**Creating Model Classes with LINQ to SQL**

이 튜토리얼의 목적은 ASP.NET MVC 어플리케이션에서 model 클래스를 만드는 하나의 메소드를 설명하는 것입니다. 이 튜토리얼에서, LINQ to SQL을 이용하여 어떻게 model 클래스를 빌드하고 데이터베이스에 연결하는지 설명할 것입니다.

예제로, 기본 Movie 데이터베이스 어플리케이션을 만들 것 입니다. 모든 데이터들은 controller actions에 바로 연결할 것입니다.

그리고, Repository 패턴을 어떻게 사용하는지 배울 것입니다. Repository 패턴은 약간의 추가 작업을 필요로 합니다. 그러나, 이 패턴을 사용함으로써 유연한 어플리케이션을 만들 수 있고, 쉽게 테스트 할 수 있습니다.

**What is a Model Class?**

MVC model은 MVC view 또는 controller에 포함되지 않은 모든 어플리케이션 로직을 가지고 있습니다. 특히, MVC model은 비즈니스 로직과 데이터 액세스 로직을 가지고 있습니다.

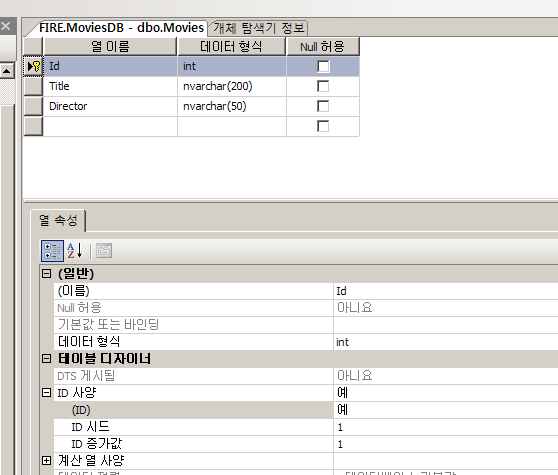
사용자는 데이터 액세스 로직 구현을 위한 다양한 기술들을 적용할 수 있습니다. 예를 들어, Microsoft Entity Framework, NHibernate, Subsonic, 또는 ADO.NET classes 등을 사용함으로써 데이터 액세스 클래스를 만들 수 있습니다.

이 튜토리얼에서는 데이터베이스에 쿼리와 업데이트하기 위해 LINQ to SQL을 사용할 것입니다. LINQ to SQL은 SQL 데이터베이스와 상호작용하는 간단한 메소드를 제공합니다. 그러나, ASP.NET MVC 프레임워크는 LINQ to SQL에만 한정적 인 것이 아니라, 다른 데이터 액세스 기술과도 잘 융합됩니다.

**Creating a Movie Database**

이 튜토리얼에서 model 클래스를 어떻게 구성하는 설명하기 위해 간단한 Movie 데이터베이스 어플리케이션을 만들 것입니다. 첫 단계로, 새로운 데이터 베이스를 만듭니다. (아래 표 참고)

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Allow Nulls** |
| Id | Int | False |
| Title | Nvarchar(200) | False |
| Director | Nvarchar(50) | False |

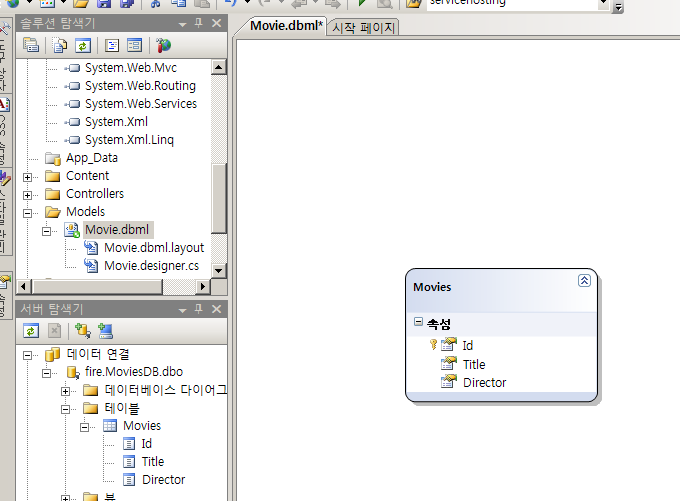


**Creating LINQ to SQL Classes**

MVC model은 데이터베이스 테이블을 의미하는 LINQ to SQL 클래스를 가지고 있어야 합니다. 이 클래스를 만드는 가장 쉬운 방법은 LINQ to SQL 템플릿을 추가 하는 것입니다.

