# Local DB CRUD Operations in MVC 4 using Entity Framework 5

Show how to perform basic CRUD operations in an MVC4 application with the help of Entity Framework 5 . EF and MVC had advanced themselves to the level that we don't have to put effort in doing extra work.

## 1) MVC

**Model**: The business entity on which the overall application operates. Many applications use a persistent storage mechanism (such as a database) to store data. MVC does not specifically mention the data access layer because it is understood to be encapsulated by the Model.

**View**: The user interface that renders the model into a form of interaction.

**Controller**: Handles a request from a view and updates the model that results a change in Model's state.

To implement MVC in .NET, we need mainly three classes (`View`, `Controller` and the `Model`).

## 2) Entity Framework

Let's have a look at the standard definition of Entity Framework given by Microsoft:

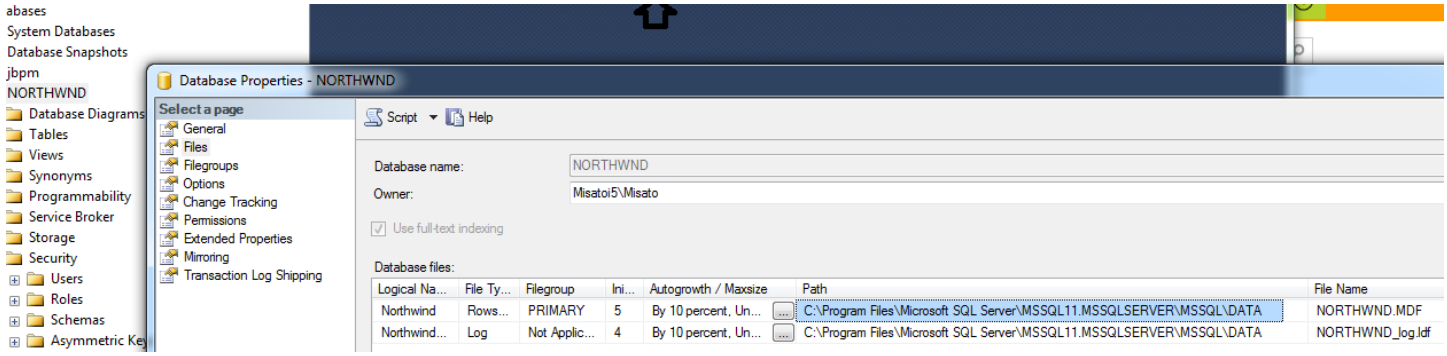
*"The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write. Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework's ORM implementation provides 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."*

In a simple language, Entity framework is an Object/Relational Mapping (ORM) framework. It is an enhancement to ADO.NET, an upper layer to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.

Hope this gives a glimpse of an ORM and EntityFramework.

