Práctica 6: Asignar una entidad a procedimientos administrados

Contenido

En este tema se muestra cómo usar ADO.NET Entity Data Model Designer (Entity Designer) para asignar las operaciones de inserción, actualización y eliminación de un tipo de entidad a procedimientos almacenados. Las operaciones de inserción, actualización y eliminación de un tipo de entidad pueden utilizar las instrucciones SQL que genera el sistema automáticamente (de forma predeterminada) o pueden utilizar los procedimientos almacenados que especifique el programador. El código de aplicación que se usa para crear, actualizar y eliminar las entidades es el mismo tanto si se utilizan procedimientos almacenados para actualizar la base de datos como si no.

En este tutorial, asignará dos tipos de entidad (en el modelo conceptual) a procedimientos almacenados (en el modelo de almacenamiento) modificando el archivo .edmx usado en la aplicación CourseManager (para obtener más información, vea la sección Requisitos previos, más adelante en este tema). También escribirá código que insertará, actualizará y eliminará los tipos de entidad.

Nota:

Si no asigna las tres operaciones de inserción, actualización o eliminación de un tipo de entidad a procedimientos almacenados, se producirá un error en las operaciones no asignadas si se ejecutan en tiempo de ejecución y se producirá una excepción UpdateException.

1. Requisitos previos

Para completar este tutorial, debe compilar la aplicación CourseManager. Para obtener más información e instrucciones, vea el Tutorial rápido de Entity Framework. Después de compilar esta aplicación, modificará su archivo .edmx asignando dos tipos de entidad a los procedimientos almacenados.

Nota:

Dado que muchos de los temas de los tutoriales de esta documentación usan la aplicación CourseManager como punto de partida, se recomienda que use una copia de la misma en este tutorial, en lugar de modificar el código original de CourseManager.

Este tutorial supone que el lector tiene conocimientos básicos de Visual Studio, .NET Framework y sobre la programación en Visual C# o Visual Basic.

2. Asignar la entidad Person a procedimientos almacenados

Al asignar la operación de inserción de una entidad a un procedimiento almacenado, si el servidor crea el valor de clave principal para la fila insertada, debe asignar de nuevo este valor a la propiedad de clave de la entidad. En este ejemplo, el procedimiento almacenado **InsertPerson** devuelve la clave principal creada recientemente como parte del conjunto de resultados del procedimiento almacenado. La clave principal se asigna a la clave de entidad (**PersonID**) mediante la característica <**Agregar Result Binding**> de Entity Designer.

Nota:

Si asigna una operación de inserción, actualización o eliminación a un procedimiento almacenado que devuelva un parámetro de salida con valores enteros, se habilita la casilla **RowsAffectedParameter**. Si la casilla está activada para un parámetro y el valor devuelto es cero cuando se llama a la operación, se producirá una excepción OptimisticConcurrencyException.

Para asignar la entidad Person a procedimientos almacenados

1. Abra la solución CourseManager en Visual Studio.
2. En el Explorador de soluciones, haga doble clic en el archivo School.edmx.

   El archivo School.edmx se abre en ADO.NET Entity Data Model Designer (Entity Designer).

3. Haga clic con el botón secundario en el tipo de entidad **Person** y seleccione **Asignación de procedimientos almacenados**.

   Las asignaciones de procedimientos almacenados aparecen en la ventana **Detalles de la asignación**.

4. Haga clic en <**Seleccionar Insert Function**>.

   El campo se convierte en una lista desplegable de los procedimientos almacenados del modelo de almacenamiento que se puede asignar a tipos de entidad del modelo conceptual.

5. En la lista desplegable, seleccione **InsertPerson**.

   Aparecen las asignaciones predeterminadas entre los parámetros de procedimiento almacenado y las propiedades de entidad. Observe que las flechas indican la dirección de la asignación: se proporcionan valores de propiedad como parámetros de procedimiento almacenado.

6. Haga clic en <**Agregar Result Binding**>.

   El campo ya se puede modificar.

7. Escriba **NewPersonID**, el nombre del parámetro devuelto por el procedimiento almacenado **InsertPerson**. Presione **Entrar**.

   De forma predeterminada, **NewPersonID** se asigna a la clave de entidad **PersonID**. Observe que una flecha indica la dirección de la asignación: el valor de la columna de resultado se proporciona para la propiedad.

