Web service, in Oracle database
    - data insertion, 740, 741, 743, 745, 747, 749, 751, 753, 755
      - modify connection string, 741
      - modify namespace directories, 741, 742
      - modify Web method GetSQLInsert, 744, 745, 746
    - modify Web method GetSQLInsertCourse, 748, 749, 750, 752, 753, 755
    - modify Web method SetSQLInsertSP, 742
    - modify Web method SQLInsertDataSet, 746, 748
  - data insertion, WebServiceOracleInsert, 740, 741
    - to update and delete data, 758, 759, 761, 763, 765, 767, 769, 771, 773
  - WebServiceOracleUpdateDelete project, 758, 759, 774
    - develop stored procedure UpdateCourse\_SP, 762, 763, 765
    - modify connection string, 759
    - modify namespace directories, 759, 760
    - modify Web method GetSQLCourse, 765, 766
    - modify Web method GetSQLCourseDetail, 766, 767
    - modify Web method SQLDeleteSP, 768, 769, 770
    - modify Web method SQLUpdateSP, 760, 761
      - stored procedure for, 770, 771, 773, 774, 775
  - ASP.NET Web service, for SQL Server database, 691
    - develop stored procedure
      - WebDeleteCourseSP, 693, 694, 695, 696, 698, 699, 701, 702
    - develop stored procedure
      - WebUpdateCourseSP, 691, 692, 693
    - to update and delete data, 682, 683
      - Web methods modifications, 683
    - Web methods modifications, 684
      - add Web method SQLDeleteSP, 689, 690, 691
    - GetSQLInsert to GetSQLCourse, 686, 687
      - GetSQLInsertCourse to
        - GetSQLCourseDetail, 687, 689
    - SetSQLInsertSP to SQLUpdateSP, 684, 685
  - ASP.NET Web Services project
    - to access SQL Server database, 588
      - base class to handle error checking, 595, 596
      - code development for Web methods, 598, 599, 600, 601, 602, 603, 604, 607
      - files and items created in, 588, 589, 590
      - modifications in default project, 594, 595
      - and project run, 590, 592, 593
      - stored procedure for data query, 607, 608, 609, 611
    - use of dataset as returning object, 609, 610, 611

Web service class and addition of Web methods in, 597, 598  
 Web service class creation, 596, 597  
 Web service deployment on servers, 630, 631, 632, 633  
 Web-based Web service client projects, 623, 624, 625, 626, 627, 628, 629, 668, 670  
 Windows-based Web service client projects, 612, 613, 614, 615, 616, 617, 618, 620, 621, 622, 653, 654, 656  
 ASP+, 585  
 Assembly Information, 305, 331, 355, 416, 431, 436, 448, 456, 463, 469, 703  
 AssemblyInfo.vb file, 492  
 attributes, 14, 18, 19, 23, 29, 492, 585, 589, 640  
 AutoPostBack property, 487, 509, 515, 516, 518, 519, 671, 675  
 AutoSizeColumnsMode, 155  
 AutoSizeMode, 155  
 base class, 490  
     to handle error checking, 595, 596  
 BEGIN, 34, 281, 285  
 bin folder, 491  
 bind a ListBox, 184  
 BindingNavigator, 133, 136, 157, 189  
 BindingSource, 130, 133, 135, 136, 157, 160, 161, 162, 163, 164, 189, 304, 325, 356, 387, 415  
     and functionalities of, 135, 136  
 BindingSource object, 136, 163  
 Bitmap indexes, 35  
 BorderStyle property, 150  
 B-tree (binary-tree), 32  
 B-tree indexes, 35  
 BuildCommand(), 254  
 built-in Web interface, 600, 605, 607, 611, 612, 614, 638, 639, 642, 646, 652, 654, 695, 697, 698, 704, 737, 738, 739, 754, 756, 772  
 C# Web services, 585  
 Caching, 491  
     in ADO.NET, 89  
 candidate key, 19  
 cardinality, 18, 35  
 cascade deleting, 557  
 Cascade option, 559, 560  
 CenterScreen, 146, 147, 150, 307, 332, 356, 387, 654  
 Change Data Source dialog, 139, 152  
 child class, 590, 596, 597, 603, 605, 606, 619, 626, 634, 635. *See also* Web service class  
 child stored procedure, 263, 264, 265  
 Class Libraries, 488  
 Clear method, 115  
 ClearBeforeFill property, 166, 338  
 Client Server databases, 28  
     Microsoft SQL Server, 28  
     Oracle as, 28  
 Close() method, 92, 96, 97, 167, 177, 198, 199, 359, 362, 498, 504, 617, 629, 658  
 clustered indexes, 32  
 clustered table, 33  
 code window, 192, 269, 309, 312, 313, 657  
 code-behind page, 490, 495, 496, 585, 586, 589, 593, 594, 595, 599, 600, 602, 608, 610, 635, 640, 649, 683, 689, 725, 726, 741, 742, 758, 759  
 code illustration  
     for data query by using Fill() method, 166  
 Columns collection, 113, 120, 122  
 ComboBox control, 171, 172, 173, 176, 181  
 ComboMethod, 199, 201  
 Command class, 99, 105  
     and categories of, 98  
     constructor of, 104, 105  
     ExecuteNonQuery method, 106, 107  
     ExecuteReader method, 105, 106  
     ExecuteScalar method, 106  
     and functionalities, 105  
     properties of, 99  
     protocols of constructor of, 104  
 Command Console, 488  
 command execution, in ADO.NET, 89  
 Command object, 98, 99, 102, 103, 104, 105, 118, 124, 125, 191, 196, 197, 201, 232, 233, 249, 255, 261, 271, 288, 302, 303, 354, 393, 394, 405, 435, 441, 444, 445, 460, 465, 471, 499, 509, 510, 538, 554, 572, 601, 602, 608, 637, 640, 644, 646, 650, 684, 688, 690, 734, 749, 767  
 CommandText, 255  
 CommandText property, 99, 117, 118, 249, 394, 460, 564, 644, 645, 646  
 CommandType property, 99, 117, 249, 261, 465, 471, 480, 565, 608, 637, 650  
 CommandType.StoredProcedure, 255, 572, 608, 637  
 Common Language Runtime, 488  
 compile directive, 589, 590  
 Component Object Model (COM), 87  
 composite key, 19, 25  
 Configuration files, 490  
 ConfigurationManager class, 600  
 Connection class, 94, 96, 97, 373, 381  
     and connection string, 94  
     Close() method, 97  
     and databases, 94  
     Dispose() method of, 97, 98  
     Open() method in, 96  
 Connection instances and databases, 96  
 Connection object, 94  
 Connection property, 99, 117  
 connection string, 373, 527, 598, 599, 741, 759  
     data connection instance and codes for, 95  
     parameters in, 95  
     parts of, 226

ConnectionState.Open, 375, 377, 383, 384, 600  
 connectionStrings section, 600  
 ConnectionStringSettingsCollection, 600  
 connectivity, 19, 87, 91  
     relationship types among entities, 19  
     many-to-many, 21, 23  
     one-to-many, 21  
     one-to-one, 21  
 Constraint Name, 74, 79, 80, 81, 82  
 constraint property, 575  
 Constraint Type, 74, 79, 80, 81, 82  
 ControlToValidate, 501, 540  
 Count property, 166  
 Create Package window, 283, 729, 730, 749  
 CREATE PROCEDURE, 250  
 Create query in Design view, 457  
 CREATE OR REPLACE, 281  
 Creation Wizard, 631  
 cursor, 174, 282, 285, 287, 345, 373, 420, 433, 495,  
     731, 732, 751, 752  
 CURSOR\_TYPE, 285, 731, 750  
 custom stored procedures, 247