(Model 폴더 우 클릭 – 추가 – Linq to sql 템플릿)

LINQ to SQL 클래스를 생성하면, Object Relational Designer 가 나타날 것입니다. 서버 탐색기에서 해당하는 데이터베이스를 Object Relational Designer로 드래그 할 수 있습니다. (아래 그림)



기본적으로, Object Relational Designer은 사용자가 드래그 한 데이터베이스 테이블과 같은 이름의 클래스를 만듭니다.

(저장 필수)

**Using LINQ to SQL in a Controller Action**

이 클래스를 만듦으로써, 데이터베이스와 통신할 수 있습니다. 이제, controller action에서 LINQ to SQL 클래스를 어떻게 사용하는지 설명할 것입니다. 우리는 MVC view에서 Movie 데이터베이스 테이블로부터 영화 목록을 보여줄 것입니다.

처음에, HomeController 클래스를 아래 코드와 같이 수정해야 합니다.

주의! Using 네임스페이스.Models

|  |
| --- |
| using System.Linq;  using System.Web.Mvc;  using MvcApplication1.Models;  namespace MvcApplication1.Controllers  {  [HandleError]  public class HomeController : Controller  {  public ActionResult Index()  {  var dataContext = new MovieDataContext();  var movies = from m in dataContext.Movies  select m;  return View(movies);  }  }  } |

위 코드에서 Index() action은 MoviesDB 데이터베이스를 나타내는 LINQ to SQL DataContext 클래스(the MovieDataContext)를 사용합니다. MovieDataContext 클래스는 Visual Studio Object Reletional Designer에 의해 생성되었습니다.

LINQ 쿼리는 데이터베이스 테이블에서 모든 movie와 통신하는 DataContext에 맞게 동작합니다. Movie 목록은 movies 라는 지역 변수에 명명되었습니다. 마지막으로, movie 목록은, view data로 view에 전달됩니다.

Movie을 보기 위해, Index view 역시 아래 코드로 수정해야 합니다.

|  |
| --- |
| <%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master" AutoEventWireup="true"  CodeBehind="Index.aspx.cs" Inherits="MvcApplication1.Views.Home.Index" %>  <%@ Import Namespace="MvcApplication1.Models" %>  <asp:Content ID="indexContent" ContentPlaceHolderID="MainContent" runat="server">  <ul>  <% foreach (Movie m in (IEnumerable)ViewData.Model)  { %>  <li> <%= m.Title %> </li>  <% } %>  </ul>  </asp:Content> |

View의 상단에 <%@ import namespace %> 을 포함하는 것을 주의하세요. Model 클래스를 작업하기 위해 이 네임스페이스가 필요합니다.

위 코드에서 view은 ViewData.Model 속성에 의해 표현될 아이템을 반복하여 보여줄 foreach 루프를 가지고 있습니다. 각 무비의 타이틀 속성이 보여질 것입니다.

ViewData.Model 속성 값은 IEnumerable 로 캐스팅하는 것을 주의하세요. ViewData.Model의 콘텐트를 loop 로 사용하기 위해 필수적입니다.

HomeController 클래스와 Index view을 수정하고 어플리케이션을 실행 시키면, 데이터베이스에 값이 없기 때문에 빈 화면이 나타날 것입니다.

데이터베이스에 값을 넣고 실행시켜 보세요.

**Using the Repository Pattern**

지금까지, controller action에서 바로 LINQ to SQL 클래스를 사용했습니다. 다시 말해 MovieDataContext 클래스에서 Index() controller action으로 바로 데이터를 넘겼습니다. 간단한 어플리케이션에서는 문제가 없지만, 복잡한 어플리케이션을 만듦에 있어서 controller 클래스 내의 LINQ to SQL 로 바로 작업하는 것은 문제가 생길 수 있습니다.

