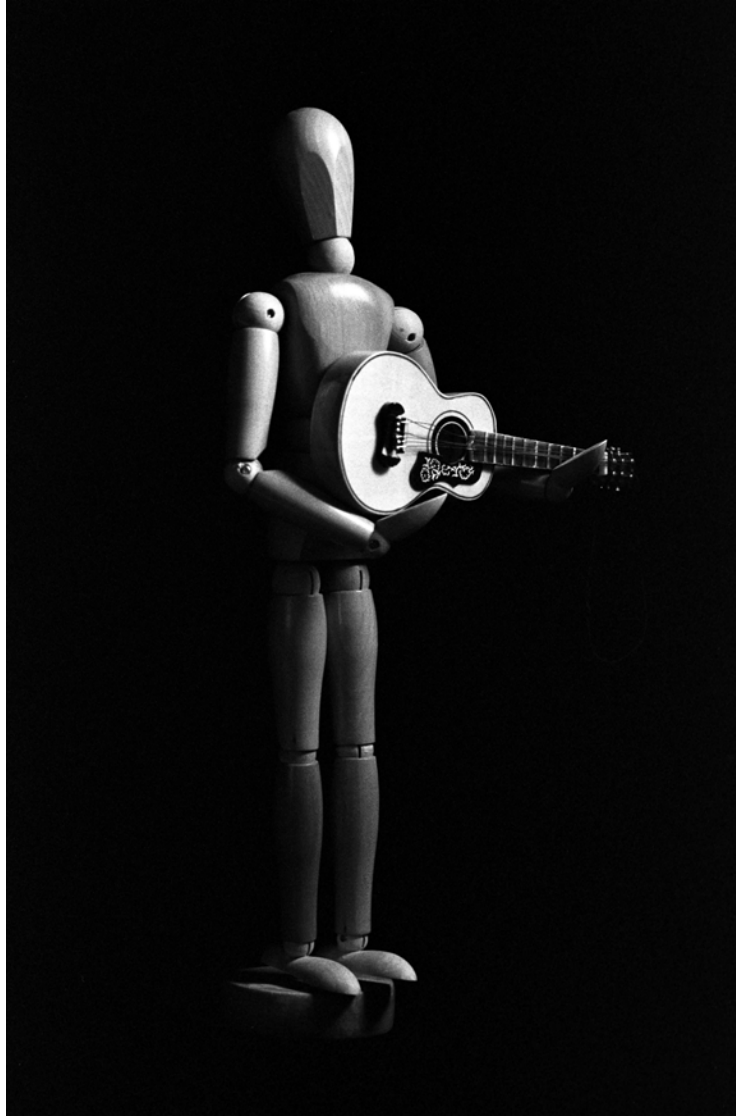


CHAPTER 12



Complex Data Access Objects

LEARNING OBJECTIVES

- *USE THE SERVICES OF A DAO FROM WITHIN ANOTHER DAO*
- *INCORPORATE CROSS-REFERENCE TABLES IN SQL QUERIES*
- *USE THE SERVICES OF MULTIPLE DAO OBJECTS IN UNIT TESTS*
- *REFACTOR CODE TO SIMPLIFY IMPLEMENTATION WHERE POSSIBLE*
- *UNDERSTAND HOW TO INSERT DATA INTO AND DELETE DATA FROM DATABASE CROSS-REFERENCE TABLES*
- *VERIFY CASCADE DELETE FOREIGN KEY CONSTRAINTS VIA UNIT TESTING*

INTRODUCTION

This chapter continues the discussion of data access objects (DAOs). It shows you how to create complex DAOs that utilize the services of other data access objects. The specific example I will employ in this chapter will be to complete the implementation of the EmployeeDAO by using the services of a TrainingDAO to populate the EmployeeVO.Courses property. The EmployeeVO.Courses property will undergo an evolutionary change in the underlying list's data type to better express the notion of a completed course. This will lead to a change in the name of the Courses property itself.

Implementation of the TrainingDAO will demonstrate the use of SQL queries that utilize cross-reference tables. Recall that a many-to-many or N-to-N database relationship is implemented with a cross-reference table. In the case of the EmployeeTraining database, the data indicating which courses an employee has completed is stored in the tbl_EmployeeCourse_XREF table.

Regarding the deletion of employees, although an organization may never delete employees from their database, it's a good idea to test the deletion of employees to ensure all related data is also deleted from the database. This should happen in foreign key relationships where the constraint is set to cascade delete. Implementing the TrainingDAO presents us with the perfect opportunity to perform this testing.

Finally, any new functionality added to the application will need to be tested either by adding new unit tests or modifying existing unit tests.

PLAN OF ATTACK

Table 12-1 lists the development tasks that need to be executed in this chapter.

Development Task	Considerations
Verify the CourseVO class	Recall that the CourseVO class was implemented in Chapter 9 - The Infrastructure Layer.
Implement the CourseDAO class	Create the CourseDAO class. This class will be responsible for interaction with the tbl_Course_LU table.
Implement CourseDAO unit tests	When both the CourseVO and CourseDAO have been coded up, create unit tests to test the CourseDAO class.
Implement the CompletedCourseVO class	Implement a new value object that represents the course an employee completed along with the date completed and the grade received.
Modify the EmployeeVO class	<ul style="list-style-type: none"> - Modify the EmployeeVO class to contain a list of Completed-CourseVOs (List<CompletedCourseVO>). - Change the name of the Courses property to CompletedCourses. - Refactor the code to simplify the implementation.
Implement the TrainingDAO class	Implement a TrainingDAO class whose responsibility it is to handle interaction with the tbl_EmployeeCourse_XREF table.
Modify the EmployeeDAO class	Modify the EmployeeDAO class to use the services of the TrainingDAO to populate the EmployeeVO.CompletedCourses property.
Implement unit tests	Test any new functionality added during this development cycle.

Table 12-1: Chapter Development Tasks

Development Task	Considerations
Verify employee data cascade delete	Verify the database cascade delete constraint specified between the tbl_Employee and tbl_EmployeeCourse_XREF tables.

Table 12-1: Chapter Development Tasks

Referring to table 21-1 — The development tasks start with the verification of the CourseVO. Recall that I initially implemented the CourseVO in Chapter 9 - The Infrastructure Layer. Upon inspecting the CourseVO I may decide to beef up its functionality to ensure all property values are initialized to an acceptable default state.

The CourseDAO class depends upon the CourseVO class. The purpose of the CourseDAO class is to interact with the tbl_Course table in the database. In this chapter I will just present the completed class without stepping through the implementation of each method. Once the CourseDAO class is completed I'll test it by creating one or more units tests.

Next, I will create a new value object class named CompletedCourseVO that better represents the notion of a completed course, the data for which is stored in the tbl_EmployeeCourse_XREF table. I will then revisit the EmployeeVO class and change the type of the Courses property to List<CompletedCoursesVO> and rename it CompletedCourses to better reflect the data it contains. (Note: This type of change often occurs during the course of application development when you realize what you created one day with good intentions will simply not work and must be changed. That such a change can be made without much hassle is the hallmark of a flexible application architecture.)

Following the change to the EmployeeVO class I will implement a new TrainingDAO class whose responsibility is to interact with the tbl_EmployeeCourse_XREF table. When I've finished the implementation of the TrainingDAO class I'll need to modify the EmployeeDAO class to use its services to populate the EmployeeVO.CompletedCourses property when retrieving employee data from the database. This development task will necessitate a discussion of business rules as they apply to the creation of employees resulting in a level of DAO method atomicity.