8. Haga clic en <**Seleccionar Update Function**> y seleccione **UpdatePerson** en la lista desplegable resultante.

   Aparecen las asignaciones predeterminadas entre los parámetros de procedimiento almacenado y las propiedades de entidad.

9. Haga clic en <**Seleccionar Delete Function**> y seleccione **DeletePerson** en la lista desplegable resultante.

   Aparecen las asignaciones predeterminadas entre los parámetros de procedimiento almacenado y las propiedades de entidad.

Las operaciones de inserción, actualización y eliminación de un tipo de entidad **Person** están asignadas ahora a procedimientos almacenados.

3. Asignar la entidad OfficeAssignment a procedimientos almacenados

En este ejemplo, se asigna el tipo de entidad **OfficeAssignment** a los procedimientos almacenados. En esta asignación se usa la opción **Usar Original Value** en la operación de actualización para habilitar una manera cómoda de comprobar la simultaneidad en el código de aplicación.

Para asignar la entidad OfficeAssignment a procedimientos almacenados

1. Haga clic con el botón secundario en el tipo de entidad **OfficeAssignment** y seleccione **Stored Procedure Mapping**.

   Las asignaciones de procedimientos almacenados aparecen en la ventana **Detalles de la asignación**.

2. Haga clic en **<Seleccionar Insert Function>** y seleccione **InsertOfficeAssignment** en la lista desplegable resultante.

   Aparecen las asignaciones predeterminadas entre los parámetros de procedimiento almacenado y las propiedades de entidad.

3. Haga clic en **<Agregar Result Binding>**.

   El campo ya se puede modificar.

4. Escriba **Timestamp**.
5. Haga clic en el campo vacío en la columna **Property/Value** junto a **Timestamp**.

   El campo se convierte en una lista desplegable de propiedades a las que podemos asignar el valor que devuelve el procedimiento almacenado **InsertOfficeAssignment**.

6. En la lista desplegable, seleccione **Timestamp**.
7. Haga clic en **<Seleccionar Update Function>** y seleccione **UpdateOfficeAssignment** en la lista desplegable resultante.

   Aparecen las asignaciones predeterminadas entre los parámetros de procedimiento almacenado y las propiedades de entidad. Las casillas aparecen en la columna **Usar Original Value** al lado de cada propiedad asignada.

8. Haga clic en el campo vacío en la columna **Propiedad** que corresponda al parámetro **OrigTimestamp** y seleccione **Timestamp** en la lista desplegable resultante.

   Entity Designer no realizó la asignación predeterminada porque el nombre de parámetro no coincidía exactamente con el nombre de propiedad.

9. Active el cuadro de la columna **Usar Original Value** que corresponda a la propiedad **Timestamp**.

   Cuando se intenta una actualización, se usa el valor de la propiedad **Timestamp** que se leyó originalmente de la base de datos al volver a escribir los datos en la base de datos. Si el valor no coincide con el de la base de datos, se iniciará una **OptimisticConcurrencyException**.

10. Haga clic en **<Agregar Result Binding>**.

    El campo ya se puede modificar.

11. Reemplace **<Agregar Result Binding>** con **Timestamp.**
12. Haga clic en el campo vacío en la columna **Property/Value** junto a **Timestamp**.

    El campo se convierte en una lista desplegable de propiedades a las que podemos asignar la columna de resultados que devuelve el procedimiento almacenado **UpdateOfficeAssignment**.

13. En la lista desplegable, seleccione **Timestamp**.
14. Haga clic en **<Seleccionar Delete Function>** y seleccione **DeleteOfficeAssignment** en la lista desplegable resultante.

    Aparecen las asignaciones predeterminadas entre los parámetros de procedimiento almacenado y las propiedades de entidad.

Las operaciones de inserción, actualización y eliminación de un tipo de entidad **OfficeAssignment** están asignadas ahora a procedimientos almacenados.

4. Construir la interfaz de usuario

A continuación, agregará dos formularios a la aplicación CourseManager. Un formulario proporciona una interfaz para ver y actualizar la información de instructor. El otro formulario proporciona una interfaz para ver y actualizar las asignaciones de la oficina.

Para construir la interfaz de usuario

1. Haga clic con el botón secundario en el proyecto **CourseManager** en el **Explorador de soluciones**, seleccione **Agregar** y seleccione **Nuevo elemento**.

   Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.

2. Seleccione **Windows Forms**, establezca el nombre del formulario en InstructorViewer.vb o InstructorViewer.cs, y haga clic en **Agregar**.

   Se agrega un formulario nuevo al proyecto y se abre en el diseñador de formularios. El nombre del formulario se establece en **InstructorViewer** y el texto en **InstructorViewer**.

3. Arrastre un control DataGridView del cuadro de herramientas al formulario y establezca su **Nombre** en **instructorGridView** en la ventana **Propiedades**.
4. Arrastre un control Button del cuadro de herramientas al formulario. Establezca su **Nombre** en **updateInstructor** y su **Texto** en **Update Instructor**.
5. Arrastre otro control **Button** del cuadro de herramientas al formulario. Establezca su **Nombre** en **viewOffices** y su **Texto** en **View Offices**.
6. Haga clic con el botón secundario en el proyecto **CourseManager** en el **Explorador de soluciones**, seleccione **Agregar** y seleccione **Nuevo elemento**.

   Aparecerá el cuadro de diálogo **Agregar nuevo elemento**.

7. Seleccione **Windows Forms**, establezca el nombre del formulario en OfficeViewer.vb u OfficeViewer.cs, y haga clic en **Agregar**.

   Se agrega un formulario nuevo al proyecto y se abre en el diseñador de formularios. El nombre del formulario se establece en **OfficeViewer** y el texto en **OfficeViewer**.

8. Arrastre un control ComboBox del cuadro de herramientas al formulario y establezca su **Nombre** en **instructorList**.
9. Arrastre un control TextBox del cuadro de herramientas al formulario y establezca su **Nombre** en **officeLocation**.
10. Arrastre un control **Button** del cuadro de herramientas al formulario. Establezca el valor de **Nombre** en **updateOffice** y el de **Texto** en **Update Office**.
11. En el **Explorador de soluciones**, haga doble clic en el formulario **CourseViewer.vb** o **CourseViewer.cs**.

    La vista de diseño del formulario **CourseViewer** aparece.

12. Arrastre un control **Button** del cuadro de herramientas al formulario.
13. En la ventana **Propiedades**, establezca la propiedad **Name** de **Button** en **viewInstructors** y establezca la propiedad **Text** en **View Instructors**.
14. Haga doble clic en el control **viewInstructorsButton**.

    Se abre el archivo de código subyacente para el formulario **CourseViewer**.

15. Agregue el código siguiente al controlador de eventos **viewInstructors_Click**:

```
InstructorViewer instructorViewer = new InstructorViewer();
instructorViewer.Visible = true;
```

16. Vuelva a la vista de diseño del formulario **InstructorViewer**.
17. Haga doble clic en el control **viewOfficesButton**.

    El archivo de código subyacente para el formulario **Form2** se abre.

18. Agregue el código siguiente al controlador de eventos **viewOffices_Click**:

```
OfficeViewer officeViewer = new OfficeViewer();
officeViewer.Visible = true;
```

La interfaz de usuario está completa ahora.

5. Ver y actualizar información del instructor

En este procedimiento, agregará código al formulario **InstructorViewer** que le permita ver y actualizar la información del instructor. Más específicamente, el código hace lo siguiente:

- Enlaza **DataGridView** a una consulta que devuelve información sobre los tipos **Person** que son instructores. Para obtener más información sobre cómo enlazar objetos a controles, vea Binding Objects to Controls (Entity Framework).

- Guarda cualquier cambio (inserciones, actualizaciones o eliminaciones) en el control **DataGridView** en la base de datos.

- Utiliza los procedimientos almacenados que asignamos anteriormente para escribir los datos en la base de datos al llamar a **SaveChanges()** en el controlador de eventos **updateInstructor_Click**.

Para ver y actualizar información del instructor

1. Con el formulario **InstructorViewer** abierto en el diseñador de formularios, haga doble clic en el formulario **InstructorViewer**.

   El archivo de código subyacente para el formulario **InstructorViewer** se abre.