Data Access Objects (DAO), 87, 124  
 data accessing methods and operations, 91  
     history of, 87  
 data actions, 551, 552  
 data actions, in Oracle databases  
     modifications for data updating  
         add coding to Update button and  
             UpdateParameters procedures, 568, 569,  
             570  
         Select button's click event procedure  
             modifications, 567, 568  
     stored procedures development for data  
         deletion, 570, 571  
         coding for Delete button's event, 572, 573,  
             574  
         constraint property on Delete cascade, 575,  
             576, 577  
         data deletion actions validation, 574, 575  
         existing record deletion, 571, 572  
 data binding, 136, 162, 173, 174, 185, 188, 294, 323,  
     324, 325, 340, 341, 417, 418  
     to associated controls, 181, 183  
         steps used in, 181, 182, 183, 184, 185  
     to controls in LogIn form, 162, 163, 164, 165  
     to DataGridView and methods, 294  
 data components  
     relationship between, 135  
     in toolbox window, 133  
 data connections, 249, 258, 388  
 data consistency, 12  
 data deleting query, 444  
 data encryption, 12  
 data files, 30, 33, 607  
 data independence, 12  
 data insertion methods, 3  
 data insertion, using stored procedures  
     into Oracle database  
         coding of data, 401, 402, 403, 404, 405  
         development of stored procedures, 398, 399,  
             400  
     into SQL Server database  
         coding of data, 390, 392, 393, 394, 395, 396  
         development of stored procedures, 388, 389  
         insertion of user interfaces, 386, 387  
 data insertion, with Visual Basic.NET  
     into Microsoft Access, 304  
         coding of data, 312, 313  
         coding using TableAdapter.Insert Method,  
             315, 316, 317  
         coding using TableAdapter.Update Method,  
             318, 320, 321, 322  
         creation of InsertWizardProject.vbproj, 305  
         development of graphical user interfaces,  
             305, 306, 307, 308  
         development of Insert Query, 314  
         validation of data, 308, 309, 310, 311, 312,  
             322, 323, 324, 325, 326, 327, 328, 329  
     into Oracle database, 352  
     with runtime object method, 353, 354  
         development of user interfaces, 369, 370, 371,  
             372  
         into Microsoft Access database, 372, 373, 374,  
             375, 376, 377, 378, 379  
         into Oracle database, 379, 380, 381, 382, 384,  
             385  
     into SQL Server database, 355, 356, 357, 358,  
         359, 360, 361, 363, 364, 365, 366, 367, 368,  
         369

into SQL Server Database  
     coding using TableAdapter.Insert method,  
         337, 338, 339, 340  
     coding using TableAdapter.Update method,  
         342, 343, 344  
     configuration of Table Adapters, 336, 337  
     data-binding procedure, 341  
     development of user interfaces, 331, 332  
     modification of data, 331  
     using stored procedures, 349, 350, 352  
     validation of data, 344, 345, 346, 347, 348  
     validation and initialization of data, 332, 333,  
         334, 335

into Visual Studio.NET, 302, 303  
     using TableAdapter.Insert method, 303  
     using TableAdapter.Update method, 304

Data model, 14  
 data normalization  
     and steps in (normal forms), 23  
 data processing techniques, 87  
 Data Provider, 3  
     and components of, 190  
     and data types, 100  
     functionalities of, 91  
     popular versions of, 91, 92

- ODBC Data Provider, 92
- OLEDB Data Provider, 93
- Oracle Data Provider, 94
- SQL Server Data Provider, 93
- subclasses in
  - connection class, 94, 95, 96
- subcomponents of, 124
- versions of, 124
- Data Provider class, 89
- Data Provider-dependent components, 89
- Data Provider-dependent objects, 105, 236
- data query, 90, 91
  - applications, 129
  - code development for, 165
  - method, 98
  - stored procedure to perform, 607
    - develop WebSelectFacultySP, 607
  - technique, 125
  - using runtime objects, 212, 213, 216
- data redundancy minimization, 12
- data selection queries, 3
- data sharing, 12
- Data Sheet view, 37
- Data Source Configuration Wizard, 137, 138, 139, 140, 151, 153
  - steps included in, 139, 140, 141, 142
  - supports data selection, 138, 139
- Data Source parameter, 95, 227, 228, 231, 268
- Data Source wizard, 129, 130, 145
- data sources, 139
- Data Sources window, 114, 134, 138, 141, 142, 144, 151, 155, 168, 174, 183, 302, 304, 314, 336, 344, 350, 412, 415, 418, 432
  - features of, 137
    - adding new data source, 137, 138
    - Data Source Configuration Wizard, 138, 139, 140, 141
    - DataSet Designer, 143, 144
- Data tool, 66, 68
- Data Type, 36, 48
- data updating and deleting methods, 4
- data updating process, 556
- data updating and deletion, against database
  - using runtime objects
    - for Oracle database, 447, 448, 449, 450–452, 453, 454
    - for SQL Server database, 436, 437, 438, 439, 441, 442, 443, 444, 445, 446
- using stored procedures
  - for Microsoft Access database, 455, 456, 457, 458, 459, 460, 461
  - for Oracle database, 469, 470
  - for SQL Server database, 462, 463, 464, 465, 466, 467, 468, 469
- data updating and deletion, in SQL Server databases
  - coding for delete button event
    - cascade deleting option usage, 559, 560
  - codes to call stored procedure for deletion, 564, 565, 566
  - data deletion sequences, 558, 559
  - relation between tables, 557, 558
  - stored procedure for deletion, 561, 562, 563, 564
- data updating and deletion, using
  - VisualBasic.NET
    - against databases
      - from related tables, 413
      - TableAdapter.DBDirect methods, 414
      - TableAdapter.Update methods, 414, 415
    - for Microsoft Access database
      - creation of application user interfaces, 416, 417
      - creation of new folder with Insert Wizard project, 416
      - creation of update/delete queries, 418, 419, 420
      - development of deletion codes using
        - TableAdapter DBDirect method, 425, 426
      - development of deletion codes using
        - TableAdapter Update method, 423, 424, 426
      - development of updating codes using
        - TableAdapter DBDirect method, 420, 421, 423
      - development of updating codes using
        - TableAdapter Update method, 423
      - validation of data, 418, 427, 428, 429
    - for Oracle database using
      - OracleUpdateDeleteWizard, 433
    - for SQL Server database using
      - SQLUpdateDeleteWizard, 430, 431, 432, 433
- data validation process, 348, 379, 397, 405, 544, 551
- DataAdapter, 89
  - versions of, 107
- DataAdapter class, 109, 611
  - as link between DataSet and data source, 107, 108
  - constructors of, 108
  - events of, 109, 110
  - methods of, 108, 109
  - properties of, 108
- DataAdapter object, 89
- DataAdapter.Update(), 354, 434
- database administrators (DBAs), 12
- database alias, 228, 267
- database connection coding, 231
- database connectivity, 490
- database design, concepts of, 10
- database development process
  - steps and phases involved in, 13, 14
- database files, in Microsoft Access, 29
- database management programs, 10

