Lesson 5: Data Persistence and Manipulation

List Data-Create-Update-Delete(CRUD) using EF

Writing and managing ADO.Net code for data access is a tedious and monotonous job. Microsoft has provided an O/RM framework called "Entity Framework" to automate database related activities for your application. Microsoft has given the following definition of Entity Framework:

The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write.

Entity framework is an Object/Relational Mapping (O/RM) framework. It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.

The ADO.NET Entity Framework is an Object Relational Mapper (ORM) included with the .NET Framework. It basically generates business objects and entities depending on the database tables. It provides basic CRUD operations, easily managing relationships among entities with the ability to have an inheritance relationship among entities. When using the EF we interact with an entity model instead of the application's relational database model.

TheCode

So, we will create our Model class first and will then create the table in the database. Let's first create the two model classes as in the following:

```
1. using System.Collections.Generic;
2. namespace ExampleCF.Models
```

```
{
    public class Teacher
    {
        public int TeacherId { get; set; }
        public string TeacherName { get; set; }
        public string Address { get; set; }
        public ICollection<Student> Students { get; set; }
    }
}

namespace ExampleCFA.Models
{
    public class Student
    {
        public int StudentId { get; set; }
        public string Name { get; set; }
        public string Class { get; set; }
        public int TeacherId { get; set; }
        public Teacher Teacher { get; set; }
    }
}
```

Create DB Connection

This is the heart of Code-First Approach. First of all we need a connection string, so we can connect to our database from our application. As in traditional ASP.NET, we specify this connection in the WebConfig file as in the following:

```
<connectionStrings>
    <add name="DbConnectionString"
      connectionString="Data Source=mypc-PC;Initial Catalog=CodeFirst;User ID=sa;
      Password=*******" providerName="System.Data.SqlClient" />
</connectionStrings>
```

We will now create our DBContext class. For the code-first approach we need to inherit from the DBContext base class and then override the OnModelCreating

method. OnModelCreating is the method where we specify all the mappings as shown below (note that the connection string is passed through the constructor):

```
1.  using System.ComponentModel.DataAnnotations.Schema;
2.  using System.Data.Entity;
3.  namespace ExampleCFA.Models
4.  {
5.      public class SchoolContext :DbContext
6.      {
7.          public SchoolContext()
8.              : base("name=DbConnectionString")
9.          {
10.         }
11.         public DbSet<Teacher> Publishers { get; set; }
12.         public DbSet<Student> Students { get; set; }
13.
14.         protected override void OnModelCreating(DbModelBuilder modelBuilder)
15.         {
16.             modelBuilder.Entity<Teacher>().HasKey(t => t.TeacherId); //primary key defination
17.             modelBuilder.Entity<Teacher>().Property(t => t.TeacherrId)
18.                 .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);  //identity col
19.             modelBuilder.Entity<Student>().HasKey(s => s.StudentId);
20.             modelBuilder.Entity<Student>().Property(s => s.StudentId)
21.                 .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
22.             modelBuilder.Entity<Student>().HasRequired(s => s.Student)
23.                 .WithMany(s => s.Students).HasForeignKey(s=>s.TeacherId); //Foreign Key
24.             base.OnModelCreating(modelBuilder);
25.         }
26.     }
27. }
```

This DbModelBuilder class maps POCO classes to the database schema. This method is called only once when the first instance of a derived context is created. The model for that context is then cached and is for all further instances of the context in the app domain. This caching can be disabled by setting the

ModelCaching property on the given ModelBuidler, but this can seriously degrade performance. More control over caching is provided through use of the DbModelBuilder and DbContext classes directly.

