Lab Answer Key: Module 7: Accessing a Database

Lab: Retrieving and Modifying Grade Data

Exercise 1: Creating an Entity Data Model from The School of Fine Arts Database

Task 1: Build and generate an EDM by using a table from the SchoolGradesDB database

1. Start the MSL-TMG1 virtual machine if it is not already running.
2. Start the 20483B-SEA-DEV11 virtual machine.
3. Log on to Windows 8 as **Student** with the password **Pa$$w0rd**. If necessary, click **Switch User** to display the list of users.
4. Switch to the Windows 8 **Start** window and then type Explorer.
5. In the **Apps** list, click **File Explorer**.
6. Navigate to the **E:\Mod07\Labfiles\Databases** folder, and then double-click **SetupSchoolGradesDB.cmd**.
7. Close File Explorer.
8. Switch to the Windows 8 **Start** window.
9. Click **Visual Studio 2012**.
10. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
11. In the **Open Project** dialog box, browse to **E:\Mod07\Labfiles\Starter\Exercise 1**, click **GradesPrototype.sln**, and then click **Open**.
12. On the **File** menu, point to **Add**, and then click **New Project**.

13. In the **Add New Project** dialog box, in the **Installed Templates** list, click **Visual C#**, and then click **Class Library**.

14. In the **Name** box, type **Grades.DataModel**, and then click **OK**.

15. On the **Project** menu, click **Add New Item**.

16. In the **Add New Item – Grades.DataModel** dialog box, in the templates list, click **ADO.NET Entity Data Model**, in the **Name** box, type **GradesModel**, and then click **Add**.

17. In the Entity Data Model Wizard, on the **Choose Model Contents** page, click **Generate from database**, and then click **Next**.

18. On the **Choose Your Data Connection** page, click **New Connection**.

19. If the **Choose Data Source** dialog box appears, in the **Data source** list, click **Microsoft SQL Server**, and then click **Continue**.

20. In the **Connection Properties** dialog box, in the **Server name** box, type **(localdb)\v11.0**, in the **Select or enter a database name** list, click **SchoolGradesDB**, and then click **OK**.

21. In the Entity Data Model Wizard, on the **ChooseYour Data Connection** page, click **Next**.

22. On the **Choose Your Database Objectsand Settings** page, expand **Tables**, expand **dbo**, select the following tables, and then click **Finish**:

    - **Grades**
    - **Students**
    - **Subjects**
    - **Teachers**
    - **Users**

23. If the **Security Warning** dialog box appears, click **Do not show this message again**, and then click **OK**.

24. On the **Build** menu, click **Build Solution**.

Task 2: Review the generated code

1. In the designer window, review the entities that have been generated.
2. Review the properties and navigation properties of the **Grade** entity.
3. Right-click the heading of the **Grade** entity, and then click **Table Mapping**.
4. In the **Mapping Details – Grade** pane, review the mappings between the columns in the database table and the properties of the entity.
5. In Solution Explorer, expand **GradesModel.edmx**, expand **GradesModel.Context.tt**, and then double-click **GradesModel.Context.cs**.
6. In the code window, note that the wizard has created a **DbContext** object named **SchoolGradesDBEntities**.
7. In Solution Explorer, expand **GradesModel.tt**, and then double-click **Grade.cs**.
8. Note that the wizard has created one property for each column in the **Grades** database table.
9. On the **File** menu, click **Save All**.
10. On the **File** menu, click **Close Solution**.

Results: After completing this exercise, the prototype application should include an EDM that you can use to access The School of Fine Arts database.

Exercise 2: Updating Student and Grade Data by Using the Entity Framework

---

Task 1: Display grades for the current student

1. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.

2. In the **Open Project** dialog box, browse to **E:\Mod07\Labfiles\Starter\Exercise 2**, click **GradesPrototype.sln**, and then click **Open**.

3. In Solution Explorer, right-click GradesPrototype, and then click Set as StartUp Project.

4. On the **View** menu, click **Task List**.

5. In the **Task List** window, in the **Categories** List, click **Comments**.

6. Double-click the **TODO: Exercise 2: Task 1a: Find all the grades for the student** task.

7. In the code editor, click in the blank line below the comment, and then type the following code:

```
List<Grades.DataModel.Grade> grades = new
List<Grades.DataModel.Grade>();
foreach (Grades.DataModel.Grade grade in
SessionContext.DBContext.Grades)
{
   if (grade.StudentUserId ==
SessionContext.CurrentStudent.UserId)
   {
      grades.Add(grade);
   }
}
```

8. In the **Task List** window, double-click the **TODO: Exercise 2: Task 1b: Display the grades in the studentGrades ItemsControl by using databinding** task.

