**CREATE,READ,UPDATE,DELETE OPERATION USING ASP.NET MVC AND ENTITY FRAMEWORK**

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://1.bp.blogspot.com/-jS-gEcHugIk/VKguRbyCyzI/AAAAAAAAf44/RuREzKxRJwg/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework.gif) |
| Click on image to enlarge |

**Introduction**:  In this article I am going to explain what is entity framework (EF) and object relational mapping (ORM) and also example on How to perform simple Create, Retrieve, Update and Delete (CRUD) operations in ASP.NET MVC 4 using Entity Framework 5, razor view engine, scaffolding and Sql Server database.(Database first approach)

In previous articles I explained [What is Asp.Net MVC, Its work flow,goals and advantages](http://www.webcodeexpert.com/2013/11/what-is-aspnet-mvc-its-work-flowgoals.html) and [Difference between Asp.Net Webforms and Asp.Net MVC](http://www.webcodeexpert.com/2013/11/what-is-difference-between-aspnet.html) and [Use of ViewBag, ViewData and TempData in MVC](http://www.webcodeexpert.com/2013/12/what-is-use-of-viewbag-viewdata-and_3.html) and [Pass data from controller to view in MVC with example](http://www.webcodeexpert.com/2013/12/how-to-pass-data-from-controller-to.html) and [MVC application to Create,Read,Update,Delete and Search functionality using Razor view engine and Entity Framework](http://www.webcodeexpert.com/2013/11/aspnet-mvc-application-to.html)

Before start we must be familiar with the terms Entity Framework and ORM(Object Relational Mapping).

What is Entity Framework?

Microsoft Entity Framework (EF) is an object-relational mapping (ORM) framework that enables developers to work with relational data using domain-specific objects. Or in simple words we  can say it is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.

It enables developers to deal with data as objects and properties. It eliminates the need for most of the data-access code that developers usually need to write.

Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects.

It provides the services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.

What is O/RM(Object Relational Mapping)?

We perform different kind of operations on data like CRUD (Create, Retrieve, Update, and Delete) operations when working with database driven application. We might be using high level languages which are mostly object oriented like C#, VB.NET or Java, python etc. On the other hand, our databases are mostly relational in nature like MS SQL Server and Oracle. That means we are working with two different and incompatible type systems (i.e. Object Oriented Programming Languages with Relational Databases).

So an ORM (Object Relational Mapping) framework provides mapping between these two incompatible systems. It enables us to interact with data in an object oriented way.

ORM is used for storing data from domain objects to relational database like MS SQL Server and retrieve them in an automated way. It uses metadata information to interact with database so our data-layer code doesn’t know about the database structure. This makes the application maintainable and extendable. Thus O/RM tool becomes middleware that completely hides the complexity.

ORM specifies how a class and its properties are related to one or more tables in the database which is used by the O/RM tool’s engine to dynamically build SQL code that retrieves data and transforms it into objects.

**Implementation:** Let’s create a sample MVC Book application to perform basic CRUD operation.

But first of all create a Sql Server Database and name it "DbBooks" and in this database create a table and name it "tbBooks" .

You can create the above mentioned Database, Table and insert data in table by pasting the following script in sql server query editor:

CREATE DATABASE DbBooks

GO

USE DbBooks

GO

CREATE TABLE tbBooks

(

                BookId INT PRIMARY KEY IDENTITY(1,1) NOT NULL,

                BookName  VARCHAR(100),

                Author    VARCHAR(100),

                Publisher VARCHAR(150),

                Price     DECIMAL(10,2)

)

GO

INSERT INTO tbBooks (BookName,Author,Publisher,Price) VALUES ('Learn MVC','Lalit','Lalit Publications',1600.00)

INSERT INTO tbBooks (BookName,Author,Publisher,Price) VALUES ('Learn ASP.NET','Neha','Neha Publications',1200.00)

INSERT INTO tbBooks (BookName,Author,Publisher,Price) VALUES ('Learn SQL','Shaurya','Shaurya Publications',1150.00)

INSERT INTO tbBooks (BookName,Author,Publisher,Price) VALUES ('Learn jquery','John','John Publications',1600.00)

INSERT INTO tbBooks (BookName,Author,Publisher,Price) VALUES ('Learn Javascript','Scott','Scott Publications',1600.00)

 Now we have database and a table with some dummy data. It’s time to create MVC application step by step demonstrating CRUD operations.

**Step 1**: Open Visual Studio 12 or greater. I am using Visual studio 12 for this applications.  File Menu -> New project.

Select you preferred language either Visual C# or Visual Basic. For this tutorial we will use Visual C#.

Name the project “MVC\_BooksApplication”.  Specify the location where you want to save this project as shown in image below. Click on OK button.

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://2.bp.blogspot.com/-LHpFCqt9LxQ/VKgrJQ3qGuI/AAAAAAAAf3k/-OmsFNFuFDw/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-1.jpg) |
| Click on image to enlarge |

A "New ASP.NET MVC 4" project dialog box having various Project Templates will open. Select Internet template from available templates. Select Razor as view engine as shown in image below. Click on Ok button.