2. Agregue las instrucciones **using** (C#) o **Imports** (Visual Basic) siguientes:

```
using System.Data.Objects;
using System.Data.Objects.DataClasses;
```

3. Agregue una propiedad a la clase **InstructorViewer** que representa el contexto del objeto:

```
// Create an ObjectContext instance based on SchoolEntity.
private SchoolEntities schoolContext;
```

4. En el controlador de eventos **InstructorViewer_Load**, agregue el código para inicializar el contexto del objeto y establecer el origen de datos para el control **DataGridView** en una consulta que devuelve todos los tipos **Person** que no tienen **null** en **HireDate**.

```
// Initialize schoolContext.
schoolContext = new SchoolEntities();

// Define the query to retrieve instructors.
ObjectQuery<Person> instructorQuery = schoolContext.People
    .Include("OfficeAssignment")
    .Where("it.HireDate is not null")
    .OrderBy("it.LastName");

// Execute and bind the instructorList control to the query.
instructorGridView.DataSource = instructorQuery.
    Execute(MergeOption.OverwriteChanges);
instructorGridView.Columns["EnrollmentDate"].Visible = false;
instructorGridView.Columns["OfficeAssignment"].Visible = false;
instructorGridView.Columns["StudentGrades"].Visible = false;
instructorGridView.Columns["Courses"].Visible = false;
```

5. Vuelva a la vista de diseño del formulario **InstructorViewer** y haga doble clic en el control **updateInstructorButton**.

   El controlador de eventos **updateInstructor_Click** se agrega al archivo de código subyacente.

6. Agregue el código al controlador de eventos **updateInstructor_Click** que guarda cualquier cambio realizado en la información del instructor en el control **instructorGridViewDataGridView**.

```
// Save object changes to the database, display a
// message, and refresh the form.
schoolContext.SaveChanges();
MessageBox.Show("Change(s) saved to the database.");
this.Refresh();
```

Presione Ctrl + F5 para ejecutar la aplicación. La información del instructor se puede ver y actualizar ahora haciendo clic en **View Instructors**, realizando modificaciones en la tabla que aparece y haciendo clic en **Update Instructor**.

6. Ver y actualizar la información de Office

En este procedimiento, agregará código al formulario **OfficeViewer** que le permita ver y actualizar la información de la asignación de oficina. Más específicamente, el código hace lo siguiente:

- Enlaza el **ComboBox** a una consulta que devuelve información del instructor.
- Muestra información de la ubicación de la oficina para el instructor seleccionado en **TextBox**.
- Utiliza los procedimientos almacenados que asignamos anteriormente para escribir los datos en la base de datos al llamar a **SaveChanges()** en el controlador de eventos **updateOffice_Click**.

Para ver y actualizar información de la oficina

1. Con el formulario **OfficeViewer** abierto en el diseñador de formularios, haga doble clic en el formulario **OfficeViewer**.

   El archivo de código subyacente para el formulario **OfficeViewer** se abre.

2. Agregue las instrucciones **using** (C#) o **Imports** (Visual Basic) siguientes:

```
using System.Data.Objects;
using System.Data.Objects.DataClasses;
```

3. Agregue una propiedad a la clase **OfficeViewer** que representa el contexto del objeto:

```
// Create an ObjectContext instance based on SchoolEntity.
private SchoolEntities schoolContext;
```

4. Agregue el método siguiente al formulario:

```
private void ExecuteInstructorQuery()
{
    // Define the query to retrieve instructors.
    ObjectQuery<Person> instructorQuery = schoolContext.People
        .Include("OfficeAssignment")
        .Where("it.HireDate is not null")
        .OrderBy("it.LastName");

    //Execute and bind the instructorList control to the query.
    //Using MergeOption.OverwriteChanges overwrites local data
    //with data from the database.
    instructorList.DataSource = instructorQuery
        .Execute(MergeOption.OverwriteChanges);
    instructorList.DisplayMember = "LastName";
}
```

   Este método ejecuta una consulta que devuelve información del instructor y enlaza los resultados al control **instructorListComboBox**.

5. En el controlador de eventos **OfficeViewer_Load**, agregue el código para inicializar el contexto del objeto y llamar a un método que enlaza el control **ComboBox** a una consulta que devuelve todos los tipos **Person** que no tienen el valor **null** en **HireDate**.

```
schoolContext = new SchoolEntities();
ExecuteInstructorQuery();
```

6. Vuelva a la vista de diseño del formulario **OfficeViewer** y haga doble clic en el control **instructorListComboBox**.