Finally, I'll need to add several new unit tests to test the new application functionality. Now that the TrainingDAO is created it will be a good time to test the cascade delete constraint between the tbl_Employee and tbl_EmployeeCourse_XREF tables. Recall that the foreign key constraint between the two tables is set to cascade delete, which means that when an employee record is deleted from the tbl_Employee table, all the corresponding records in the tbl_EmployeeCourse_XREF table should be automatically deleted as well.

Verify CourseVO

I originally created the CourseVO class back in Chapter 9 - The Infrastructure Layer. Now that it's time to start working on the CourseDAO, it's a good idea to revisit the CourseVO and see if any changes need to be made to the code.

Upon inspection I realize that I would like to initialize the instance properties with explicit values. The modified code for the CourseVO class is listed in example 12.1

12.1 CourseVO.cs (Modified)

```

1      using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5
6      namespace Infrastructure.ValueObjects
7      {
8          public class CourseVO
9          {
10             #region Public_Properties
11             public int CourseID { get; set; }
12             public String Code { get; set; }
13             public String Title { get; set; }
14             public String Description { get; set; }
15             #endregion
16
17             #region Constructors
18             public CourseVO(): this(0, string.Empty, string.Empty, string.Empty) { }

```

```

19
20     public CourseVO(int courseID, string code, string title, string description){
21         CourseID = courseID;
22         Code = code;
23         Title = title;
24         Description = description;
25     }
26
27     public CourseVO(string code, string title, string description){
28         Code = code;
29         Title = title;
30         Description = description;
31     }
32     #endregion
33
34     public override string ToString(){
35         return CourseID + " " + Code + " " + Title + " " + Description;
36     }
37 } // end CourseVO class
38 } // end namespace

```

Referring to example 12.1 — I've modified the default constructor on line 18 to pass in a set of default values for each of the properties. The CourseID property would normally be created with a default value of zero since it's an Int32 type, but the string properties would be created with default values of null, and that doesn't sit too well with me, so I initialize each of the string properties (Code, Title, and Description) to String.Empty.

Now that the CourseVO is squared away, I can start work on the CourseDAO.

IMPLEMENT COURSEDAO

The purpose of the CourseDAO is to interact with the tbl_Course table. The CourseDAO class depends upon the CourseVO class. The code for the CourseDAO is given in example 12.2.

12.2 CourseDAO.cs

```

1     using System;
2     using System.Collections.Generic;
3     using System.Linq;
4     using System.Text;
5     using System.Data;
6     using System.Data.Common;
7
8     using Microsoft.Practices.EnterpriseLibrary.Data;
9     using Microsoft.Practices.EnterpriseLibrary.Data.Sql;
10
11     using Infrastructure.Exceptions;
12     using Infrastructure.ValueObjects;
13
14     namespace DataAccess.DAO {
15         public class CourseDAO : BaseDAO {
16             #region SQL Command Constants
17             // SQL command parameter constants
18             private const string COURSE_ID = "@courseid";
19             private const string CODE = "@code";
20             private const string TITLE = "@title";
21             private const string DESCRIPTION = "@description";
22             #endregion SQL Command Constants
23
24             #region SQL Query String Constants
25             // SQL string constants
26             private const string SELECT_ALL_COLUMNS =
27                 "SELECT tbl_Course_LU.CourseID, " +
28                     "Code, " +
29                     "Title, " +
30                     "Description ";
31
32             private const string SELECT_ALL_COURSES =
33                 SELECT_ALL_COLUMNS +
34                 "FROM tbl_Course_LU";
35
36             private const string SELECT_COURSE_BY_ID =
37                 SELECT_ALL_COLUMNS +
38                 "FROM tbl_Course_LU " +
39                 "WHERE CourseID = " + COURSE_ID;
40
41             private const string INSERT_COURSE =
42                 "INSERT INTO tbl_Course_LU " +

```

```

43         "(Code, Title, Description) " +
44         "VALUES (" + CODE + ", " + TITLE + ", " + DESCRIPTION + ") " +
45         "SELECT scope_identity()";
46
47     private const string UPDATE_COURSE =
48         "UPDATE tbl_Course_LU " +
49         "SET Code = " + CODE + ", " +
50         "Title = " + TITLE + ", " +
51         "Description = " + DESCRIPTION + " " +
52         "WHERE CourseID = " + COURSE_ID;
53
54
55     private const string DELETE_COURSE =
56         "DELETE FROM tbl_Course_LU " +
57         "WHERE CourseID = " + COURSE_ID;
58     #endregion SQL Query String Constants
59
60
61     #region Constructor
62     public CourseDAO() : base(System.Reflection.MethodBase.GetCurrentMethod().DeclaringType) { }
63     #endregion Constructor
64
65     #region Public Methods
66
67     public List<CourseVO> SelectAllCourses() {
68         LogDebug("Entering SelectAllCourses() method...");
69         List<CourseVO> list = new List<CourseVO>();
70         IDataReader reader = null;
71
72         try {
73             DbCommand command = Database.GetSqlStringCommand(SELECT_ALL_COURSES);
74             reader = Database.ExecuteReader(command);
75             while (reader.Read()) {
76                 list.Add(FillInCourseVO(reader));
77             }
78         }
79         catch (Exception e) {
80             LogError("Exception in SelectAllCourses() method...", e);
81             throw new DBException("Exception in SelectAllCourses() method...", e);
82         }
83         finally {
84             CloseReader(reader);
85         }
86
87         return list;
88     }
89
90
91     public CourseVO SelectCourse(int id) {
92         LogDebug("Entering SelectCourse() method with CourseID = " + id);
93         CourseVO vo = null;
94         IDataReader reader = null;
95
96         try {
97             DbCommand command = Database.GetSqlStringCommand(SELECT_COURSE_BY_ID);
98             Database.AddInParameter(command, COURSE_ID, DbType.Int32, id);
99             reader = Database.ExecuteReader(command);
100            if (reader.Read()) {
101                vo = FillInCourseVO(reader);
102            }
103            else {
104                throw new DBException("No course found for CourseID: " + id);
105            }
106        }
107        catch (Exception e) {
108            LogError("Error getting course by CourseID " + id, e);
109            throw new DBException("Error getting course by CourseID " + id);
110        }
111        finally {
112            base.CloseReader(reader);
113        }
114
115        return vo;
116    }
117
118
119     public CourseVO InsertCourse(CourseVO vo) {
120         LogDebug("Entering InsertCourse() method for course: " + vo);
121         int courseID = 0;
122
123         try {