- Database Management System (DBMS), 7, 11, 12, 84, 138  
 for installation and development of applications, 7
- Database parameter, 95, 227, 228  
 database processing, 18  
 database programming, 3  
 fundamental and advanced, 2  
 with Visual Basic.NET 2005, 2  
 database programs, 1, 2, 11  
 database structures and components, fundamentals of, 5  
 database technology  
 impact on daily life, 10  
 database versions, 3  
 databases, 11  
 access and manipulation of data in, 124  
 components of popular databases, 26, 28, 33  
 Microsoft Access, 28  
 Microsoft SQL Server database, 30, 31, 32  
 Oracle database, 33, 34, 35, 36  
 and database programs, 11, 12, 13  
 and developmental process of, 13, 14  
 and ER notations used in, 23  
 introduction to, 10  
 keys identification and, 18, 19  
 normalization process in, 23, 24, 25  
 and relationship defined as, 19, 21, 22, 23  
 and sample for concepts illustration, 14  
 ER model, 17, 18  
 relational data model, 14  
 structure and components of, 84  
 types of, 3, 4, 28
- DataBindings property, 162, 163, 173, 174, 184, 325
- DataTable, 118, 119, 120, 122, 203, 224
- DataTable object, 120, 122
- DataTable objects, 113, 289
- DataTableCollection, 119, 212, 289
- DataConnection, 134
- DataConnector, 137
- data-driven application development  
 by using ADO.NET, 3  
 by using classes and methods provided by ADO.NET, 191  
 by using runtime object, 3, 191  
 with design tools and wizards, 5
- data-driven project, 226
- DataGridView, 130, 133, 134, 135, 155, 156, 157, 159  
 data query and display by using, 155, 157  
 entire table view, 155, 156, 157  
 viewing each record, 157, 159  
 properties of, 135  
 relationship with other components, 135
- DataReader  
 creation of, 110  
 to query data and coding used for, 196, 197
- DataReader class, 105, 110, 111, 112, 113  
 exceptions of, 113
- DataReader method, 199, 212, 243  
 to query data, 198, 233
- DataReader object, 89, 90, 105, 110, 112, 125, 191, 196, 205, 233, 248, 260, 272, 499, 500, 510, 513, 520, 523, 524, 604, 641
- DataReader query method, 219
- DataRelation, 113
- DataRelation object, 114, 134
- DataRow, 113, 118, 119, 120, 122, 123, 203, 210, 224, 304, 318, 319, 320, 342, 343, 344, 414, 424, 426, 427, 620, 664, 678
- DataRow object, 116, 119, 123, 212, 223, 289, 318, 319, 320, 342, 343, 344, 424
- DataRowCollection, 289, 413, 551
- DataRowState, 120
- DataSet class, 88, 89, 109, 113, 115, 116, 134, 191, 626  
 constructors of, 115  
 and database, 114  
 events of, 117  
 functionalities of, 114  
 methods of, 116
- DataSet component, 113, 114  
 constructor, 115  
 and events, 115, 117, 118  
 and methods used in, 115  
 properties, 115
- DataSet constructor, 115
- DataSet Designer, 114, 142, 143, 144, 157, 159, 168, 174, 182, 183, 187, 304, 336, 344, 415  
 and data operations used in, 143, 144  
 defined as, 143  
 for editing structure of DataSet, 159  
 function to add missed table, 144
- DataSet events, 115, 117, 118
- DataSet filling, 89
- DataSet generation file, 433
- DataSet method, 115, 617, 622, 660, 661, 662, 663, 665, 667, 674, 676, 681, 708, 713, 719
- DataSet object, 113, 114, 115, 116, 117, 118, 119, 120, 134, 141, 319, 342, 616, 617, 618, 657, 665, 674, 676, 678  
 defined as, 134
- DataSet properties, 115
- DataSet use, as returning object for Web method, 609, 610, 612
- DataSet-DataAdapter method, 191, 354
- DataSetName, 115
- DataSet-TableAdapter method  
 data query by using  
 and coding used, 194, 195, 196
- DataSource, 136, 138, 184
- DataTable class, 88, 89, 113, 118, 119, 120, 122, 191, 200, 203, 620  
 events of, 122  
 methods involved in, 121

popular constructors of, 119  
 popular properties of, 121  
**DataTable** component, 118, 119  
 constructors in, 119, 120  
 events in, 122, 123  
 methods used in, 120, 122  
 properties of, 120  
**DataTable** constructor, 119  
**DataTable** events, 122  
**DataTable** object, 89, 90, 118, 119, 120, 125, 203, 213, 290  
 and relation with **DataSet**, 113  
 **DataView**, 118, 119  
**dBase III**, 10  
**DBDirect** method, 302, 303, 313, 314, 315, 414, 415, 418, 423, 425, 426, 431  
**DBMS**. *See Database Management System*  
**DbType**, 100  
**DbType** property, 100  
**DECLARE**, 248  
**DELETE**, 89, 90, 280, 358, 419, 448, 451, 478, 481  
**Delete Rule** item, 560  
**DeleteCommand**, 4, 89, 98, 107, 124, 191, 303, 304, 354, 411, 435, 481  
**Design view**, 36, 37, 39, 457  
**DialogResult**, 426, 444  
**Direction** property, 287  
disconnected working mode, 301  
Discovery Map file, 613, 623, 669  
DisplayMember property, 184  
Dispose() method, 97, 98, 108, 110, 115, 196, 198  
Disposed event, 115  
Domain indexes, 35  
Drop Column, 66  
DropDownList control, 625, 671  
DropDownStyle property, 199  
dynamic parameters, 102, 137, 160, 166, 197, 232, 234, 271, 272, 450–452, 527, 724, 734, 740, 742, 758, 760  
dynamic query, 160  
Edit **DataSet** with Designer, 143, 157, 159, 168, 174, 183, 314, 336, 344, 418, 419  
Edit Relationships dialog, 41, 46  
END, 281, 282, 285  
EndEdit(), 424  
endpoint, 583, 584  
Enforce Referential Integrity, 41  
Enterprise edition, in Oracle, 33  
Enterprise Manager Wizard, 247  
entity, 18, 19  
 attributes of, 29  
entity integrity rule, 19  
entity-relationship diagrams (ERDs), 13, 18  
 components of, 18  
entity-relationship (ER) model, 11, 17, 18  
 developed by Peter Chen in 1976, 17  
ER diagram, 13, 18  
ER model, 11, 17, 18  
ER notation, 23, 25  
error ORA-02298, 80  
ErrorMessage, 501, 540  
Execute, 89, 389, 466, 598, 648, 692, 694  
ExecuteNonQuery() method, 3, 4, 90, 106, 107, 118, 124, 191, 248, 302, 303, 330, 354, 356, 363, 387, 393, 394, 405, 411, 425, 435, 441, 445, 460, 482, 538, 554, 565, 566, 569, 572, 573, 637, 638, 644, 646, 684, 685, 690, 691  
ExecuteReader, 90, 92, 118, 124, 191, 197, 244, 261, 272, 500, 520, 523, 603, 641, 650, 687, 688  
ExecuteReader method, 105, 106, 110, 112, 191, 212, 235, 510  
 data query method, 105, 106  
ExecuteScalar method, 106, 644  
Execution methods, 91  
Exit Sub, 166, 439  
Exit() method, 173  
extended stored procedures, 247  
Faculty Web form, 512, 513, 539, 552, 564, 568, 572  
FacultyLabel object array, 236  
Field Name, 36  
Field Properties, 38  
Field Size, 38  
FieldCount property, 112, 217, 524  
file processing system, 11, 12  
 advantages of, 11  
File Server database, 28  
 Microsoft Access as, 28  
File System, 588, 589, 590, 614, 654  
Fill() method, 108, 109, 110, 114, 117, 118, 125, 134, 137, 141, 157, 161, 163, 166, 183, 184, 196, 197, 210, 243, 244, 255, 288, 358, 611, 646  
 and arguments, 166  
 code illustration for data query by using, 166  
FillBy() method, 174, 175  
FillByUserNamePassWord() method, 165  
FillFacultyReader(), 510  
 and functionality of, 512  
FillSchema method, 108  
FindBy(), 423, 426  
FindName function, 223  
first normal form (1NF), 24, 25  
 faculty table in, 24  
FixedSingle, 150  
foreign key, 19, 57, 59, 60, 61, 74, 79, 80, 81, 82, 560, 575  
Foreign Key Relationships dialog, 57, 59, 60, 61  
Form\_Load() event procedure  
 coding for, 235  
FromFile() method, 328  
FrontPage Server Extensions 2002, 493  
FrontPage Server Extensions 2000, 7, 493, 578  
 and the installation process of, 4