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://3.bp.blogspot.com/-L1sYB_HV3Mc/VKgrYax2DYI/AAAAAAAAf3s/qHjlNljtFf0/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-2.jpg) |
| Click on image to enlarge |

**Step 2**: It will add required files and folder automatically in the solution .So a default running MVC applications is ready. You can run the application using F5 or from the Debug menu of Visual Studio select Start Debugging.

As we are using default template, you can see the links to the Home, About and Contact pages as shown in image below. All are functional.

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://4.bp.blogspot.com/-W3s0kHL3n7c/VKgrfykW8AI/AAAAAAAAf30/TS99LkCn468/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-2.1.jpg) |
| Click on image to enlarge |

But our aim is to create a web application to manage Book records

 Now to move further in creating CRUD operation using Entity framework we need Entity framework installed. If it is not installed on your system then add entity framework package by right clicking the project name "MVC\_BooksApplication" in solution explorer, select manage nugget packages and search and install Entity Framework as shown in image below. (In my system it is already installed that is why it is showing uninstall button instead of install)

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://3.bp.blogspot.com/-y-QHNgcmClQ/VKgrrXEOvvI/AAAAAAAAf38/ZbIovNqlzqE/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-3.jpg) |
| Click on image to enlarge |

**Step 3**: Now right click on the project name "MVC\_BooksApplication" in solution explorer. Select Add New Item. Select ADO.NET Entity Data Model and name it "BooksDataModel.edmx" as shown in image below:

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://1.bp.blogspot.com/-yquJ61fejFE/VKgr0X49KTI/AAAAAAAAf4E/LaIDG6yiKDQ/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-4.jpg) |
| Click on image to enlarge |

**Step 4**: Select Generate Model from Database as shown in image below and click next:

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://4.bp.blogspot.com/-dIsQgIpulGk/VKgr7UAxm8I/AAAAAAAAf4M/hO3M2LFXWds/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-5.jpg) |
| Click on image to enlarge |

**Step 5**:  Select your database from server as shown in image below. It will automatically create connection string in web.config file with the name "DbBooksEntities" . You can also change the name of the connection string. But leave it as it is.

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://4.bp.blogspot.com/-PP4_7x58OPs/VKgsHQh-9CI/AAAAAAAAf4U/3JVFENeGJ90/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-6.jpg) |
| Click on image to enlarge |

**Step 6**:  Select Database object that you wish to add in the Model as shown in image below. In our case it is "tbBooks" Table. Leave the Model namespace "DbBooksModel" as it is.

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://3.bp.blogspot.com/-3nQI3w1TUiw/VKgsOmCCESI/AAAAAAAAf4c/9p-CpLMDBL4/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-7.jpg) |
| Click on image to enlarge |

**Step 7**: EDMX and Model files are added in the solution as shown in image below (HIGHLIGHT):

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://2.bp.blogspot.com/-IvVmFY5k9VM/VKgsWRslGuI/AAAAAAAAf4k/beZvjYNNwtY/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-8.png) |
| Click on image to enlarge |

Now build the solution. Go to Build Menu-> Build Solution.

**Step 8**: Now add a new controller by right clicking on Controller folder in solution explorer -> Add -> Controller and name it BookController.

|  |
| --- |
| [Create, Read, Update, Delete operation using MVC and Entity Framework](https://4.bp.blogspot.com/-h8H0alYa2R0/VKgsdjENbVI/AAAAAAAAf4s/dkxugtIhCV8/s1600/Create%2CRead%2CUpdate%2CDelete%2Boperation%2Busing%2BMVC%2Band%2BEntity%2BFramework%2BPic-9.jpg) |
| Click on image to enlarge |

1. Name the controller as **BookController**.
2. From Scaffolding Options, select “**MVC controller with read/write actions and views, using Entity Framework**”.
3. Select Model class as **tbBooks(MVC\_BookApplication)**
4. Select Data context class as **DbBooksEntities (MVC\_BookApplication)**.
5. Select Razor as rendering engine for views.
6. Also Click Advanced Options, select layout or master page and then expand Views folder and select \_Layout.cshtml from the shared folder.

**Note:** Model class and Data Context class will not appear until you build the solution

Visual studio will automatically create the following files and folder under your project.

1. A **BookController.cs** file in the **Controllers** folder of the project.
2. A **Book**folder in the **Views** folder of the project and in this folder it will create the views Create.cshtml, Delete.cshtml, Details.cshtml, Edit.cshtml, and Index.cshtml

**Congrats** you have created the MVC application having Create, Read, Update and Delete (CRUD) functionality that the Asp.Net MVC automatically created so the phenomenon of creating CRUD action methods and views automatically is called**Scaffolding**.