9. In the code editor, click in the blank line below the comment, and then type the following code:

```
studentGrades.ItemsSource = grades;
```

10. On the **Build** menu, click **Build Solution**.
11. On the **Debug** menu, click **Start Without Debugging**.
12. When the application loads, in the **Username** box, type **vallee**, and in the **Password** box, type **password99**, and then click **Log on**.
13. In the **Class 3C** view, click **Kevin Liu**.
14. Verify that Kevin Liu's grades are listed.
15. Note that the subject column uses the subject ID rather than the subject name, and then close the application.

Task 2: Display the subject name in the UI

1. In Visual Studio, in the **Task List** window, double-click the **TODO: Exercise 2: Task 2a: Convert the subject ID provided in the value parameter** task.
2. In the code editor, click in the blank line below the comment, and then type the following code:

```
int subjectId = (int)value;
var subject =
SessionContext.DBContext.Subjects.FirstOrDefault(s => s.Id ==
subjectId);
```

3. In the **Task List** window, double-click the **TODO: Exercise 2: Task 2b: Return the subject name or the string "N/A"** task.
4. In the code editor, delete the following line of code:

```
return value;
```

5. In the code editor, click in the blank line below the comment, and then type the following code:

```
return subject.Name != string.Empty ? subject.Name : "N/A";
```


Task 3: Display the GradeDialog view and use the input to add a new grade

1. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3a: Use the GradeDialog to get the details of the new grade** task.

2. In the code editor, click in the blank line below the comment, and then type the following code:

```
GradeDialog gd = new GradeDialog();
```


3. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3b: Display the form and get the details of the new grade** task.

4. In the code editor, click in the blank line below the comment, and then type the following code:

```
if (gd.ShowDialog().Value)
{
```


5. Click in the blank line below the final TODO comment in this **try** block, and then type the following code:

```
}
```

6. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3c: When**

the user closes the form, retrieve the details of the assessment grade from the form and use them to create a new Grade object task.

7. In the code editor, click in the blank line below the comment, and then type the following code:

```
Grades.DataModel.Grade newGrade = new Grades.DataModel.Grade();
newGrade.AssessmentDate = gd.assessmentDate.SelectedDate.Value;
newGrade.SubjectId = gd.subject.SelectedIndex;
newGrade.Assessment = gd.assessmentGrade.Text;
newGrade.Comments = gd.comments.Text;
newGrade.StudentUserId = SessionContext.CurrentStudent.UserId;
```

8. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3d: Save the grade** task.
9. In the code editor, click in the blank line below the comment, and then type the following code:

```
SessionContext.DBContext.Grades.Add(newGrade);
SessionContext.Save();
```

10. In the **Task List** window, double-click the **TODO: Exercise 2: Task 3e: Refresh the display so that the new grade appears** task.
11. In the code editor, click at the end of the comment, press Enter, and then type the following code:

```
Refresh();
```

Task 4: Run the application and test the grade-adding functionality

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. When the application loads, in the **Username** box, type **vallee**, and in the **Password** box, type **password99**, and then click **Log on**.
4. In the **Class 3C** view, click **Kevin Liu**.
5. Verify that the list of grades now displays the subject name, not the subject ID.
6. In the **Report Card** view, click **Add Grade**.
7. In the **New Grade Details** dialog box, in the **Subject** list, click **Geography**, in the **Assessment** box, type **A+**, in the **Comments** box, type **Well done!**, and then click **OK**.
8. Verify that the new grade is added to the list, and then close the application.
9. In Visual Studio, on the **File** menu, click **Close Solution**.

Results: After completing this exercise, users will be see the grades for the current student and add new grades.

Exercise 3: Extending the Entity Data Model to Validate Data

Task 1: Throw the ClassFullException exception

1. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, browse to **E:\Mod07\Labfiles\Starter\Exercise 3**, click **GradesPrototype.sln**, and then click **Open**.
3. In Solution Explorer, right-click **GradesPrototype**, and then click **Set as StartUp Project**.
4. In Solution Explorer, right-click **Grades.DataModel**, point to **Add**, and then click

Class.

5. In the **Add New Item – Grades.DataModel** dialog box, in the **Name** box, type **customTeacher.cs**, and then click **Add**.

6. In the code editor, modify the class declaration as shown in the following code:

```
public partial class Teacher
```

7. In the code editor, in the **Teacher** class, type the following code:

```
private const int MAX_CLASS_SIZE = 8;
```

8. In the code editor, in the **Teacher** class, type the following code:

```
public void EnrollInClass(Student student)
 {
    // Verify that this teacher's class is not already full.
    // Determine how many students are currently in the class.
    int numStudents = (from s in Students
        where s.TeacherUserId == UserId
        select s).Count();
    // If the class is already full, another student cannot be
enrolled.
    if (numStudents >= MAX_CLASS_SIZE)
    {
        // So throw a ClassFullException and specify the class
that is full.
        throw new ClassFullException("Class full: Unable to
enroll student", Class);
    }
    // Verify that the student is not already enrolled in
another class.
    if (student.TeacherUserId == null)
    {
```

```
            // Set the TeacherID property of the student.
            student.TeacherUserId = UserId;
        }
        else
        {
            // If the student is already assigned to a class, throw
    an ArgumentException.
            throw new ArgumentException("Student", "Student is
    already assigned to a
    class");
        }
    }
```

9. In the **Task List** window, double-click **the TODO: Exercise 3: Task 1a: Call the EnrollInClass method to assign the student to this teacher's class** task.