   El controlador de eventos **instructorList_SelectedIndexChanged** se agrega al archivo de código subyacente.

7. Agregue el código al controlador de eventos que muestra la ubicación de la oficina del instructor seleccionado en el control ListBox y deshabilita el control **updateOfficeButton**. Este control se habilitará cuando se realice un cambio en la ubicación de una oficina seleccionada.

```
Person instructor = (Person)this.instructorList.
    SelectedItem;

if (instructor.OfficeAssignment != null)
{
    this.officeLocation.Text = instructor.
        OfficeAssignment.Location.ToString();
}
else
{
```

```
        this.officeLocation.Text = "";
    }

    // Disable the updateOffice button until a change
    // has been made to the office location.
    updateOffice.Enabled = false;

    //forceChanges.Enabled = false;
```

8. Vuelva a la vista de diseño del formulario **OfficeViewer** y haga doble clic en el control **updateOfficeButton**.

   El controlador de eventos **updateOffice_Click** se agrega al archivo de código subyacente.

9. Agregue el código que guarda los cambios realizados en la información de oficina en el control **officeLocationTextBox**:

```
try
{
    Person currentInstructor = (Person)this.instructorList.
        SelectedItem;
    if (this.officeLocation.Text != string.Empty)
    {
        if (currentInstructor.OfficeAssignment != null)
        {
            currentInstructor.OfficeAssignment.Location
                = this.officeLocation.Text;
        }
        else
        {
            currentInstructor.OfficeAssignment
                = OfficeAssignment.CreateOfficeAssignment(
                currentInstructor.PersonID,
this.officeLocation.Text,
                new byte[8]);
        }
    }
    else
    {
        schoolContext.DeleteObject(currentInstructor
            .OfficeAssignment);
    }
    schoolContext.SaveChanges();
    MessageBox.Show("Change(s) saved to the database.");
}
catch (OptimisticConcurrencyException oce)
{
    MessageBox.Show(oce.Message + " The conflict "
        + "occurred on " + oce.StateEntries[0].Entity
        + " with key value " + oce.StateEntries[0].
        EntityKey.EntityKeyValues[0].Value);

    //forceChanges.Enabled = true;
}
catch (UpdateException ue)
{
    MessageBox.Show(ue.Message + " Click OK to retrieve "
```

```
            + "the latest data from the database.");
        ExecuteInstructorQuery();
        this.Refresh();
    }
    finally
    {
        // Disable the updateOffice button until another
        // change has been made to the location.
        updateOffice.Enabled = false;
    }
```

10. Vuelva a la vista de diseño del formulario **OfficeViewer** y haga doble clic en el control **officeLocationTextBox**.

    El controlador de eventos **officeLocation_TextChanged** se agrega al archivo de código subyacente.

11. Agregue código para habilitar el control **updateOfficeButton** cuando se haya realizado un cambio en la ubicación de la oficina seleccionada:

```
// Enable the udateOffice button when there is a change
// to write to the database.
updateOffice.Enabled = true;
```

La aplicación se ha completado ahora. Presione Ctrl+F5 para ejecutar la aplicación. Ahora puede ver y actualizar la información de oficina en el formulario **OfficeViewer**.

7. Administrar los conflictos de simultaneidad

En este procedimiento, agregará código al formulario **Office Viewer** que aplique los cambios del cliente a la base de datos después de producirse un conflicto de simultaneidad.

Para administrar los conflictos de simultaneidad

1. Haga doble clic en **InstructorViewer.vb** o en **InstructorViewer.cs** en el **Explorador de soluciones**.

   El formulario se abre el diseñador de formularios.

2. Haga doble clic en el botón **View Offices**.

   El archivo de código subyacente para el formulario **InstructorViewer** se abre.

3. Agregue el código siguiente al controlador de eventos **viewOffices_Click** para que se carguen dos formularios **OfficeViewer** cuando se hace clic en el botón **View Offices**.

```
OfficeViewer officeViewer2 = new OfficeViewer();
officeViewer2.Text = "Demonstrate Conflict";
officeViewer2.Visible = true;
```

4. Haga doble clic en **OfficeViewer.vb** o en **OfficeViewer.cs** en el **Explorador de soluciones**.

   El formulario se abre el diseñador de formularios.