```

```

124         DbCommand command = Database.GetSqlStringCommand(INSERT_COURSE);
125         Database.AddInParameter(command, CODE, DbType.String, vo.Code);
126         Database.AddInParameter(command, TITLE, DbType.String, vo.Title);
127         Database.AddInParameter(command, DESCRIPTION, DbType.String, vo.Description);
128         courseID = Convert.ToInt32(Database.ExecuteScalar(command));
129     }
130     catch (Exception e) {
131         LogError("Exception inserting course into database!", e);
132         throw new DBException("Exception inserting course into database!", e);
133     }
134
135     return SelectCourse(courseID);
136 }
137
138
139 public CourseVO UpdateCourse(CourseVO vo) {
140     LogDebug("Entering the UpdateCourse() method with course: " + vo);
141     int rowsAffected = 0;
142
143     try {
144         DbCommand command = Database.GetSqlStringCommand(UPDATE_COURSE);
145         Database.AddInParameter(command, CODE, DbType.String, vo.Code);
146         Database.AddInParameter(command, TITLE, DbType.String, vo.Title);
147         Database.AddInParameter(command, DESCRIPTION, DbType.String, vo.Description);
148         Database.AddInParameter(command, COURSE_ID, DbType.Int32, vo.CourseID);
149         rowsAffected = Database.ExecuteNonQuery(command);
150     }
151     catch (Exception e) {
152         LogError("Error updating course record: " + e);
153         throw new DBException("Error updating course record: " + e);
154     }
155
156     if (rowsAffected == 0) {
157         LogError("No rows updated for course: " + vo);
158         throw new DBException("No rows updated for course: " + vo);
159     }
160     return SelectCourse(vo.CourseID);
161 }
162
163
164 public void DeleteCourse(CourseVO vo) {
165     LogDebug("Entering DeleteCourse() method with course: " + vo);
166     int rowsAffected = 0;
167
168     try {
169         DbCommand command = Database.GetSqlStringCommand(DELETE_COURSE);
170         Database.AddInParameter(command, COURSE_ID, DbType.Int32, vo.CourseID);
171         rowsAffected = Database.ExecuteNonQuery(command);
172     }
173     catch (Exception e) {
174         LogError("Error deleting course from database: " + vo);
175         throw new DBException("Error deleting course from database: " + vo, e);
176     }
177
178     if (rowsAffected == 0) {
179         LogError("No row deleted from database for course: " + vo);
180         throw new DBException("No row deleted from database for course: " + vo);
181     }
182 }
183
184 #endregion Public Methods
185
186 #region Private Methods
187
188 private CourseVO FillInCourseVO(IDataReader reader) {
189     CourseVO vo = new CourseVO();
190     vo.CourseID = reader.GetInt32(0);
191     vo.Code = reader.GetString(1);
192     vo.Title = reader.GetString(2);
193     vo.Description = reader.GetString(3);
194     return vo;
195 }
196
197 #endregion Private Methods
198
199 } // end CourseDAO class definition
200 } // end namespace
201

```

Referring to example 12.2 — The implementation of the CourseDAO class is straightforward. It's responsible for inserting, updating, retrieving, and deleting data from the tbl_Course table.

IMPLEMENT COURSEDAOTEST CLASS

Now that the CourseDAO class has been implemented it's time to implement a set of unit tests to exercise its various methods. The code for the CourseDAOTest class is listed in example 12.3.

12.3 CourseDAOTest.cs

```

1      using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5      using System.Drawing;
6      using System.Drawing.Imaging;
7      using System.IO;
8
9      using NUnit.Framework;
10
11     using Microsoft.Practices.EnterpriseLibrary.Data;
12     using Microsoft.Practices.EnterpriseLibrary.Data.Sql;
13     using System.Data.Sql;
14     using System.Data.SqlClient;
15     using System.Data;
16     using System.Data.Common;
17
18     using Infrastructure.ValueObjects;
19     using Infrastructure.Exceptions;
20     using DataAccess.DAO;
21
22
23     namespace Tests.DataAccess.DAO {
24         [TestFixture]
25         public class CourseDAOTest {
26
27             // course reference data matches first database record
28             private const int COURSE_ID = 1;
29             private const string CODE = "IST101";
30             private const string TITLE = "Introduction to Programming";
31             private const string DESCRIPTION = "Description test here...";
32
33
34             // other test constants
35             private const int NUMBER_OF_COURSES_IN_DATABASE = 5;
36
37             // fields
38             private CourseDAO _courseDAO;
39             private EmployeeDAO _employeeDAO;
40             private CourseVO _referenceCourse;
41
42
43             [SetUp]
44             public void Setup() {
45                 _courseDAO = new CourseDAO();
46                 _employeeDAO = new EmployeeDAO();
47                 _referenceCourse = new CourseVO(COURSE_ID, CODE, TITLE, DESCRIPTION);
48             }
49
50             [Test]
51             public void SelectAllCoursesTest() {
52                 List<CourseVO> list = _courseDAO.SelectAllCourses();
53                 Assert.IsTrue(list.Count == NUMBER_OF_COURSES_IN_DATABASE);
54             }
55
56
57             [Test]
58             public void InsertAndDeleteCourseTest() {
59                 CourseVO vo = new CourseVO();
60                 vo.Code = "FRE201";
61                 vo.Title = "Intermediate French";
62                 vo.Description = "Dites-moi quelque choses en francais!";
63                 Assert.IsTrue(vo.CourseID == 0);
64                 vo = _courseDAO.InsertCourse(vo);
65                 Assert.IsTrue(vo.CourseID > 0);
66                 List<CourseVO> list = _courseDAO.SelectAllCourses();
67                 Assert.IsTrue(list.Count == (NUMBER_OF_COURSES_IN_DATABASE + 1));
68                 _courseDAO.DeleteCourse(vo);
69                 list = _courseDAO.SelectAllCourses();
70                 Assert.IsTrue(list.Count == NUMBER_OF_COURSES_IN_DATABASE);
71             }

```

```

72
73
74     [Test]
75     public void UpdateCourseTest() {
76         CourseVO vo = _courseDAO.SelectCourse(1);
77         Assert.IsTrue(vo.CourseID == _referenceCourse.CourseID);
78         Assert.IsTrue(vo.Code == _referenceCourse.Code);
79         Assert.IsTrue(vo.Title == _referenceCourse.Title);
80         Assert.IsTrue(vo.Description == _referenceCourse.Description);
81         vo.Code = "1234";
82         vo.Title = "Changed";
83         vo.Description = "Changed here too...";
84         vo = _courseDAO.UpdateCourse(vo);
85         Assert.IsTrue(vo.CourseID == _referenceCourse.CourseID);
86         Assert.IsTrue(vo.Code == "1234");
87         Assert.IsTrue(vo.Title == "Changed");
88         Assert.IsTrue(vo.Description == "Changed here too...");
89         vo.Code = _referenceCourse.Code;
90         vo.Title = _referenceCourse.Title;
91         vo.Description = _referenceCourse.Description;
92         vo = _courseDAO.UpdateCourse(vo);
93         Assert.IsTrue(vo.CourseID == _referenceCourse.CourseID);
94         Assert.IsTrue(vo.Code == _referenceCourse.Code);
95         Assert.IsTrue(vo.Title == _referenceCourse.Title);
96         Assert.IsTrue(vo.Description == _referenceCourse.Description);
97     }
98 } // end class definition
99 } // end namespace
100

```

Referring to example 12.3 — This set of unit tests uses two DAO classes: CourseDAO and EmployeeDAO. The reference course data constants defined on lines 28 through 31 correspond to the first test record inserted into the tbl_Course table. By now you should be able to scan through the code and figure out on your own what the unit tests are doing.

Use a unit test naming convention that lends a hint about the type of test being conducted. For example, on line 58, the method name alone, InsertAndDeleteCourseTest(), pretty much sums up what this test is doing. Note that even though test data has been inserted into the various database tables, it's often necessary to create new database records as part of a unit test. Your goal in writing unit tests is to not only test the code you have written, but also to leave the database in a stable state. If you insert a new record into the database as part of a unit test, you should delete that temporary record at the end of the test to return the database to its pre-test state. The two unit tests that make up the TrainingDAOTest class demonstrate this concept.