- Full Table view, 155
- GetData() methods, 143, 157, 159, 161, 303
- GetOracleString(), 385, 452, 532
- GetSQLInsertCourse method, 648
- GetSqlString(), 385, 452, 532
- GetString, 214, 291, 521, 604
- Global.asax file, 490
- Group By, 168
- HasChanges, 116
- HasErrors, 116
- HasRows property, 112, 197, 203, 212, 235, 603, 604, 641, 651, 687, 688
- Hide() method, 166, 392
- HTTP Post, 592
- HTTP protocol, 592
- HttpApplication, 490
- HttpApplicationState class, 498
- HttpContext class, 498
- HttpModules, 490
- Human Genome Project, 10
- hypertext transfer protocol (HTTP), 488
- IBM DB2, 10
- IIS Manager, 631
- IIS virtual directory, 630
- image properties, 254
- image storage, in database, 178  
  steps in configuration, 178, 179
- Image.FromFile(), 622
- ImageUrl property, 512
- implementation phase, 14
- ImportRow, 120
- Imports System.Data, 219, 449, 726, 742, 759
- IN, 282, 285, 399, 763, 771
- Index organized table, 33
- indexes, 32, 35, 283  
  function based, 35
- IndexOutOfRangeException, 113
- initialization parameter files, 35  
  types of, 35
- Initialized event, 115
- in-line SQL statement, 159
- inner join method, 241
- INSERT, 89, 90, 280, 314, 336, 337, 350, 358, 363, 372, 385, 448, 452, 645
- Insert page, 534, 535, 539, 540, 541, 542, 543, 544, 545, 548, 549, 550
- Insert Web form, 536, 537, 540, 541, 546
- Insert Web page, creation and coding for  
  Imports commands, modifications to, 546  
  Page\_Load Event Procedure, modifications to, 546  
  subroutine and procedure coding, modifications to, 546
- InsertCommand, 89, 98, 107, 124, 191, 304, 354, 435, 646
- InstanceName, 599
- integrated database approach  
  advantages over file processing, 12  
    data consistency and integrity, 12  
    data independence, 12  
    data redundancy minimization, 12  
    data security, 12  
    data sharing, 12  
    standards enforcement, 12
- integrated databases, 11, 12, 13
- Integrated Security parameter, 227
- Internet Information Services (IIS), 493, 586, 630  
  installation steps for, 493
- InvalidOperationException, 97
- InvalidOperationExceptionError, 194
- Invoke button, 592, 595, 600, 605, 609, 611, 639, 642, 647, 652, 696, 697, 698, 699, 736, 737, 739, 753, 754, 755, 772, 773, 774
- IS operator, 285, 731, 751
- IsClosed property, 111
- IsDBNull, 111
- IsInitialized, 115
- IsPostBack property, 509
- Item(Int32) property, 112
- Java Web services, 585
- Javascript alert() method, 498
- joined table query, 242, 520
- Joint Engine Technology, 28
- Just-In-Time compiler, 488
- Key parameter, 311, 312
- Keys folder, 56, 57, 58, 59, 60, 61, 560
- key identification, 18  
  candidate key, 19  
  and entity integrity, 18, 19  
  foreign keys, 19  
  primary key, 18, 19
- KeyValuePair structure, 310
- LIKE, 101, 102, 175, 183, 232, 234, 242, 243, 271, 374, 376, 382, 384, 385, 456, 520, 522, 523, 527, 529, 532, 727, 735, 748
- List Box  
  AutoPostBack Property of, 515
- LoadDataRow, 120
- Local File System, 588
- Local IIS, 631, 632
- localhost, 95, 227, 231, 493, 598, 599, 614, 632, 633, 654, 704
- logical design, 13  
  for database development, 14
- LogIn Form  
  code for data query and operation sequence, 165  
  data binding to associated controls, 162
- LogIn page, 501, 505, 514, 518, 524, 526, 533, 539, 543, 555, 570, 573

- LogIn TableAdapter, 131
- LogIn Web form, 500, 501, 526
- LogInTableApt object, 165
- many-to-many relationship, 21, 22, 23, 82
- MapFacultyTable(), 204, 511
  - and functionality of, 513
- Me, 166, 187, 198, 313, 335, 359, 392, 420, 617
- Me.Close(), 225, 257
- Merge method, 115
- Mergefailed event, 116
- MessageBox, 367, 426, 429, 439, 444, 481
- MessageBox buttons, 426
- Microsoft.NET Framework, 87, 88
- Microsoft Access, 2, 3, 4, 6, 28, 455, 457
  - and connection instances, 96
  - data insertion with Visual Basic.NET
    - coding of data, 312, 313
    - coding using TableAdapter.Insert method, 315, 316, 317, 318
    - coding using TableAdapter.Update method, 318, 319, 320, 321, 322
  - creation of InsertWizardProject.vbproj, 305
  - development of graphical user interfaces, 305, 306, 307
  - development of Insert Query, 314, 315
  - validation of data, 308, 309, 310, 311, 312, 322, 323, 324, 325, 326, 327, 328, 329
- illustration of, 29
- objects and database files, 29
- role of queries in, 29
- sample database creation, 11, 36, 37, 39
  - Faculty Table creation, 37, 39
  - LogIn Table creation, 36, 37
  - relationship creation among tables, 41
  - table creation, 39
- sample project, 192, 193, 195, 197, 199, 201, 203, 205, 207, 209, 211, 213, 215, 217, 219, 221, 223, 225
- coding for Selection form, 199
- connection to data source by runtime objects, 193, 194
- data query using runtime objects, 192, 204, 205, 206, 207
- data query using runtime objects for Course form, 210, 212, 214, 216, 217, 218; and functionality of codes, 208, 209, 210, 212; and Select button coding, 209
- data query using runtime objects for student form, 219
- data query using runtime objects, for Faculty form, 199, 201; and functionality of codes, 199, 200, 201; and Select button coding, 201, 202, 203
- DataReader to query data, 196, 197
- DataSet-TableAdapter to query data, 194, 195, 196
- object cleanup in, 197, 198
- runtime object declaration, 193
- Select button event procedure, coding for, 220, 222, 223, 224, 225
- Student form\_Load event procedure, coding for, 219
- table creation in, 29
- Microsoft Data Engine, 138
- Microsoft Intermediate Language (MSIL), 488, 492
- Microsoft ODBC, 91
- Microsoft OLE DB, 91
- Microsoft SQL Server, 52, 91
  - sample database creation, 41, 47
    - faculty table creation, 49, 50
    - LogIn Table creation, 48, 49
    - relationship creation between LogIn and student table, 58
    - relationship creation among tables, 55, 56, 57, 58, 59, 60, 61
    - tables creation, 50, 52, 55
  - Microsoft SQL Server 2005 Express, 7, 55
  - Microsoft.Jet.OLEDB.4.0 driver, 97
  - MissingSchemaAction, 118
  - Modify Column, 66
  - MsgBox, 600, 628, 673, 675, 679, 689, 702
  - multiple input arguments, 34
  - multitable data deleting, 571
  - Name page, 283, 729, 730, 750, 751
  - Named Parameter Mapping, 101, 102
  - namespace attribute, 590
  - nested stored procedures
    - code and functionality of, 263, 264, 265
    - data query using, 263
  - Network Identification tab, 232
  - New Connection, 139, 152
  - New Database dialog, 46
  - NewRow, 119, 120, 123
  - nonclustered indexes, 32
  - normal forms, 23
  - Not Null, 64, 70
  - Not Populated, 65, 70
  - Nothing property, 196, 198
  - NULL, 48, 55, 67, 79, 308, 310, 334, 338, 339, 347
  - nvarchar, 48, 49, 50, 52
  - Object Browser page, 280, 282, 283, 398, 402, 472, 729, 749, 762, 770
  - Object Explorer, 46, 48, 50, 56, 58, 59, 60, 61
  - Object Linking and Embedding (OLE), 87
  - Object list, 457, 458
  - Object tool, 66, 68
  - ODBC. *See* Open Database Connectivity
  - ODBC.NET data provider, 92
  - ODBC Data Provider, 91, 92, 124
  - ODBC databases
    - and connection string, 95
  - ODBC Driver Manager (DM), 92