5. Arrastre un control **Button** del cuadro de herramientas al formulario. Establezca el valor de **Nombre** en **forceChanges** y el de **Texto** en **Force Changes**.
6. Haga doble clic en el botón **Force Changes**.

   El archivo de código subyacente para el formulario **Office Viewer** se abre.

7. Agregue el código siguiente al controlador de eventos **forceChanges_Click** para que se apliquen al servidor los cambios en el cliente o los datos enlazados al control **instructorListComboBox** se actualicen desde la base de datos.

```
Person currentInstructor = (Person)this.instructorList
    .SelectedItem;
try
{
    currentInstructor.OfficeAssignment.Location
                = this.officeLocation.Text;

    // Using RefreshMode.ClientWins disables the
    // optimistic concurrency check.
    schoolContext.Refresh(RefreshMode.ClientWins,
            currentInstructor.OfficeAssignment);
    schoolContext.SaveChanges();
    MessageBox.Show("Change(s) saved to the database.");

    //forceChanges.Enabled = false;
}
catch (InvalidOperationException ioe)
{
    MessageBox.Show(ioe.Message + " Click OK to retrieve "
        + "the latest data from the database.");
    ExecuteInstructorQuery();
    this.Refresh();
}
```

8. Quite los comentarios de la línea de código `forceChanges = False` (Visual Basic) o `forceChanges = false;` (C#) en el controlador de eventos **instructorList_SelectedIndexChanged** para que el botón **Force Changes** se deshabilite cuando se seleccione un instructor nuevo.
9. Quite los comentarios de la línea de código `forceChanges = True` (Visual Basic) o `forceChanges = true;` (C#) en el controlador de eventos **updateOffice_Click** para que el botón **Force Changes** se habilite cuando se produzca un conflicto de simultaneidad.
10. Quite los comentarios de la línea de código `forceChanges = False` (Visual Basic) o `forceChanges = false;` (C#) en el controlador de eventos **forceChanges_Click** para que el botón **Force Changes** se deshabilite una vez aplicados los cambios a la base de datos.

Para ver cómo administra la aplicación un conflicto de simultaneidad, ejecute la aplicación (presione Ctrl+F5), haga clic en **View Instructors** y, a continuación, haga clic en **View Offices**. Actualice la ubicación de una oficina en el formulario **Office Viewer** e intente actualizar la misma ubicación de oficina en el otro formulario **Demonstrate Conflict**. Aparecerá un cuadro de mensaje para notificar el conflicto de simultaneidad. Para aplicar los cambios del formulario **Demonstrate Conflict** a la base de datos, haga clic en **Force Changes**.

8. Códigos

Esta sección contiene las versiones finales de los archivos de código subyacente para los formularios **OfficeViewer** e **InstructorViewer**.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Objects;
using System.Data.Objects.DataClasses;

namespace CourseManager
{
    public partial class InstructorViewer : Form
    {
        // Create an ObjectContext instance based on SchoolEntity.
        private SchoolEntities schoolContext;

        public InstructorViewer()
        {
            InitializeComponent();
        }

        private void viewOffices_Click(object sender, EventArgs e)
        {
            OfficeViewer officeViewer = new OfficeViewer();
            officeViewer.Visible = true;

            OfficeViewer officeViewer2 = new OfficeViewer();
            officeViewer2.Text = "Demonstrate Conflict";
            officeViewer2.Visible = true;
        }

        private void InstructorViewer_Load(object sender, EventArgs e)
        {
            // Initialize schoolContext.
            schoolContext = new SchoolEntities();

            // Define the query to retrieve instructors.
            ObjectQuery<Person> instructorQuery = schoolContext.People
                .Include("OfficeAssignment")
                .Where("it.HireDate is not null")
                .OrderBy("it.LastName");
```