Our BookController has all the CRUD operation action created automatically as shown below:

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVC\_BooksApplication.Controllers

{

    public class BookController : Controller

    {

        private DbBooksEntities db = new DbBooksEntities();

        //

        // GET: /Book/

        public ActionResult Index()

        {

            return View(db.tbBooks.ToList());

        }

        //

        // GET: /Book/Details/5

        public ActionResult Details(int id = 0)

        {

            tbBook tbbook = db.tbBooks.Find(id);

            if (tbbook == null)

            {

                return HttpNotFound();

            }

            return View(tbbook);

        }

        //

        // GET: /Book/Create

        public ActionResult Create()

        {

            return View();

        }

        //

        // POST: /Book/Create

        [HttpPost]

        public ActionResult Create(tbBook tbbook)

        {

            if (ModelState.IsValid)

            {

                db.tbBooks.Add(tbbook);

                db.SaveChanges();

                return RedirectToAction("Index");

            }

            return View(tbbook);

        }

        //

        // GET: /Book/Edit/5

        public ActionResult Edit(int id = 0)

        {

            tbBook tbbook = db.tbBooks.Find(id);

            if (tbbook == null)

            {

                return HttpNotFound();

            }

            return View(tbbook);

        }

        //

        // POST: /Book/Edit/5

        [HttpPost]

        public ActionResult Edit(tbBook tbbook)

        {

            if (ModelState.IsValid)

            {

                db.Entry(tbbook).State = EntityState.Modified;

                db.SaveChanges();

                return RedirectToAction("Index");

            }

            return View(tbbook);

        }

        //

        // GET: /Book/Delete/5

        public ActionResult Delete(int id = 0)

        {

            tbBook tbbook = db.tbBooks.Find(id);

            if (tbbook == null)

            {

                return HttpNotFound();

            }

            return View(tbbook);

        }

        //

        // POST: /Book/Delete/5

        [HttpPost, ActionName("Delete")]

        public ActionResult DeleteConfirmed(int id)

        {

            tbBook tbbook = db.tbBooks.Find(id);

            db.tbBooks.Remove(tbbook);

            db.SaveChanges();

            return RedirectToAction("Index");

        }

        protected override void Dispose(bool disposing)

        {

            db.Dispose();

            base.Dispose(disposing);

        }

    }

}

**Step 9**:  As "BookId" is the auto incremented primary key of the "tbBooks" table. We don’t need to enter or display it on the pages. So open "Create.cshtml" and "Edit.cshtml" views and remove the following lines:

        <div class="editor-label">

            @Html.LabelFor(model => model.BookId)

        </div>

        <div class="editor-field">

            @Html.EditorFor(model => model.BookId)

            @Html.ValidationMessageFor(model => model.BookId)

        </div>

And add following line to hide the BookId:

          @Html.HiddenFor(model => model.BookId)

**Step 10** :  Open "Delete.cshtml" and "Details.cshtml" viewa and remove the following lines:

    <div class="display-label">

         @Html.DisplayNameFor(model => model.BookId)

    </div>

    <div class="display-field">

        @Html.DisplayFor(model => model.BookId)

    </div>

Also replace the following line in Details.cshtml

@Html.ActionLink("Edit", "Edit", new { /\* id=Model.PrimaryKey \*/ }) |

With

@Html.ActionLink("Edit", "Edit", new {  id=Model.BookId }) |

**Step 11**: Open "Index.cshtml" view and remove

     <th>

            @Html.DisplayNameFor(model => model.BookId)

    </th>

and

     <td>

            @Html.DisplayFor(modelItem => item.BookId)

    </td>

To edit, delete or view details of particular book we need BookId. Whenever we click on the Edit, Delete or Details link, particular BookId will be passed to the Edit, Delete and Details action mentioned in BookController. So replace the following three lines

 @Html.ActionLink("Edit", "Edit", new { /\* id=item.PrimaryKey \*/ }) |

 @Html.ActionLink("Details", "Details", new { /\* id=item.PrimaryKey \*/ }) |

 @Html.ActionLink("Delete", "Delete", new { /\* id=item.PrimaryKey \*/ })

With following three lines

@Html.ActionLink("Edit", "Edit", new {  id=item.BookId }) |

@Html.ActionLink("Details", "Details", new { id=item.BookId }) |

@Html.ActionLink("Delete", "Delete", new { id=item.BookId })

**Step 12**: Now Let’s set our BookController as the default controller and index method as the default action instead of default HomeController’s index method.

In App\_Start folder -> open RouteConfig.cs file:

Change the name of default controller from Home to Book in RegisterRoutes method

public static void RegisterRoutes(RouteCollection routes)

        {

            routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

            routes.MapRoute(

                name: "Default",

                url: "{controller}/{action}/{id}",

                defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

            );

        }

After changing from Home to our Book controller the RegisterRoutes method will become:

public static void RegisterRoutes(RouteCollection routes)

        {

            routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

            routes.MapRoute(

                name: "Default",

                url: "{controller}/{action}/{id}",

                defaults: new { controller = "Book", action = "Index", id = UrlParameter.Optional }

            );

        }

Now all is set. Just press F5 to run the application and play with the CRUD  operation.

***Now over to you:***

*"A blog is nothing without reader's feedback and comments. So please provide your valuable feedback so that i can make this blog better and If you like my work; you can appreciate by leaving your comments, hitting Facebook like button, following on Google+, Twitter, Linked in and Pinterest, stumbling my posts on stumble upon and subscribing for receiving free updates directly to your inbox . Stay tuned and stay connected for more technical updates."*