IMPLEMENT COMPLETEDCOURSEVO

The CompletedCourseVO is a user-defined type whose purpose is to convey throughout the application the notion of a completed course, the data for which is stored in the tbl_EmployeeCourse_XREF table. I start the implementation of the CompletedCourseVO by looking at the tbl_EmployeeCourse_XREF table creation script that I have repeated here in example 12.4.

12.4 tbl_EmployeeCourse_XREF table creation script

```

1     IF NOT EXISTS (SELECT *
2                   FROM sys.objects
3                   WHERE object_id = OBJECT_ID(N'[dbo].[tbl_EmployeeCourse_XREF]') AND type in (N'U'))
4     BEGIN
5         CREATE TABLE [dbo].[tbl_EmployeeCourse_XREF] (
6             [FK_EmployeeID] [int] NOT NULL,
7             [FK_CourseID] [int] NOT NULL,
8             [DateCompleted] [datetime] NOT NULL,
9             [Grade] [float] NOT NULL
10            ) ON [PRIMARY]
11        END
12        GO

```

Referring to example 12.4 — Note that the Grade column is defined as an SQL float type. This corresponds to a C#.NET double type.

The code for the CompletedCourseVO class is given in example 12.5.

12.5 CompletedCourseVO.cs

```

1     using System;
2     using System.Collections.Generic;
3     using System.Linq;

```



```

4      using System.Text;
5
6      namespace Infrastructure.ValueObjects {
7          public class CompletedCourseVO {
8
9              #region Public Properties
10             public int EmployeeID { get; set; }
11             public CourseVO Course { get; set; }
12             public DateTime DateCompleted { get; set; }
13             public double Grade { get; set; }
14             #endregion
15
16             #region Constructors
17             public CompletedCourseVO() : this(0, new CourseVO(), DateTime.Now, 0.0) { }
18
19
20             public CompletedCourseVO(int employeeID, CourseVO course, DateTime dateCompleted, double grade)
21             {
22                 EmployeeID = employeeID;
23                 Course = course;
24                 DateCompleted = dateCompleted;
25                 Grade = grade;
26             }
27             #endregion
28
29             public override string ToString() {
30                 return EmployeeID + " " + Course + " " + DateCompleted + " " + Grade;
31             }
32         } // end CompletedCourseVO class definition
33     } // end namespace

```

Referring to example 12.5 — The CompletedCourseVO class implementation consists of four properties. Note that instead of just using the CourseID, which is what's captured in the tbl_EmployeeCourse_XREF table, I've created a field called Course, which is of type CourseVO. The rest of the implementation is straightforward. The default constructor explicitly initializes the VO's properties to known values.

Now that the CompletedCourseVO is completed, it's time to revisit the EmployeeVO and modify its code to use the new class.

Modify EmployeeVO To Use CompletedCourseVO

The EmployeeVO class needs to be modified to use the newly created CompletedCourseVO in place of the CourseVO. The code for the modified EmployeeVO class is listed in example 12.6.

12.6 EmployeeVO.cs (modified)

```

1      using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5
6      namespace Infrastructure.ValueObjects
7      {
8          public class EmployeeVO
9          {
10             #region Public Properties
11             public int EmployeeID { get; set; }
12             public string FirstName { get; set; }
13             public string MiddleName { get; set; }
14             public string LastName { get; set; }
15             public DateTime Birthday { get; set; }
16             public byte[] Picture { get; set; }
17             public DateTime HireDate { get; set; }
18             public bool IsActive { get; set; }
19             public List<CompletedCourseVO> CompletedCourses { get; set; }
20             #endregion
21
22             #region Constructors
23
24             public EmployeeVO() { }
25
26             public EmployeeVO(int employeeID, string first_name, string middle_name, string last_name,
27                 DateTime birthday, byte[] picture, DateTime hiredate, bool is_active,
28                 List<CompletedCourseVO> completedCourses){
29                 EmployeeID = employeeID;
30                 FirstName = first_name;

```

```

31         MiddleName = middle_name;
32         LastName = last_name;
33         Birthday = birthday;
34         Picture = picture;
35         HireDate = hiredate;
36         IsActive = is_active;
37         CompletedCourses = completedCourses;
38     }
39
40     public EmployeeVO(string first_name, string middle_name, string last_name,
41         DateTime birthday, byte[] picture, DateTime hiredate, bool is_active,
42         List<CompletedCourseVO> completedCourses){
43         FirstName = first_name;
44         MiddleName = middle_name;
45         LastName = last_name;
46         Birthday = birthday;
47         Picture = picture;
48         HireDate = hiredate;
49         IsActive = is_active;
50         CompletedCourses = completedCourses;
51     }
52
53     public override string ToString(){
54         return EmployeeID + " " + FirstName + " " + MiddleName + " " + LastName + " " +
55             Birthday.ToShortDateString() + " " + HireDate.ToShortDateString() + " " + IsActive;
56     }
57
58     #endregion
59 } // end EmployeeVO class
60 }

```

Referring to example 12.6 — I changed the name of the Courses property to CompletedCourses and changed its underlying data type to List<CompletedCoursesVO>. Note that changes like this will have a ripple effect throughout the application wherever the old EmployeeVO.Courses property is currently being used. Changes like this are commonplace in an evolving application development effort and are made as a matter of routine.

REFACTORING EMPLOYEEVO

The code for the EmployeeVO shown in example 12.6 above can stand to be improved on several fronts. First, the bold-faced code highlights not only the changes made to the class, but also how much code repetition there is in the class. Second, I'd like to explicitly set the properties to known default values. Example 12.7 gives the modified EmployeeVO class code.

12.7 EmployeeVO.cs (refactored)

```

1     using System;
2     using System.Collections.Generic;
3     using System.Linq;
4     using System.Text;
5
6     namespace Infrastructure.ValueObjects
7     {
8         public class EmployeeVO
9         {
10            #region Public Properties
11            public int EmployeeID { get; set; }
12            public string FirstName { get; set; }
13            public string MiddleName { get; set; }
14            public string LastName { get; set; }
15            public DateTime Birthday { get; set; }
16            public byte[] Picture { get; set; }
17            public DateTime HireDate { get; set; }
18            public bool IsActive { get; set; }
19            public List<CompletedCourseVO> CompletedCourses { get; set; }
20            #endregion Public Properties
21
22            #region Constructors
23
24            public EmployeeVO():this(0, string.Empty, string.Empty, string.Empty, DateTime.MinValue, null,
25                DateTime.MinValue, true, new List<CompletedCourseVO>()) { }
26
27
28            public EmployeeVO(string first_name, string middle_name, string last_name,
29                DateTime birthday, byte[] picture, DateTime hiredate, bool is_active,
30                List<CompletedCourseVO> completedCourses)
31                : this(0, first_name, middle_name, last_name, birthday, picture, hiredate, is_active,
32                    completedCourses) { }
33

```

```

34
35
36     public EmployeeVO(int employeeID, string first_name, string middle_name, string last_name,
37                       DateTime birthday, byte[] picture, DateTime hiredate, bool is_active,
38                       List<CompletedCourseVO> completedCourses){
39         EmployeeID = employeeID;
40         FirstName = first_name;
41         MiddleName = middle_name;
42         LastName = last_name;
43         Birthday = birthday;
44         Picture = picture;
45         HireDate = hiredate;
46         IsActive = is_active;
47         CompletedCourses = completedCourses;
48     }
49
50 #endregion Constructors
51
52 #region Overridden Object Methods
53
54     public override string ToString(){
55         return EmployeeID + " " + FirstName + " " + MiddleName + " " + LastName + " " +
56                Birthday.ToShortDateString() + " " + HireDate.ToShortDateString() + " " + IsActive;
57     }
58
59 #endregion Overridden Object Methods
60
61 } // end EmployeeVO class
62
63

```

Referring to example 12.7 — Note that all these changes were made without breaking any existing functionality, and with the exception of the change to the default constructor, were made for purely aesthetic purposes.