Create Controllers for CRUD operation:

```
1.  using System.Linq;
2.  using System.Web.Mvc;
3.  using ExampleCodeFirstApproch.Models;
4.  namespace ExampleCFA.Controllers
5.  {
6.      public class TeacherController : Controller
7.      {
8.          SchoolContext objContext;
9.          public TeacherController()
10.         {
11.             objContext = new SchoolContext();
12.         }
13.         #region List and Details Teacher
14.         public ActionResult Index()
15.         {
16.             var teachers = objContext.Teachers.ToList();
17.             return View(teachers);
18.         }
19.         public ViewResult Details(int id)
20.         {
21.             Teacher teacher =
22.               objContext.Teachers.Where(x=>x.TeacherId==id).SingleOrDefault();
23.             return View(teacher);
24.         }
25.         #endregion
26.         #region Create Teacher
27.         public ActionResult Create()
28.         {
29.             return View(new Teacher());
30.         }
31.         [HttpPost]
32.         public ActionResult Create(Teacher teacher)
```

```
33.         {
34.             objContext.Teachers.Add(teacher);
35.             objContext.SaveChanges();
36.             return RedirectToAction("Index");
37.         }
38.         #endregion
39.         #region edit teacher
40.         public ActionResult Edit(int id)
41.         {
42.             Teacher teacher = objContext.Teachers.Where(
43.               x => x.TeacherId == id).SingleOrDefault();
44.             return View(teacher);
45.         }
46.         [HttpPost]
47.         public ActionResult Edit(Teacher model)
48.         {
49.             Teacher teacher = objContext.Teachers.Where(
50.               x => x.TeacherId == model.TeacherId).SingleOrDefault();
51.             if (teacher != null)
52.             {
53.                 objContext.Entry(teacher).CurrentValues.SetValues(model);
54.                 objContext.SaveChanges();
55.                 return RedirectToAction("Index");
56.             }
57.             return View(model);
58.         }
59.        #endregion
60.         #region Delete Teacher
61.         public ActionResult Delete(int id)
62.         {
63.             Teacher teacher = objContext.Teachers.Find(id);
64.
65.             return View(publisher);
66.         }
67.         [HttpPost]
68.         public ActionResult Delete(int id, Teacher model)
69.         {
70.            var teacher =
71.              objContext.Teachers.Where(x => x.TeacherId == id).SingleOrDefault();
```

```
            if (teacher != null)
            {
                objContext.Teachers.Remove(teacher);
                objContext.SaveChanges();
            }
            return RedirectToAction("Index");
        }
        #endregion
    }
}
```

```
using System.Linq;
using System.Web.Mvc;
using ExampleCodeFirstApproch.Models;
namespace ExampleCFA.Controllers
{
    public class StudentController : Controller
    {
        SchoolContext objContext;
        public StudentController()
        {
            objContext = new SchoolContext();
        }
        #region List and Details Students
        public ActionResult Index()
        {
            var studs = objContext.Students.ToList();
            return View(studs);
        }
        public ViewResult Details(int id)
        {
            Student stud = objContext.Students.Where(x=>x.StudentId==id).SingleOrDefault();
            return View(stud);
        }
        #endregion
        #region Create Student
        public ActionResult Create()
        {
```

```
28.             return View(new Student());
29.         }
30.         [HttpPost]
31.         public ActionResult Create(Student stud)
32.         {
33.             objContext.Students.Add(stud);
34.             objContext.SaveChanges();
35.             return RedirectToAction("Index");
36.         }
37.         #endregion
38.         #region Edit Student
39.         public ActionResult Edit(int id)
40.         {
41.             Student stud = objContext.Students.Where(x => x.StudentId == id).SingleOrDefault();
42.             return View(stud);
43.         }
44.         [HttpPost]
45.         public ActionResult Edit(Student model)
46.         {
47.             Student stud = objContext.Students.Where(x => x.StudentId == model.StudentId).SingleOrDef
    ault();
48.             if (stud != null)
49.             {
50.                 objContext.Entry(stud).CurrentValues.SetValues(model);
51.                 objContext.SaveChanges();
52.                 return RedirectToAction("Index");
53.             }
54.             return View(stud);
55.         }
56.         #endregion
57.         #region Delete Student
58.         public ActionResult Delete(int id)
59.         {
60.             Student stud = objContext.Students.Find(id);
61.             return View(book);
62.         }
63.         [HttpPost]
64.         public ActionResult Delete(int id, Student model)
65.         {
```

```
66.             var stud = objContext.Students.Where(x => x.StudentId == id).SingleOrDefault();
67.             if (stud != null)
68.             {
69.                 objContext.Students.Remove(stud);
70.                 objContext.SaveChanges();
71.             }
72.             return RedirectToAction("Index");
73.         }
74.         #endregion
75.     }
76. }
```