```
            // Execute and bind the instructorList control to the
query.
            instructorGridView.DataSource = instructorQuery.
                Execute(MergeOption.OverwriteChanges);
            instructorGridView.Columns["EnrollmentDate"].Visible =
false;
            instructorGridView.Columns["OfficeAssignment"].Visible =
false;
            instructorGridView.Columns["StudentGrades"].Visible =
false;
            instructorGridView.Columns["Courses"].Visible = false;
        }

        private void updateInstructor_Click(object sender, EventArgs
e)
        {
            // Save object changes to the database, display a
            // message, and refresh the form.
            schoolContext.SaveChanges();
            MessageBox.Show("Change(s) saved to the database.");
            this.Refresh();
        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Objects;
using System.Data.Objects.DataClasses;

namespace CourseManager
{
    public partial class OfficeViewer : Form
    {
        // Create an ObjectContext instance based on SchoolEntity.
        private SchoolEntities schoolContext;

        public OfficeViewer()
        {
            InitializeComponent();
        }

        private void OfficeViewer_Load(object sender, EventArgs e)
        {
            schoolContext = new SchoolEntities();
            ExecuteInstructorQuery();
        }

        private void instructorList_SelectedIndexChanged(object
sender,
        EventArgs e)
        {
            Person instructor = (Person)this.instructorList.
                SelectedItem;
```

```
            if (instructor.OfficeAssignment != null)
            {
                this.officeLocation.Text = instructor.
                    OfficeAssignment.Location.ToString();
            }
            else
            {
                this.officeLocation.Text = "";
            }

            // Disable the updateOffice button until a change
            // has been made to the office location.
            updateOffice.Enabled = false;

            //forceChanges.Enabled = false;
        }

        private void updateOffice_Click(object sender, EventArgs e)
        {
            try
            {
                Person currentInstructor =
(Person)this.instructorList.
                    SelectedItem;
                if (this.officeLocation.Text != string.Empty)
                {
                    if (currentInstructor.OfficeAssignment != null)
                    {
                        currentInstructor.OfficeAssignment.Location
                            = this.officeLocation.Text;
                    }
                    else
                    {
                        currentInstructor.OfficeAssignment
                            = OfficeAssignment.CreateOfficeAssignment(
                            currentInstructor.PersonID,
this.officeLocation.Text,
                            new byte[8]);
                    }
                }
                else
                {
                    schoolContext.DeleteObject(currentInstructor
                        .OfficeAssignment);
                }
                schoolContext.SaveChanges();
                MessageBox.Show("Change(s) saved to the database.");
            }
            catch (OptimisticConcurrencyException oce)
            {
                MessageBox.Show(oce.Message + " The conflict "
                    + "occurred on " + oce.StateEntries[0].Entity
                    + " with key value " + oce.StateEntries[0].
                    EntityKey.EntityKeyValues[0].Value);

                //forceChanges.Enabled = true;
            }
            catch (UpdateException ue)
            {
                MessageBox.Show(ue.Message + " Click OK to retrieve "
                    + "the latest data from the database.");
```

```
                ExecuteInstructorQuery();
                this.Refresh();
            }
            finally
            {
                // Disable the updateOffice button until another
                // change has been made to the location.
                updateOffice.Enabled = false;
            }
        }

        private void officeLocation_TextChanged(object sender,
EventArgs e)
        {
            // Enable the udateOffice button when there is a change
            // to write to the database.
            updateOffice.Enabled = true;
        }

        private void forceChanges_Click(object sender, EventArgs e)
        {
            Person currentInstructor = (Person)this.instructorList
                .SelectedItem;
            try
            {
                currentInstructor.OfficeAssignment.Location
                            = this.officeLocation.Text;

                // Using RefreshMode.ClientWins disables the
                // optimistic concurrency check.
                schoolContext.Refresh(RefreshMode.ClientWins,
                        currentInstructor.OfficeAssignment);
                schoolContext.SaveChanges();
                MessageBox.Show("Change(s) saved to the database.");
                //forceChanges.Enabled = false;
            }
            catch (InvalidOperationException ioe)
            {
                MessageBox.Show(ioe.Message + " Click OK to retrieve "
                    + "the latest data from the database.");
                ExecuteInstructorQuery();
                this.Refresh();
            }
        }
        private void ExecuteInstructorQuery()
        {
            // Define the query to retrieve instructors.
            ObjectQuery<Person> instructorQuery = schoolContext.People
                .Include("OfficeAssignment")
                .Where("it.HireDate is not null")
                .OrderBy("it.LastName");
            //Execute and bind the instructorList control to the
query.
            //Using MergeOption.OverwriteChanges overwrites local data
            //with data from the database.
            instructorList.DataSource = instructorQuery
                .Execute(MergeOption.OverwriteChanges);
            instructorList.DisplayMember = "LastName";
        }
    }
}
```