Let's now turn our attention to a DAO whose job it is to interact with the `tbl_EmployeeCourse_XREF` table.

IMPLEMENT TRAININGDAO

The hardest part about implementing this class was coming up with a decent name. I chose `TrainingDAO` because it best represented the notion of the type of data being captured in the `tbl_EmployeeCourse_XREF` table. At first I thought this functionality might fit into the `CourseDAO`, but when I started down that implementation path, I realized I was confusing the purpose of the `CourseDAO` class. This leads to the notion that an application's architecture should lend itself to being understood. To me, it simply wasn't intuitive that the `CourseDAO` would be the place to retrieve an employee's completed courses. Thus the need for another DAO whose name and purpose were a better fit for the job at hand.

The code for the `TrainingDAO` class is given in example 12.8.

12.8 TrainingDAO.cs

```

1     using System;
2     using System.Collections.Generic;
3     using System.Linq;
4     using System.Text;
5     using System.Data;
6     using System.Data.Common;
7
8     using Microsoft.Practices.EnterpriseLibrary.Data;
9     using Microsoft.Practices.EnterpriseLibrary.Data.Sql;
10
11     using Infrastructure.Exceptions;
12     using Infrastructure.ValueObjects;
13
14
15     namespace DataAccess.DAO {
16         public class TrainingDAO : BaseDAO {
17             #region SQL Command Parameters
18             //SQL command parameter constants
19             private const string FK_EMPLOYEE_ID = "@fkEmployeeId";
20             private const string FK_COURSE_ID = "@fkCourseID";
21             private const string DATE_COMPLETED = "@dateCompleted";
22             private const string GRADE = "@grade";
23             #endregion
24
25             #region SQL Query String Constants
26             //SQL query string constants

```

```

27     private const string SELECT_ALL_COLUMNS =
28         "SELECT FK_EmployeeID, FK_CourseID, DateCompleted, Grade ";
29
30     private const string SELECT_ALL_TRAINING_COMPLETED =
31         SELECT_ALL_COLUMNS +
32         "FROM tbl_EmployeeCourse_XREF";
33
34     private const string SELECT_COURSES_COMPLETED_BY_EMPLOYEE_ID =
35         SELECT_ALL_COLUMNS +
36         "FROM tbl_EmployeeCourse_XREF " +
37         "WHERE tbl_EmployeeCourse_XREF.FK_EmployeeID = " + FK_EMPLOYEE_ID;
38
39     private const string INSERT_COMPLETED_TRAINING =
40         "INSERT INTO tbl_EmployeeCourse_XREF " +
41         "(FK_EmployeeID, FK_CourseID, DateCompleted, Grade) " +
42         "VALUES (" + FK_EMPLOYEE_ID + ", " + FK_COURSE_ID + ", " + DATE_COMPLETED + ", " + GRADE +
43         ")";
44
45     private const string DELETE_EMPLOYEE_TRAINING =
46         "DELETE FROM tbl_EmployeeCourse_XREF " +
47         "WHERE FK_EmployeeID = " + FK_EMPLOYEE_ID;
48 #endregion
49
50 #region Constructor
51
52     public TrainingDAO() : base(System.Reflection.MethodBase.GetCurrentMethod().DeclaringType) { }
53
54 #endregion
55
56 #region Public Methods
57     public List<CompletedCourseVO> SelectAllCompletedTraining() {
58         LogDebug("Entering SelectAllCompletedTraining() method...");
59         List<CompletedCourseVO> list = new List<CompletedCourseVO>();
60         IDataReader reader = null;
61         try {
62             DbCommand command = Database.GetSqlCommand(SELECT_ALL_TRAINING_COMPLETED);
63             reader = Database.ExecuteReader(command);
64             CourseDAO courseDAO = new CourseDAO();
65             while (reader.Read()) {
66                 list.Add(FillInCompletedCourseVO(reader, courseDAO));
67             }
68         }
69         catch (Exception e) {
70             LogError("Exception in SelectAllCompletedTraining() method...", e);
71             throw new DBException("Exception in SelectAllCompletedTraining() method...", e);
72         }
73         finally {
74             CloseReader(reader);
75         }
76         return list;
77     }
78
79
80     public List<CompletedCourseVO> InsertCompletedTraining(CompletedCourseVO vo) {
81         LogDebug("Entering InsertCompletedTraining() method with CompletedCourseVO = " + vo);
82         List<CompletedCourseVO> list = new List<CompletedCourseVO>();
83         try {
84             DbCommand command = Database.GetSqlCommand(INSERT_COMPLETED_TRAINING);
85             Database.AddInParameter(command, FK_EMPLOYEE_ID, DbType.Int32, vo.EmployeeID);
86             Database.AddInParameter(command, FK_COURSE_ID, DbType.Int32, vo.Course.CourseID);
87             Database.AddInParameter(command, DATE_COMPLETED, DbType.DateTime, vo.DateCompleted);
88             Database.AddInParameter(command, GRADE, DbType.Double, vo.Grade);
89             Database.ExecuteScalar(command);
90         }
91         catch (Exception e) {
92             LogError("Error inserting completed training record!", e);
93             throw new DBException("Error inserting completed training record!", e);
94         }
95         return this.GetTrainingCompletedByEmployee(vo.EmployeeID);
96     }
97
98
99     public List<CompletedCourseVO> InsertEmployeeTrainingRecords(EmployeeVO employeeVO) {
100         LogDebug("Entering InsertEmployeeTrainingRecords() method with employeeVO = " + employeeVO);
101         foreach (CompletedCourseVO completedCourse in employeeVO.CompletedCourses) {
102             this.InsertCompletedTraining(completedCourse);
103         }
104         return this.GetTrainingCompletedByEmployee(employeeVO.EmployeeID);
105     }
106