ODBC.NET, supports to other providers, 92  
 OdbcCommand, 98, 105, 108  
 OdbcDataAdapter, 107, 108  
 OdbcDataReader, 110  
 Ole, 107  
 OLE DB Data Provider, 93, 101, 124, 193  
     and connection instances, 96  
 OLE DB data source, 93  
 OLE DB.NET data access technique, 93  
     advantages of, 93  
 OleDb, 3  
 OLEDB, 87, 91, 93, 456  
 OLEDB and OLEDB.NET, compatibility  
     between, 93  
 OLEDB connection, 96  
 OleDb namespace, 93  
 OLEDB provider, 228  
 OLEDB.NET Data Provider, 91  
 OleDbCommand, 92, 98, 105, 295, 377, 457  
 OleDbConnection, 92, 96, 97, 193, 194, 227, 228  
 OleDbConnection classes, 193  
 OleDbDataAdapter, 93, 107  
 OleDbDataReader, 92, 110, 238, 243, 245, 377, 457  
 OleDbException, 97  
 OleDbExceptionError, 194  
 OleDbType, 377, 457  
 ON clause, 242, 520  
 On Delete Cascade, 74, 79, 80, 81, 82, 477, 575  
 one-to-many relationship, 21, 58, 74, 79, 80, 82,  
     148  
 one-to-one relationship, 21  
 Open Database Connectivity, 87, 124  
 Open() method, 97  
     and errors occurring, 97  
 Oracle Client namespace, 268  
 Oracle Data Provider, 93, 96, 102, 124, 268, 269,  
     275, 526, 726, 741, 759  
 Oracle database, 33  
     adding to VB.NET applications, 7  
     architecture of, 33  
     and control file use in, 35  
     data files and types of, 33  
     data insertion with Visual Basic.NET, 352, 353  
     functions and procedures stored in, 34, 35  
     indexes used and classification of, 35  
     and initialization parameter files, 35  
     levels and applications of, 33  
     password files and, 36  
     redo log files and, 36  
     sample database creation, 62, 63, 64  
         foreign key creation among tables, 74, 79, 80,  
             81, 82  
     LogIn table creation, 64, 65, 66, 67, 68, 69, 70,  
         71  
         table creation, 72, 74  
     syntax to create stored procedure in, 34  
     tables used for storing data and types of, 33, 34  
     views, 34

Oracle Database 10g Express Edition, 2, 3, 7, 62,  
     63, 74, 79, 80, 82, 83, 280, 282, 283, 398, 401,  
     472, 477, 574, 726, 729, 749, 762, 765, 770,  
     772  
 Oracle database assignment operator, 527  
 Oracle database connection, 95  
 Oracle database environment  
     stored procedures in, 280  
         syntax of creating, 280, 281  
         syntax of creating a package in, 281, 282, 283  
 Oracle database sample project, 266  
     coding of the Selection form, 273  
     connection to data source with runtime objects,  
         269, 270  
     connection string configuration, 267, 268  
 CourseList\_SelectedIndexChanged() event  
     procedure and coding used, 291, 292  
 data query, 268  
     using DataReader and coding used, 272, 273  
     using Oracle package for course form, 286,  
         287, 288, 289, 290, 293  
     using runtime objects for Course form, 277,  
         279, 280  
     using runtime objects for Faculty form, 274,  
         275, 276, 277  
     using TableAdapter, 270, 272  
     using TableAdapter and coding used, 272  
 declaration of runtime objects, 269  
 Faculty.Course Package creation for course  
     form, 283, 284, 285, 286  
 Oracle Database 10g Express Edition  
     intallation, 266  
 package in, 280  
     and query method selection, 277  
     stored procedures in, 280  
         syntax of creating stored procedure in, 280, 281  
 Oracle Database XE Server, 266  
 Oracle Database, sample project, 266, 267, 269,  
     271, 273, 275, 277, 279, 281, 283, 285, 287,  
     289, 291, 293  
 Oracle Development Tools (ODT), 283  
 Oracle Parameter objects, 271  
 OracleClient, 228, 268, 269, 278, 380, 401, 449,  
     526, 529, 530, 533, 546, 724, 726, 742, 759  
 OracleCommand, 98, 105, 108, 385, 452, 547  
 OracleCommand object, 276  
 OracleConnection, 96, 229, 727, 744, 760, 761  
 OracleConnection class, 269  
 OracleDataAdapter, 107, 108  
 OracleDataReader, 110, 276, 289, 384, 385, 530,  
     532, 727, 746, 749, 766  
 OracleParameter objects, 271, 272, 276, 749, 767  
 Oracle-related objects, 269  
 OracleSelectRTOBJECT, 191, 266, 267, 269, 271,  
     273, 275, 277, 279, 281, 283, 285, 287, 289,  
     291, 293  
 OracleType.Char, 271  
 OracleType.Number, 743, 748, 760

OUT, 282, 285  
**OUTER JOIN**, 118  
**OUTPUT**, 264

package, 138, 285, 286, 472, 597, 729, 731, 732, 733, 750, 751, 752, 753  
 body part, 282  
 component, 397, 398  
 definition part, 282  
**Page Events**, 497, 504, 508, 517, 526, 536

**Parameter class**, 125  
 and constructors of, 99  
 constructors and properties of, 99, 100  
 parameter mappings, 100  
 for Data Providers, 101, 102

**Parameter object**, 98, 99, 100, 102, 125, 509  
 add to Parameters collection  
 ways used for adding, 103  
 and properties of, 99

**ParameterCollection class**, 102  
 and methods involved, 102

**ParameterName**, 102, 103, 234, 272

**Parameters collection**, 88, 98, 99, 101, 102, 103, 105, 125, 196, 201, 232, 233, 249, 255, 261, 288, 363, 394, 480, 500, 509, 538, 570, 637, 724, 740, 748, 758, 767

**Parameters collection property**, 554

**Parameters property**, 99

parent stored procedure, 262, 263

partitioned table, 33

passing by reference, 222, 320, 343, 344, 604, 605, 651

passing-by-value, 604

password file, 33, 36

**PasswordChar property**, 164

**PerformClick() method**, 317, 338

Perl Web services, 585

persistent parameter file, 35

physical design, for database development, 13, 14

PictureBox, 132, 179, 327, 328

PL/SQL. *See* Procedural Language Extension for SQL

Port Type, 584

Position property, 514, 515, 540

Positional Parameter Mapping, 101, 103

Precision, 64

primary data files, 30

primary key, 11, 18, 19, 48, 50, 65, 70, 71, 72, 74

Procedural Language Extension for SQL, 284, 285, 299, 399, 400, 472, 731, 751, 763, 771