Controller 클래스에서 LINQ to SQL을 사용하는 것은 차후 데이터 액세스 기술을 변동할 때 어렵습니다. 예를 들어, Microsoft Entity Framework 로 데이터 액세스 기술을 바꾼다고 할 때, 데이터베이스와 연결된 모든 controller을 다시 작성해야 하는 일이 생깁니다.

앞으로의 변화에 더 유연한 MVC 어플리케이션을 만들기 위해서, Repository 패턴을 생각해 볼 수 있습니다. Repository 패턴을 사용할 때, 데이터베이스 액세스 로직을 구성하는 분할된 repository 클래스를 만듭니다.

Repository 클래스에서 사용되는 모든 메소드를 나타내는 인터페이스를 만들어야 합니다. controller에서는 repository가 아닌 인터페이스에 맞는 코드를 작성해야 합니다. 이와 같은 방식으로, repository을 구현할 수 있습니다.

아래 코드에서 IMovieRepository 인터페이스는 IList<Movie>을 리턴하는 하나의 ListAll() 메소드를 가지고 있습니다.

|  |
| --- |
| using System.Collections.Generic;  namespace MvcApplication1.Models  {  public interface IMovieRepository  {  IList<Movie> ListAll();  }  } |

아래 코드에서 repository 클래스는 IMovieRepository 인터페이스를 구현하였습니다. IMovieRepository 인터페이스에 정의된 ListAll 메소드가 구현된 것을 주의하세요.

|  |
| --- |
| using System.Collections.Generic;  using System.Linq;  namespace MvcApplication1.Models  {  public class MovieRepository : IMovieRepository  {  private MovieDataContext \_dataContext;  public MovieRepository()  {  \_dataContext = new MovieDataContext();  }  #region IMovieRepository Members  public IList<Movie> ListAll()  {  var movies = from m in \_dataContext.Movies  select m;  return movies.ToList();  }  #endregion  }  } |

마지막으로, 아래 코드의 MoviesController 클래스는 Repository 패턴을 사용하고 있습니다.

|  |
| --- |
| using System.Web.Mvc;  using MvcApplication1.Models;  namespace MvcApplication1.Controllers  {  public class MoviesController : Controller  {  private IMovieRepository \_repository;  public MoviesController() : this(new MovieRepository())  {  }  public MoviesController(IMovieRepository repository)  {  \_repository = repository;  }  public ActionResult Index()  {  return View(\_repository.ListAll());  }  }  } |

위 코드에서 MoviesController은 두 가지 생성자를 가지고 있는 것을 주의해야 합니다. 첫 번째 생성자는, 파라메터가 없는 생성자로서, 어플리케이션이 동작할 때, 호출됩니다. 이 생성자는 MovieRepository 클래스의 개체를 만들고 그것을 두 번째 생성자로 전달합니다.

두 번째 생성자는 IMovieRepository 파라메터 하나를 매개변수로 가지고 있습니다. 이 생성자는 \_repository 라는 클래스 단위 필드에 파라메터 값으로 지정합니다.

MoviesController 클래스는 Dependency Injection pattern 이라 불리는 디자인 패턴입니다. 특히, Contructor Dependency Injection 이라 불리는 것을 사용하는 것입니다. 다음 링크에서 이 패턴에 관한 자세한 글을 확인 할 수 있습니다.

<http://martinfowler.com/articles/injection.html>

MoviesController 클래스의 모든 코드는 실제 MovieRepository 클래스가 아니라 IMovieRepository 인터페이스와 작용합니다. 코드는 구체적인 인터페이스의 구현이 아니라 추상인터페이스 구현입니다.

데이터 액세스 기술을 수정하기를 원한다면 다른 기술을 사용하는 클래스 내의 IMovieRepository 인터페이스를 구현함으로써 적용할 수 있습니다. 예를 들어, EntityFrameworkMovieRepository 클래스나 SubSonicMovieRepository 클래스를 만들 수 있습니다. Controller 클래스가 interface에 맞게 구현하였기에, 새로 구현된 IMovieRepository을 controller 클래스에 전달하여 계속 작업할 수 있습니다.