```

```

107
108     public void DeleteEmployeeTrainingRecords(int employeeID) {
109         LogDebug("Entering DeleteEmployeeTraining() method for employeeID = " + employeeID);
110         try {
111             DbCommand command = Database.GetSqlStringCommand(DELETE_EMPLOYEE_TRAINING);
112             Database.AddInParameter(command, FK_EMPLOYEE_ID, DbType.Int32, employeeID);
113             Database.ExecuteNonQuery(command);
114         }
115         catch (Exception e) {
116             LogError("Error deleting employee training records!", e);
117             throw new DBException("Error deleting employee training records!", e);
118         }
119     }
120
121
122     public List<CompletedCourseVO> GetTrainingCompletedByEmployee(int employeeID) {
123         LogDebug("Entering GetTrainingCompletedByEmployee() method for employeeID = " + employeeID);
124         List<CompletedCourseVO> list = new List<CompletedCourseVO>();
125         IDataReader reader = null;
126         try {
127             DbCommand command =
128                 Database.GetSqlStringCommand(SELECT_COURSES_COMPLETED_BY_EMPLOYEE_ID);
129             Database.AddInParameter(command, FK_EMPLOYEE_ID, DbType.Int32, employeeID);
130             reader = Database.ExecuteReader(command);
131             CourseDAO courseDAO = new CourseDAO();
132             while (reader.Read()) {
133                 list.Add(FillInCompletedCourseVO(reader, courseDAO));
134             }
135         }
136         catch (Exception e) {
137             LogError("Error selecting courses completed for employeeID = " + employeeID, e);
138             throw new DBException("Error selecting courses completed for employeeID = "
139                 + employeeID, e);
140         }
141         finally {
142             CloseReader(reader);
143         }
144         return list;
145     }
146
147     #endregion Public Methods
148
149     #region Private Methods
150
151     private CompletedCourseVO FillInCompletedCourseVO(IDataReader reader, CourseDAO courseDAO) {
152         CompletedCourseVO vo = new CompletedCourseVO();
153         vo.EmployeeID = reader.GetInt32(0);
154         vo.Course = courseDAO.SelectCourse(reader.GetInt32(1));
155         vo.DateCompleted = reader.GetDateTime(2);
156         vo.Grade = reader.GetDouble(3);
157
158         return vo;
159     }
160
161     #endregion Private Methods
162
163 } // end TrainingDAO class definition
164 } // end namespace
165

```

Referring to example 12.8 — The TrainingDAO class converts data stored in the `tbl_EmployeeCourse_XREF` table into `CompletedCourseVO`s and vice versa. Note that the `FillInCompletedCourseVO()` method defined on line 151 takes two arguments: an `IDataReader` and a `CourseDAO`. The `courseDAO` parameter is used on line 154 to populate the `CompletedCourse.Course` property by retrieving the desired course from the `tbl_Course_LU` table. Note also that although the SQL server type of the `tbl_EmployeeCourse_XREF.Grade` column is `[float]`, it corresponds to the .NET double type, thus on line 156 the `IDataReader.GetDouble()` method is used to retrieve the value from the database.

Now that we have a TrainingDAO it's time to create some unit tests.

IMPLEMENT TRAININGDAO UNIT TESTS

Before using the TrainingDAO class somewhere in the application it needs to be tested. Example 12.9 gives the code for the TrainingDAOTest class.

12.9 TrainingDAOTest.cs

```

1      using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5      using System.Drawing;
6      using System.Drawing.Imaging;
7      using System.IO;
8
9      using NUnit.Framework;
10
11     using Microsoft.Practices.EnterpriseLibrary.Data;
12     using Microsoft.Practices.EnterpriseLibrary.Data.Sql;
13     using System.Data.Sql;
14     using System.Data.SqlClient;
15     using System.Data;
16     using System.Data.Common;
17     using System.Transactions;
18
19     using Infrastructure.ValueObjects;
20     using Infrastructure.Exceptions;
21     using DataAccess.DAO;
22
23     namespace Tests.DataAccess.DAO {
24
25         [TestFixture]
26         public class TrainingDAOTest {
27
28             private EmployeeVO newEmployee;
29             private CourseDAO _courseDAO;
30             private EmployeeDAO _employeeDAO;
31             private TrainingDAO _trainingDAO;
32
33             [SetUp]
34             public void Setup() {
35                 newEmployee = new EmployeeVO();
36                 newEmployee.FirstName = "Diana";
37                 newEmployee.MiddleName = "Lee";
38                 newEmployee.LastName = "Bath";
39                 newEmployee.Birthday = new DateTime(1961, 3, 3);
40                 newEmployee.HireDate = new DateTime(2012, 12, 24);
41                 newEmployee.IsActive = true;
42
43                 _courseDAO = new CourseDAO();
44                 _employeeDAO = new EmployeeDAO();
45                 _trainingDAO = new TrainingDAO();
46             }
47
48
49             [Test]
50             public void InsertAndDeleteTrainingTest() {
51                 // Insert new employee
52                 newEmployee = _employeeDAO.InsertEmployee(newEmployee);
53
54                 // Get list of courses
55                 List<CourseVO> courseList = _courseDAO.SelectAllCourses();
56
57                 // Should be at least four courses in database for testing
58                 Assert.IsTrue(courseList.Count > 4);
59
60                 // Assign some completed courses to employee
61                 newEmployee.CompletedCourses.Add(new CompletedCourseVO(newEmployee.EmployeeID,
62                                     courseList[0], DateTime.Now, 100.00));
63                 newEmployee.CompletedCourses.Add(new CompletedCourseVO(newEmployee.EmployeeID,
64                                     courseList[1], DateTime.Now, 100.00));
65                 newEmployee.CompletedCourses.Add(new CompletedCourseVO(newEmployee.EmployeeID,
66                                     courseList[2], DateTime.Now, 100.00));
67
68                 // Insert one completed employee training into tbl_EmployeeCourse_XREF table
69                 _trainingDAO.InsertCompletedTraining(new CompletedCourseVO(newEmployee.EmployeeID,
70                                     courseList[3], DateTime.Now, 100.00));
71

```

```

72         // Retrieve completed training record
73         List<CompletedCourseVO> employeeTrainingList =
74             _trainingDAO.GetTrainingCompletedByEmployee(newEmployee.EmployeeID);
75         Assert.IsTrue(employeeTrainingList.Count == 1);
76
77         // Insert remaining training
78         newEmployee.CompletedCourses = _trainingDAO.InsertEmployeeTrainingRecords(newEmployee);
79         Assert.IsTrue(newEmployee.CompletedCourses.Count == 4);
80
81         // Delete all employee training records
82         _trainingDAO.DeleteEmployeeTrainingRecords(newEmployee.EmployeeID);
83
84         // See if the employee has completed any training .. should return empty list
85         newEmployee.CompletedCourses =
86             _trainingDAO.GetTrainingCompletedByEmployee(newEmployee.EmployeeID);
87         Assert.IsTrue(newEmployee.CompletedCourses.Count == 0);
88
89         // Delete employee
90         _employeeDAO.DeleteEmployee(newEmployee);
91     }
92
93
94     [Test]
95     public void EmployeeCourseXREFCascadeDeleteTest() {
96         // Insert new employee
97         newEmployee = _employeeDAO.InsertEmployee(newEmployee);
98
99         // Get list of courses
100        List<CourseVO> courseList = _courseDAO.SelectAllCourses();
101
102        // Should be at least four courses in database for testing
103        Assert.IsTrue(courseList.Count > 4);
104
105        // Add some completed courses to employee
106        newEmployee.CompletedCourses.Add(new CompletedCourseVO(newEmployee.EmployeeID,
107                                                                courseList[0], DateTime.Now, 100.00));
108        newEmployee.CompletedCourses.Add(new CompletedCourseVO(newEmployee.EmployeeID,
109                                                                courseList[1], DateTime.Now, 100.00));
110        newEmployee.CompletedCourses.Add(new CompletedCourseVO(newEmployee.EmployeeID,
111                                                                courseList[2], DateTime.Now, 100.00));
112
113        // Insert completed training
114        _trainingDAO.InsertEmployeeTrainingRecords(newEmployee);
115
116        // Retrieve completed training
117        newEmployee.CompletedCourses =
118            _trainingDAO.GetTrainingCompletedByEmployee(newEmployee.EmployeeID);
119        Assert.IsTrue(newEmployee.CompletedCourses.Count == 3);
120
121        // Delete employee
122        _employeeDAO.DeleteEmployee(newEmployee);
123
124        // Should be no training for this employee
125        List<CompletedCourseVO> completedCourseList =
126            _trainingDAO.GetTrainingCompletedByEmployee(newEmployee.EmployeeID);
127        Assert.IsTrue(completedCourseList.Count == 0);
128    }
129 } // end TrainingDAOTest class definition
130 } // end namespace