Project Assembly files, 491

Project Data Sources, 162, 173, 325

Property Page, 533

Provider parameter, 95, 227, 228

Publish Web Site, 632

query, 29

Query Analyzer, 247

query applications, 129

**Query Builder**, 159, 160, 168, 169, 174, 175, 182, 183, 185, 314, 336, 337, 345, 349, 350, 411, 418, 419, 432, 457, 482  
 for Update and Delete query, 419, 420, 432, 433

query data, developing code to, 166

query method, 134, 207

Read() method, 112, 204, 512, 524, 604, 651

ReadXml, 119

Redirect() method, 504, 505, 513, 519, 537, 629

redo log files, 33, 36

Reference Table Column List, 79, 80, 81, 82

Reference Table Name, 74, 79, 80, 81, 82

referential integrity, 19, 32, 413, 551

referential integrity rules, 19

regular table, 33

RejectChanges, 121

relational data model, 10, 11, 14, 17

relational databases  
 structure and components of, 3

Relations collection, 113

Remote Data Objects (RDO), 87, 124

Remote Procedure Call (RPC), 583

Rename Column, 66

repeating groups, 24

REPLACE, 281

Request object, 498

RequiredFieldValidator, 501, 502, 540, 541

Response object, 498, 500, 504, 505, 510, 512, 513, 519, 537, 627, 628, 673

RETURN, 248

RowChanged, 122

RowDeleted, 122

Rows collection, 113, 120, 123, 304

Rows.Count property, 202, 272

Rows.Item(0), 210

Run Stored Procedure dialog, 265, 389, 390, 466, 562, 607, 648, 692, 694

Run without Debugging, 632

runtime object method, 3, 5, 353, 354, 434, 435  
 for SQL Server database, 365, 370

runtime object method for data insertion  
 into Microsoft Access database, 373  
 modification of data connection strings, 373  
 modification of Faculty query strings, 375, 376, 377  
 modification of Imports commands, 373  
 modification of login query strings, 374  
 modifications to other forms, 377, 378

into Oracle database, 379, 380  
 modification of database connection strings, 380, 381  
 modification of Faculty Query Strings, 383, 384  
 modification of Imports commands, 380  
 modification of Login Query Strings, 382  
 modification to other forms, 384, 385

- runtime object (*cont.*)
    - into SQL Server database, 355
      - creation of data from window, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369
    - development of user interfaces, 369, 370, 371
  - runtime objects, 194, 226, 294
    - advantages over tools and wizards, 189
    - creation to query data from SQL Server database, 232, 233
    - for data-driven applications, 295
    - defined as, 192
    - and development of database applications, 129
  - Scale, 64
  - second normal form (2NF), 24, 25, 26
  - secondary data files, 30
  - SELECT, 89, 105, 106, 118, 157, 159, 160, 168, 174, 182, 183, 241, 270, 280, 282, 285, 317, 323, 337, 345, 350, 358, 371, 372, 385, 432, 448, 449, 450, 451, 452, 477
  - Select button coding, for data query, 175
    - sequences of, 175, 176
  - Select button event procedure, 176
    - code, modification for, 179
    - steps used in, 179
    - function to catch the matched image, 180, 181
  - Select Case structure, 180, 205, 206, 222
  - Select Page to Start dialog, 533
  - SELECT statement, 90, 106
  - SELECT that returns a single value, 182
  - SelectCommand, 89, 98, 107, 109, 117, 124, 125, 191, 192, 210, 233, 255, 288, 354, 435, 646
  - SelectedIndex, 172, 176, 186, 201, 214, 220, 243, 313, 334, 421, 523, 617, 658
  - SelectedIndexChanged, 713
  - SelectionForm class, 165
  - SelectWizard project, sample for application creation, 144, 145, 147, 149, 189
  - selForm object, 165
  - Sequence object, 65, 70
  - server, 28
  - Server Explorer, 247, 249, 250, 251, 252, 258, 259, 260, 280, 283, 295, 388, 389, 465, 466, 562, 607, 648, 691, 692, 693, 694, 695, 702, 712, 723
  - ServerName, 599
  - Service.asmx, 585, 588, 590, 591, 592
  - Set As Start Page, 505, 629, 680
  - SET command, 259, 264, 286, 732, 752
  - Show All Files button, 230, 380
  - Show Table Data, 389, 466, 693, 695
  - ShowFaculty(), 510, 511
  - Simple Object Access Protocol (SOAP), 488, 582
  - SizeMode, 179, 327
  - SOAP 1.1, 592
  - SOAP 1.2, 592
  - SOAP interfaces, 613
  - SOAP message, 583, 586, 592
  - SOAP namespaces, 582
  - source codes, and usage of, 6
  - Source file, 495
  - SQL Commands page, 280, 282
  - SQL Connection object, 600
  - SQL objects, 242
  - SQL query, dynamic, 159
  - SQL Server, 3, 5
  - SQL Server Data Provider, 91, 93, 96, 99, 101, 102, 104, 108, 116, 124, 230, 231, 236, 497, 599, 775
  - SQL Server database, 30, 49, 152, 227, 249, 388, 588, 589, 591, 593, 595, 597, 599, 601, 603, 605, 607, 609, 611, 613, 615, 617, 619, 621, 623, 625, 627, 629, 631
  - and ASP.NET Web service project for data insertion in, 633, 635, 637
  - clustered index used in, 32
  - as compared to Oracle, 33
  - and connection instances, 96
  - and connection string, 95
  - data files in, 30
  - data insertion with Visual Basic.NET
    - coding using Table Adapter.Insert method, 337, 338, 339, 340
    - coding using TableAdapter.Update method, 342, 343, 344
  - configuration of TableAdapters, 336, 337
  - data-binding procedure, 341
  - development of user interfaces, 331, 332
  - modification of data, 331
  - using stored procedures, 349, 350, 351, 352
  - validation of data, 344, 345, 346, 347, 348
  - validation and initialization of data, 332, 333, 334, 335
- editions available in, 30
- indexes used in, 32
- keys and relational database relationships, 31, 32
- storage and file types, 30
- stored procedures in, 31
- structure of, 31
- tables in, 30
- tasks performed by, 30
- transaction log files, 32
- views, as virtual tables, 31
- with Visual Basic.NET
  - validation of data, 348
- SQL Server database assignment operator, 527
- SQL Server database project, as sample, 226, 227, 229, 231, 233, 235, 237, 239, 241, 243, 245, 247, 249, 251, 253, 255, 257, 259, 261, 263, 265
- data query using nested stored procedures, 263
- data query using runtime objects, 229, 230, 238
  - connection to data source, 231, 232