10. In the code editor, click in the blank line below the comment, and then type the following code:

```
SessionContext.CurrentTeacher.EnrollInClass(student);
```

11. In the **Task List** window, double-click the **TODO: Exercise 3: Task 1b: Save the updated student/class information back to the database** task.

12. In the code editor, click in the blank line below the comment, and then type the following code:

```
SessionContext.Save();
```

Task 2: Add validation logic for the Assessment and AssessmentDate properties

1. In Solution Explorer, right-click **Grades.DataModel**, point to **Add**, and then click **Class**.

2. In the **Add New Item – Grades.DataModel** dialog box, in the **Name** box, type **customGrade.cs**, and then click **Add**.

3. In the code editor, modify the class declaration as shown in the following code:

```
public partial class Grade
```

4. In the code editor, in the Grade class, type the following code:

```
public bool ValidateAssessmentDate(DateTime assessmentDate)
{
    // Verify that the user has provided a valid date.
    // Check that the date is no later than the current date.
    if (assessmentDate > DateTime.Now)
    {
        // Throw an ArgumentOutOfRangeException if the date is
after the current
date.
        throw new ArgumentOutOfRangeException("Assessment
Date", "Assessment date
must be on or before the current date");
    }
    else
    {
        return true;
    }
}
```

5. In the code editor, below the existing **using** directives, type the following code:

```
using System.Text.RegularExpressions;
```

6. In the code editor, in the **Grade** class, type the following code:

```
public bool ValidateAssessmentGrade(string assessment)
{
    // Verify that the grade is in the range A+ to E-.
    // Use a regular expression: A single character in the
range A-E at the start of
the string followed by an optional + or - at the end of the
string.
    Match matchGrade = Regex.Match(assessment, @"^[A-E][+-]?
$");
    if (!matchGrade.Success)
    {
        // If the grade is not valid, throw an
ArgumentOutOfRangeException.
        throw new ArgumentOutOfRangeException("Assessment",
"Assessment grade must be
in the range A+ to E-");
    }
    else
    {
        return true;
    }
}
```

7. In the **Task List** window, double-click the **TODO: Exercise 3: Task 2a: Create a Grade object** task.
8. In the code editor, click in the blank line below the comment, and then type the following code:

```
Grades.DataModel.Grade testGrade = new
Grades.DataModel.Grade();
```

9. In the **Task List** window, double-click the **TODO: Exercise 3: Task 2b: Call the**

ValidateAssessmentDate method task.

10. In the code editor, click in the blank line below the comment, and then type the following code:

```
testGrade.ValidateAssessmentDate(assessmentDate.SelectedDate.Va
lue);
```

11. In the **Task List** window, double-click the **TODO: Exercise 3: Task 2c: Call the ValidateAssessmentGrade** task.

12. In the code editor, click in the blank line below the comment, and then type the following code:

```
testGrade.ValidateAssessmentGrade(assessmentGrade.Text);
```


Task 3: Run the application and test the validation logic

1. On the **Build** menu, click **Build Solution**.
2. On the **Debug** menu, click **Start Without Debugging**.
3. When the application loads, in the **Username** box, type **vallee**, and in the **Password** box, type **password99**, and then click **Log on**.
4. When the application loads, click **Enroll Student**.
5. In the **Assign Student** dialog box, click **Eric Gruber**, in the **Confirm** message box, click **Yes**, and then in the **Error enrolling student** message box, click **OK**.
6. In the **Assign Student** dialog box, click **Close**.
7. In the **Class 3C** view, click **Kevin Liu**, and then click **Add Grade**.
8. In the **New Grade Details** dialog box, in the **Date** box, type tomorrow’s date,

and then click **OK**.

9. In the **Error creating assessment** message box, click **OK**.

10. In the **New Grade Details** dialog box, in the **Date** box, type **8/19/2012**, in the **Assessment** box, type **F+**, and then click **OK**.

11. In the **Error creating assessment** message box, click **OK**.

12. In the **New Grade Details** dialog box, in the **Assessment** box, type **A+**, in the **Comments** box, type **Well done!**, and then click **OK**.

13. Verify that the new grade is added to the list, and then close the application.

14. In Visual Studio, on the **File** menu, click **Close Solution**.

Results: After completing this exercise, the application will raise and handle exceptions when invalid data is entered.