```

Referring to example 12.9 — The `TrainingDAOTest` class consists of two unit test methods: `InsertAndDeleteTrainingTest()` and `EmployeeCourseXREFCascadeDeleteTest()`. The `TrainingDAOTest` class also uses the services of three DAOs: `TrainingDAO`, `CourseDAO`, and `EmployeeDAO`. In the first unit test, a new employee is inserted into the database. Next, all the courses are retrieved from the database and then three completed courses are added to the employee's `CompletedCourses` property. On line 69, one completed course for the employee is inserted into the database, followed on line 73 by the retrieval of that completed course to ensure the insertion and retrieval worked properly. On line 78 the remaining employee's completed courses are inserted into the database and the `Assert.IsTrue()` statement on line 79 verifies that the completed course count in the database for the employee now stands at four. On line 82 all the employee's training records are deleted and again an assertion is made to ensure the deletion works as expected. Finally, the employee is deleted from the database to return it to its pre-test state.

The second unit test verifies the cascade delete constraint that says when a `tbl_Employee` record is deleted, its related entries in the `tbl_EmployeeCourse_XREF` table should also be deleted.

Now that the `TrainingDAO` has been tested, it can be used in the `EmployeeDAO` to populate an `EmployeeVO.CompletedCourses` property.

Modify EmployeeDAO To Use TrainingDAO

The modified EmployeeDAO class is given in example 12.10.

12.10 EmployeeDAO.cs (modified)

```

1      using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5      using System.Data;
6      using System.Data.Common;
7
8      using Microsoft.Practices.EnterpriseLibrary.Data;
9      using Microsoft.Practices.EnterpriseLibrary.Data.Sql;
10
11     using Infrastructure.Exceptions;
12     using Infrastructure.ValueObjects;
13
14     namespace DataAccess.DAO {
15         public class EmployeeDAO : BaseDAO {
16             #region SQL Command Parameter Constants
17             // SQL Command Parameter Constants
18             private const String EMPLOYEE_ID = "@employeeID";
19             private const String FIRST_NAME = "@firstName";
20             private const String MIDDLE_NAME = "@middleName";
21             private const String LAST_NAME = "@lastName";
22             private const String BIRTHDAY = "@birthday";
23             private const String PICTURE = "@picture";
24             private const String HIRE_DATE = "@hireDate";
25             private const String IS_ACTIVE = "@isactive";
26             #endregion SQL Command Parameter Constants
27
28             #region SQL Query String Constants
29             // SQL Query String Constants
30             private const String SELECT_ALL_COLUMNS =
31                 "SELECT tbl_Employee.EmployeeID, " +
32                     "FirstName, " +
33                     "MiddleName, " +
34                     "LastName, " +
35                     "Birthday, " +
36                     "HireDate, " +
37                     "IsActive, " +
38                     "Picture "; // now retrieving Picture column
39
40             private const String SELECT_ALL_EMPLOYEES =
41                 SELECT_ALL_COLUMNS +
42                 "FROM tbl_Employee ";
43
44
45             private const String SELECT_EMPLOYEE_BY_ID =
46                 SELECT_ALL_COLUMNS +
47                 "FROM tbl_Employee " +
48                 "WHERE EmployeeID = " + EMPLOYEE_ID;
49
50
51
52
53             private const String INSERT_EMPLOYEE =
54                 "INSERT INTO tbl_Employee " +
55                     "(FirstName, MiddleName, LastName, Birthday, Picture, HireDate, IsActive) " +
56                 "VALUES (" + FIRST_NAME + ", " + MIDDLE_NAME + ", " + LAST_NAME + ", " + BIRTHDAY + ", " +
57                     "PICTURE + ", " + HIRE_DATE + ", " + IS_ACTIVE + ") " +
58                 "SELECT scope_identity()";
59
60
61             private const String UPDATE_EMPLOYEE =
62                 "UPDATE tbl_Employee " +
63                 "SET FirstName = " + FIRST_NAME + ", " +
64                     "MiddleName = " + MIDDLE_NAME + ", " +
65                     "LastName = " + LAST_NAME + ", " +
66                     "Birthday = " + BIRTHDAY + ", " +
67                     "Picture = " + PICTURE + ", " +
68                     "HireDate = " + HIRE_DATE + ", " +
69                     "IsActive = " + IS_ACTIVE + " " +
70                 "WHERE EmployeeID = " + EMPLOYEE_ID;
71
72
73             private const String DELETE_EMPLOYEE =

```



```

74         "DELETE FROM tbl_Employee " +
75         "WHERE EmployeeID = " + EMPLOYEE_ID;
76
77
78     private const String UPDATE_ISACTIVE_COLUMN =
79         "UPDATE tbl_Employee " +
80         "SET IsActive = " + IS_ACTIVE + " " +
81         "WHERE EmployeeID = " + EMPLOYEE_ID;
82
83     #endregion SQL Query String Constants
84
85     #region Constructor
86     public EmployeeDAO() : base(System.Reflection.MethodBase.GetCurrentMethod().DeclaringType) { }
87     #endregion Constructor
88
89     #region Public Methods
90
91     public List<EmployeeVO> SelectAllEmployees() {
92         LogDebug("Entering SelectAllEmployees() method..");
93
94         List<EmployeeVO> list = new List<EmployeeVO>();
95         IDataReader reader = null;
96         TrainingDAO trainingDAO = new TrainingDAO();
97
98         try {
99             DbCommand command = Database.GetSqlCommand(SELECT_ALL_EMPLOYEES);
100            reader = Database.ExecuteReader(command);
101            while (reader.Read()) {
102                list.Add(FillInEmployeeVO(reader, trainingDAO));
103            }
104        }
105        catch (Exception e) {
106            LogError("Exception in SelectAllEmployees() method.", e);
107            throw new DBException("Exception in SelectAllEmployees() method.", e,
108                BaseException.Severity.ERROR);
109        }
110        finally {
111            base.CloseReader(reader);
112        }
113
114        return list;
115    }
116
117
118     public EmployeeVO SelectEmployee(int id) {
119         LogDebug("Entering SelectEmployee() method for EmployeeID = " + id);
120         EmployeeVO vo = null;
121         IDataReader reader = null;
122         TrainingDAO trainingDAO = new TrainingDAO();
123
124         try {
125             DbCommand command = Database.GetSqlCommand(SELECT_EMPLOYEE_BY_ID);
126             Database.AddInParameter(command, EMPLOYEE_ID, DbType.Int32, id);
127             reader = Database.ExecuteReader(command);
128             if (reader.Read()) {
129                 vo = FillInEmployeeVO(reader, trainingDAO);
130             }
131             else {
132                 throw new DBException("No employee found for EmployeeID: " + id);
133             }
134         }
135         catch (Exception e) {
136             LogError("Error getting employee by EmployeeID " + id, e);
137             throw new DBException("Error getting employee by EmployeeID " + id);
138         }
139         finally {
140             base.CloseReader(reader);
141         }
142
143         return vo;
144     }
145
146
147     public EmployeeVO InsertEmployee(EmployeeVO vo) {
148         LogDebug("Entering InsertEmployee() method with employee: " + vo.ToString());
149         int employeeID = 0;
150         try {
151             DbCommand command = Database.GetSqlCommand(INSERT_EMPLOYEE);
152             Database.AddInParameter(command, FIRST_NAME, DbType.String, vo.FirstName);
153             Database.AddInParameter(command, MIDDLE_NAME, DbType.String, vo.MiddleName);
154             Database.AddInParameter(command, LAST_NAME, DbType.String, vo.LastName);