DataReader to query data and coding for,  
   233, 234, 235  
 runtime object declaration, 230  
 selection form query, 235  
 TableAdapter to query data and coding for,  
   232, 233  
 data query using stored procedures, 246, 247,  
   248, 249, 251, 252, 253, 255, 256, 257, 258,  
   259, 260, 261, 262  
   and detailed coding of, 259  
 migration from Access to SQL Server and  
   Oracle databases, 226, 227, 228, 229  
 table joins for data retrieval, 240, 241, 243,  
   245  
 SQL Server Enterprise Manager, 247  
 SQL Server Management Studio Express, 41, 229,  
   280, 465, 543, 560, 563  
 SQL statements, 29, 106  
   and dynamic parameters, 102  
 SQL stored procedure, for data query, 295  
 SQL View, 457  
 SqlCommand, 98, 103, 105, 106, 107, 109, 190, 226,  
   237, 243, 276, 363, 377, 385, 452, 457, 520,  
   523, 547, 601  
 SqlCommand class, 109  
 SqlCommand objects, 104, 105, 109, 117, 242  
 SqlConnection, 96, 117, 190, 226, 227, 231, 232,  
   497, 727, 744, 760  
 SqlConnection class, 230  
 SqlConnection objects, 107, 109, 117  
 SqlDataAdapter, 107, 109, 110, 117, 190, 363  
 SqlDataAdapter class, 109  
 SqlDataAdapter object, 117  
 SqlDataReader, 106, 110, 112, 190, 226, 236, 238,  
   243, 245, 276, 377, 384, 385, 457, 530, 532,  
   727, 746, 749, 766, 768  
 SqlDataReader class  
   methods of, 111  
   properties of, 111  
   useful methods of, 111  
 SqlDataReader object  
   coding for using, 112  
 sqlDataSet, 117  
 sqlDataTable, 236, 237, 276  
 SqlDbType, 377, 385, 451, 452, 457, 531, 532, 548,  
   603, 646, 727, 736, 743, 746, 748, 760, 765,  
   770  
 SqlDbType.Text, 603, 743, 748, 760  
 SQLEXPRESS, 44, 95, 96, 227, 231, 598, 599  
 SQLOLEDB Data Provider, 93  
 SqlParameter, 103, 105, 232, 234, 236, 237  
 SqlParameter objects, 105  
 SQLSelectRTOBJECT, *See also*  
   AccessSelectRTOBJECT  
 SQLWebSelect project, 494  
   code for opening pages, 503, 504, 505  
   coding for procedures in Faculty page, 510, 511,  
     512, 513, 514  
   coding for selection of course information, 516,  
     517  
   for Course Page Loading Procedures, 517,  
     518, 519  
   for ending event procedures, 517, 518, 519  
   for procedure FillCourseReaderTextBox(),  
     523, 524, 525  
   for Select button's click event procedure, 519,  
     520, 521, 522  
   for SelectedIndexChanged event procedure,  
     522, 523  
   coding for selection of faculty information, 508  
   code for Page\_Load event procedure, 508,  
     509  
   code for Select button event procedure, 509,  
     510  
   coding to access and select data, 496, 497, 498,  
     499, 500  
   data validation in client side, 501, 502  
   user interface – course page creation, 514, 515  
     and AutoPostBack Property, 515, 516  
   user interface – faculty page creation, 505, 506  
   user interface – LogIn form creation, 495, 496  
   user interface – selection page creation, 502, 503  
   StartPosition property, 146, 147, 150, 654  
   static parameter file, 35  
   Step Into Stored Procedure, 252  
   stored procedures  
     advantages of, 34  
     body and exceptions section in, 34  
     calling of, 248  
     coding for, 248  
     and command for, 31  
     creation of, 247  
     for data insertion  
       into SQL Server database, insertion of user  
         interfaces, 387  
     data query using, for student form, 249, 250,  
       251, 252, 253, 286  
   Oracle database environment and, 280  
   prototype or syntax of creating, 247  
   syntax of, 248  
   testing of, 262  
   types of, 247  
   stored procedures for data insertion  
     into Oracle database  
       coding of data, 401, 402, 403, 404, 405  
       development of stored procedures, 398, 399,  
         401  
     into SQL Server database  
       coding of data, 390, 392, 393, 394, 395,  
         396  
       development of stored procedures, 388, 389,  
         390  
       insertion of user interfaces, 386, 387  
   StretchImage, 179, 327  
   String.Empty, 313, 359  
   Student form window, 245

Style Builder dialog, 506, 515  
 Style property, 506  
 Style Sheet, 496  
 system stored procedures, 247  
 System.Collections.Generic, 308, 309  
 System.Data, 88, 91, 93, 118, 119, 190, 193, 200,  
     219, 226, 230, 235, 239, 268, 269, 274, 278,  
     373, 380, 401, 449, 456, 497, 503, 508, 517,  
     526, 529, 530, 533, 536, 546, 602, 626, 672,  
     724, 726, 742, 759  
 System.Data.dll, 88  
 System.Data.Odbc, 190  
 System.Data.OleDb, 200, 219, 239  
 System.Data.OracleClient, 190, 226, 274, 449, 726,  
     742, 759  
 System.Data.SqlClient, 190, 226, 235, 278  
 System.Drawing(), 512  
 System.Drawing.Image.FromFile(), 179, 205  
 System.Web, 512, 590  
 System.Web.Services, 585, 590  
 System.Web.UI.WebControls.Image, 512  
 System.Xml.dll, 88  
 Systems Development Life Cycle, 13  
  
 TabIndex, 655  
 TableAdapter, 98, 136, 137, 159, 160, 161, 164,  
     174, 182, 183, 199, 304, 314, 315, 316, 318,  
     319, 322, 323, 326, 330, 331, 332, 334, 336,  
     337, 338, 339, 340, 342, 343, 344, 345, 349,  
     350, 352, 353, 374, 381, 405, 406, 411, 412,  
     413, 414, 415, 418, 419, 420, 423, 424, 425,  
     426, 427, 429, 431, 432, 434, 449, 481, 482  
 TableAdapter Configuration Wizard, 157, 162,  
     303  
 TableAdapter DBDirect methods, 304, 414  
 TableAdapter method, 201, 243, 244  
 TableAdapter Query Configuration Wizard, 168,  
     182, 301, 314, 316, 332, 336, 344, 345, 349,  
     350, 352, 353, 412, 418, 419, 432, 434  
 TableAdapter query method, 219  
 TableAdapter.Delete(), 4, 303, 411, 412, 414, 425,  
     481, 482  
 TableAdapter.Insert, 3, 302, 303, 306, 307, 312,  
     313, 314, 315, 316, 322, 330, 331, 334, 337,  
     405, 406  
 TableAdapter.Update, 4, 302, 303, 304, 306, 307,  
     312, 313, 314, 316, 318, 326, 330, 331, 334,  
     338, 340, 342, 349, 405, 406, 411, 412, 414,  
     415, 418, 423, 425, 426, 427, 429, 431, 481,  
     482  
 Tables and Columns dialog, 57, 58  
 Tables And Columns Specification, 57, 59  
 Tables, joined, data retrieval from multiple tables  
     by using, 240, 241, 243, 244  
 Target Location, 632  
 TextChanged event procedure, 317, 318, 344, 364,  
     443, 660, 669, 705, 713  
 third normal form (3NF), 24, 25, 26  
  
 tnsnames.ora file, 228, 267, 268  
 Toolbox window  
     data components in, 133  
         BindingNavigator, 136  
         BindingSource, 135, 136  
         DataGridView, 135  
         DataSet object, 134  
         TableAdapter, 136, 137  
 ToString, 604, 620, 651  
 ToString method, 112  
 transaction log files, 30  
 transaction logs, in SQL Server, 32  
 Try....Catch block, 96, 97, 662  
 tuples, 14, 19, 29  
 typed DataSet, 114, 143, 433, 611, 620  
  
 UDDI file, 613  
 UDDI, XML-based directory, 584  
 Universal Description, Discovery, and Integration  
     (UDDI), 582  
 untyped DataSet, 114, 620  
 UPDATE, 89, 90, 280, 358, 418, 432, 442, 448, 451,  
     685, 764  
 Update(), 3, 302, 303, 318, 319, 343, 354, 405, 406,  
     424, 426, 435  
 Update() method, 3, 108, 137, 302, 303, 318, 319,  
     343, 354, 405, 406, 415, 424, 426, 434, 435  
 UpdateCommand, 4, 89, 98, 107, 124, 303, 304,  
     354, 411, 435, 481  
 updating query string, 553  
 Use SQL Statements, 168  
 user-defined query, 159  
 Validate Data, 308, 357, 365, 390, 401, 418,  
     656  
 Validate(), 424  
 Validation controls, 501  
 Validation tab  
     five controls in, 501  
 Value property, 196  
 View Code, 157, 171, 176, 180, 185, 192, 199, 207,  
     230, 269, 309, 312, 313, 373, 380, 496, 503,  
     526, 657  
 View Designer, 161, 165, 172, 184, 186, 187, 192,  
     194, 196, 197, 199, 207, 214, 232, 233, 236,  
     253, 268, 272, 310, 315, 317, 324, 327, 393,  
     496, 504  
 Views, 31, 34  
 Visual Basic collection class, 308  
 Visual Basic programmers, 87  
 Visual Basic.NET  
     code for data query by using Fill() method, 165,  
     166, 167  
     data binding to associated controls, 161, 162,  
     163, 165  
     data-binding function in, 136  
     data selection query with, 130, 131  
 Data Sources Window, 137, 138, 139, 140, 141,  
     142, 143, 144