```

```

155         Database.AddInParameter(command, BIRTHDAY, DbType.DateTime, vo.Birthday);
156         Database.AddInParameter(command, PICTURE, DbType.Binary, vo.Picture);
157         Database.AddInParameter(command, HIRE_DATE, DbType.DateTime, vo.HireDate);
158         Database.AddInParameter(command, IS_ACTIVE, DbType.Boolean, vo.IsActive);
159         employeeID = Convert.ToInt32(Database.ExecuteScalar(command));
160     }
161     catch (Exception e) {
162         LogError("Error inserting employee into database!", e);
163         throw new DBException("Error inserting employee into database!", e,
164             BaseException.Severity.ERROR);
165     }
166
167     return SelectEmployee(employeeID);
168 }
169
170
171 public EmployeeVO UpdateEmployee(EmployeeVO vo) {
172     LogDebug("Entering the UpdateEmployee() method with Employee: " + vo);
173     int rowsAffected = 0;
174
175     try {
176         DbCommand command = Database.GetSqlStringCommand(UPDATE_EMPLOYEE);
177         Database.AddInParameter(command, FIRST_NAME, DbType.String, vo.FirstName);
178         Database.AddInParameter(command, MIDDLE_NAME, DbType.String, vo.MiddleName);
179         Database.AddInParameter(command, LAST_NAME, DbType.String, vo.LastName);
180         Database.AddInParameter(command, BIRTHDAY, DbType.DateTime, vo.Birthday);
181         Database.AddInParameter(command, PICTURE, DbType.Binary, vo.Picture);
182         Database.AddInParameter(command, HIRE_DATE, DbType.DateTime, vo.HireDate);
183         Database.AddInParameter(command, IS_ACTIVE, DbType.Boolean, vo.IsActive);
184         Database.AddInParameter(command, EMPLOYEE_ID, DbType.Int32, vo.EmployeeID);
185         rowsAffected = Database.ExecuteNonQuery(command);
186     }
187     catch (Exception e) {
188         LogError("Error updating employee record: " + e);
189         throw new DBException("Error updating employee record: " + e);
190     }
191
192     if (rowsAffected == 0) {
193         LogError("No rows updated for employee: " + vo);
194         throw new DBException("No rows updated for employee: " + vo);
195     }
196     return SelectEmployee(vo.EmployeeID);
197 }
198
199
200 public void DeleteEmployee(EmployeeVO vo) {
201     LogDebug("Entering DeleteEmployee() method with employee: " + vo);
202     int rowsAffected = 0;
203
204     try {
205         DbCommand command = Database.GetSqlStringCommand(DELETE_EMPLOYEE);
206         Database.AddInParameter(command, EMPLOYEE_ID, DbType.Int32, vo.EmployeeID);
207         rowsAffected = Database.ExecuteNonQuery(command);
208     }
209     catch (Exception e) {
210         LogError("Error deleting employee from database: " + vo);
211         throw new DBException("Error deleting employee from database: " + vo, e);
212     }
213
214     if (rowsAffected == 0) {
215         LogError("No row deleted from database for employee: " + vo);
216         throw new DBException("No row deleted from database for employee: " + vo);
217     }
218 }
219
220
221 public void SetEmployeeIsActiveState(int id, bool state) {
222     LogDebug("Entering SetEmployeeIsActiveState() for EmployeeID = " + id + " and state = " +
223         state);
224     int rowsAffected = 0;
225
226     try {
227         DbCommand command = Database.GetSqlStringCommand(UPDATE_ISACTIVE_COLUMN);
228         Database.AddInParameter(command, IS_ACTIVE, DbType.Boolean, state);
229         Database.AddInParameter(command, EMPLOYEE_ID, DbType.Int32, id);
230         rowsAffected = Database.ExecuteNonQuery(command);
231     }
232     catch (Exception e) {
233         LogError("No rows updated for EmployeeID = " + id, e);
234         throw new DBException("No rows updated for EmployeeID = " + id, e);
235     }

```

```

236
237         if (rowsAffected == 0) {
238             LogError("EmployeeID " + id + " does not exist in the database.");
239             throw new DBException("EmployeeID " + id + " does not exist in the database.");
240         }
241     }
242
243     #endregion Public Methods
244
245     #region Private Methods
246
247     private EmployeeVO FillInEmployeeVO(IDataReader reader, TrainingDAO trainingDAO) {
248         EmployeeVO vo = new EmployeeVO();
249
250         vo.EmployeeID = reader.GetInt32(0);
251         vo.FirstName = reader.GetString(1);
252         vo.MiddleName = reader.GetString(2);
253         vo.LastName = reader.GetString(3);
254         vo.Birthday = reader.GetDateTime(4);
255         vo.HireDate = reader.GetDateTime(5);
256         vo.IsActive = reader.GetBoolean(6);
257
258         if (!reader.IsDBNull(7)) {
259             vo.Picture = (byte[])reader.GetValue(7);
260         }
261
262         vo.CompletedCourses = trainingDAO.GetTrainingCompletedByEmployee(vo.EmployeeID);
263
264         return vo;
265     }
266     #endregion Private Methods
267
268 } // end EmployeeDAO class
269 } // end namespace

```

Referring to example 12.10 — The changes to the EmployeeDAO class are highlighted in bold. I'll start with the changes made to the FillInEmployeeVO() method that begins on line 247. I've added a second parameter to the method of type TrainingDAO named trainingDAO. The trainingDAO parameter is then used on line 262 to populate an EmployeeVO's CompletedCourses property. In the SelectAllEmployees() and SelectEmployee() methods you can see where I've added the declaration for a local method variable of type TrainingDAO to be passed in to the FillInEmployeeVO() method.

Note that I did not add the capability to insert an employee's completed training when an employee is inserted into the database nor when an employee is updated. I believe these actions are best left to a business object, where complex DAO interaction can be controlled with transactions. This topic is covered in detail in the next chapter.

SUMMARY

One data access object can use the services of another data access object to create complex DAOs.

Your goal in writing unit tests is not only to test the code you have written, but also to leave the database in a stable state. If you insert a new record into the database as part of a unit test, you should delete that temporary record at the end of the test to return the database to its pre-test state.

Refactor code where possible to simplify implementation and remove repetitive code.

One purpose or goal of an application architecture is to help programmers understand how the application works. In other words, architecture should be an aid to understanding. Many design and implementation decisions should be made with this architectural goal in mind.

Trust your instincts. If something doesn't feel right about the code, it might not be right. Functionality that doesn't seem to fit in a DAO may be better placed in a business object.

REFERENCES

None

NOTES