- DataGridView for query and display, 155, 156, 157, 159
- DataSet Designer uses, 159, 160, 161
- design tools and wizards, 3, 133, 134, 135, 136, 137, 138, 139, 140, 141, 143, 144
  - shortcoming of using, 189
  - tools and wizards usage in, 129
- functionalities of, 151, 152, 153, 154
- for sample application creation, 130, 131, 132, 133, 144, 147, 189
  - advantages of, 188
- code modification for Select button event, 179
  - and code for data query, 185, 186, 187, 188
  - data binding to associated controls, steps used in, 181, 185
  - data binding in Faculty form, 173, 174, 175
  - image storage in database and steps performed, 178, 179, 180
  - Me. Close(), 187
  - retrieving single data value, 168, 170
  - Select button coding for data query, 175, 176, 177, 178
  - SelectionForm coding, 171, 172
  - steps included in, 145, 147, 148, 149
- toolbox window
  - data components in, 133, 134, 135, 136, 137
- Visual Studio 2005, 129
- Visual Web Developer, 585
- Web application execution, events involved in, 492
- Web configuration file, 503, 724, 726, 740, 741, 758, 759
- Web forms, 488, 489, 490, 493
- Web interfaces, 593, 739
- Web methods, 5, 582, 584, 586, 587, 589, 590, 591, 593, 597, 600, 609, 610, 611, 612, 613, 616, 617, 622, 625, 626, 629, 635, 643, 652, 653, 656, 657, 658, 659, 661, 662, 667, 671, 672, 673, 675, 679, 682, 683, 695, 697, 706, 707, 726, 740, 742, 756, 757, 760, 774, 775
- Web pages
  - data display and manipulation data, 493
- Web reference, 612, 613, 614, 615, 616, 624, 625, 653, 654, 656, 657, 659, 670, 702, 703, 704, 714, 715, 739, 740, 757, 775
- Web-referenced class, 619, 627
- Web server, 490, 492, 493, 509, 578, 582, 583, 588, 613, 629, 630, 631, 632, 671, 674, 775
  - and installation process of, 4
- Web service
  - process of, 584
- Web Service Binding attribute, 590
- Web service class, 587, 590, 591, 593, 595, 596, 597, 598, 608, 610, 612, 635, 637, 653, 706, 707, 717, 718, 726, 742, 759
  - adding Web methods into, 597, 598
- Web service clients building
  - to consume Web service, 739
  - WebServiceOracleInsert, 757
    - modification to subroutines, 743
  - Web service deployment, to servers, 630, 631
    - copy files to virtual directory, 631, 632
    - publishing Web service, 632, 633
  - Web service namespaces, 589
  - Web service project
    - and components of, 585, 586
  - Web service proxy class, 586, 612, 614, 653, 654, 704
  - Web service URL, 613, 653, 703
  - Web services, 490, 582, 583, 588, 590, 713, 715, 717, 719, 721, 723, 775
    - coding of Web methods to perform, 598
      - and code for database queries, 602, 604
      - code for subroutines used, 604, 605, 606
      - connection strings in, 598, 599, 600
      - method to call procedure, 608, 609
      - modification of Web method, 600, 601, 602
      - stored procedure development to perform data query, 607
      - stored procedure development, WebSelectFacultySP, 607, 608
    - and components of, 583, 584
    - procedures involved in building of, 585, 587
      - considerations for, 586, 587
      - and structure of service project, 585, 586
    - structure and components of, 5
    - Windows-based Web service client project
      - building for utilizing, 612, 653
  - Web Services Description Language (WSDL), 582
  - Web Services Interoperability Organization, 590
  - Web Site dialog, 632
  - web.config, 496, 508, 586, 588, 590, 598, 599, 600, 632
  - Web-based application, 487, 498, 512, 555, 570
    - user interface in, 496
  - Web-based Web service, 5
  - Web-based Web service client project, 623
    - code modification for Back button event procedure, 629
    - code modification for Back button's click event procedure, 680, 681
    - code modification for event procedures, 625, 626, 627, 671, 672, 673, 674, 675, 676, 678, 716, 717, 718, 719, 720, 721, 723
    - code modification for SelectedIndexChanged event procedure, 678, 679
    - coding for TextChanged event procedure, 674, 675
    - new Web site project creation and Web page addition, 624, 669, 714
    - user-defined subroutines addition, 628
    - for using Web services, 668, 669, 713
    - Web service reference addition and Web form Window modification, 624, 625, 670, 671, 714, 715

- WebClientSQLInsert, 757  
**WebSelectCourseSP**  
  coding to call stored procedure, 649, 650, 651, 652  
  creating stored procedure for, 648  
**WebService**, 585, 589, 590, 683, 741, 758  
**WebService attributes**, 589  
**WebServiceBinding**, 585, 589  
**WebServiceOracleSelect project**, 725, 739  
  modifications in, 725  
  code in Web method GetSQLSelectSP, 733, 734  
  connection string, 726  
  namespace directories, 726  
  stored procedure WebSelectFacultySP, 729, 731, 732  
  Web method GetSQLSelect, 726, 727  
  Web method GetSQLSelectDataSet, 734, 735, 736, 738, 739  
  Web method GetSQLSelectSP, 728  
**WebServiceSQLSelect project**, sample, 633, 634  
  code-behind page coding development, 635, 636  
  GetSQLInsert method development, 640, 641, 642  
  GetSQLInsertCourse method development, 647, 648, 649  
  SetSQLInsertSP method development, 636, 637, 639  
  SQLInsertDataSet method development, 642, 644, 645, 646, 647  
  development procedure used, 635  
**WebServiceSQLUpdateDelete project**, sample  
  add Web reference to, 703, 704  
  modifications in graphical user interface, 704  
**WinClientSQLInsert project**, sample, 702  
  modifications to file folder and project files, 703  
**WinClientSQLUpdateDelete project**, 702, 703  
**window.close()**, 500, 680  
**Windows application**, 145  
**Windows authentication**, 44, 153, 250, 599  
**Windows Components Wizard**, 493  
**Windows Forms**, 488  
**Windows NT Security Authentication**, 96, 227  
**Windows-based form**, 496  
**Windows-based Web service**, 5  
**Windows-based Web service clients**  
  coding for subroutines, 619, 620, 621  
  develop code to consume Web service, 616, 617, 657  
  code to get information for course, 666, 667, 668  
  code to initialize and terminate client project, 657, 658  
  code to insert new course record in database, 658, 659, 660, 661  
  code to perform inserted data validation, 662, 663, 664, 665–680  
  develop code for Form\_Load event procedure, 617  
  develop code for Select button's click event procedure, 618, 619  
  develop graphical user interface for, 614, 615, 616, 654, 655, 657  
  develop subroutine ShowFaculty, 621, 622  
  for using Web services, 612, 653, 702, 703, 705, 707, 709, 711  
  modifications for event procedures, 705, 706, 707, 708, 709, 710, 712  
  modifications in file folder and project files, 702, 703  
  modifications to graphical user interface, 704  
  proxy class creation, 612, 613, 614, 615, 653, 654  
**With....End With**, 604  
**Wizards**, 125, 130, 131, 302  
**Write() method**, 498, 504, 510, 512, 627, 628, 673  
**WSDL file**, 613  
**WSDL terminology**, 583  
**XML Path Language (XPath)**, 488  
**XML Schema (XSD)**, 114, 134  
**XML Web service links**, 490  
**XML Web services**, 488, 490, 613  
**XML Web services Discovery**, 613  
**xmlns attribute**, 593