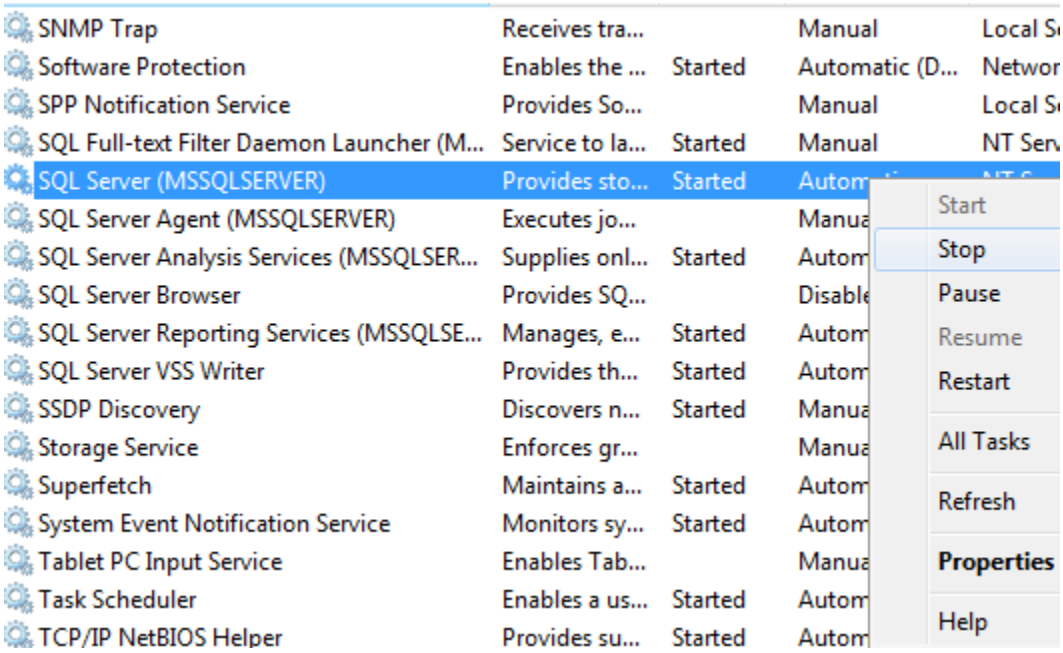
## 3) MVC Application

**Step 1**: Find the NorthWind.MDF file path.



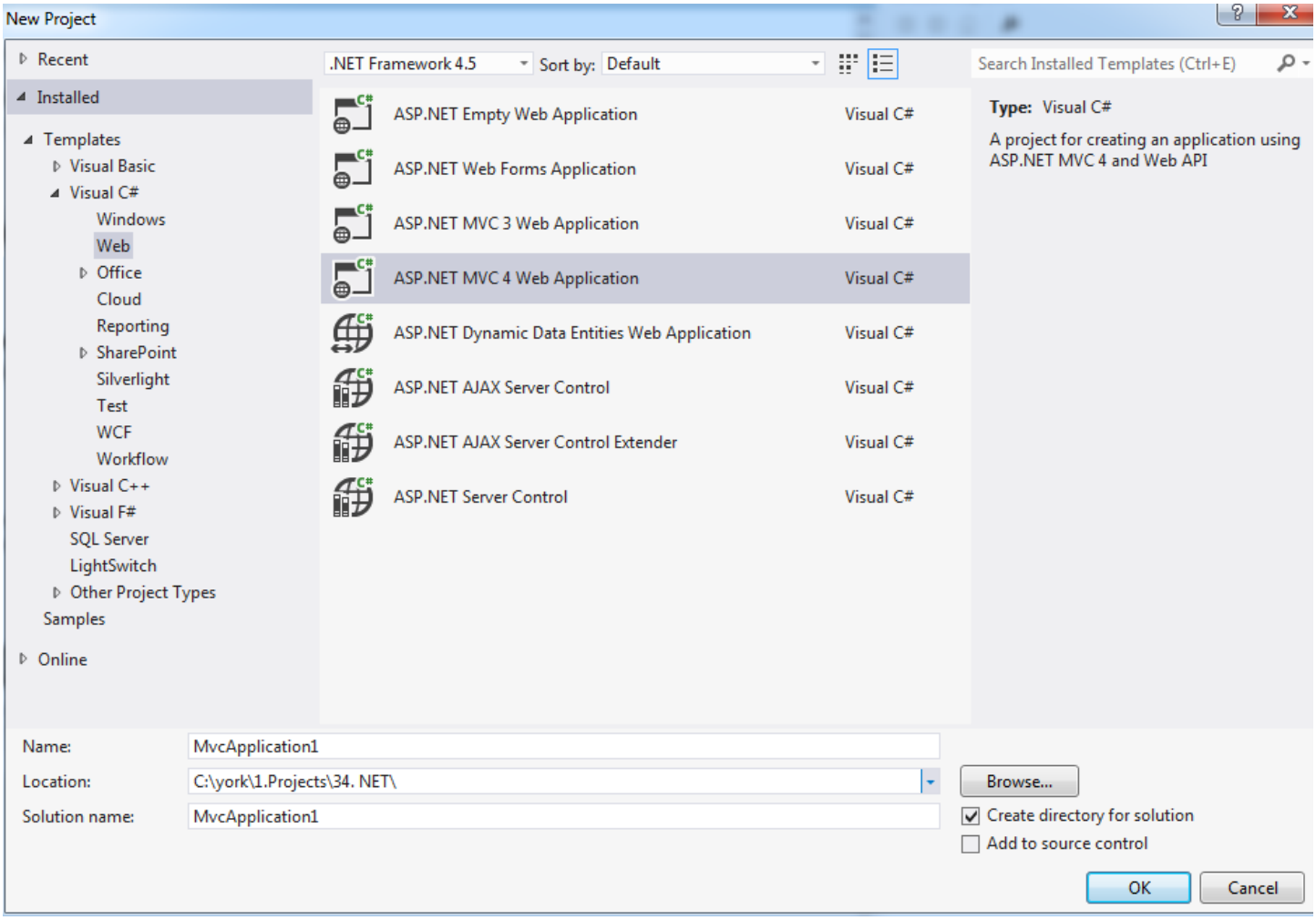

(1). Ctrl + C to cope the file path of NORTHWND.MDF

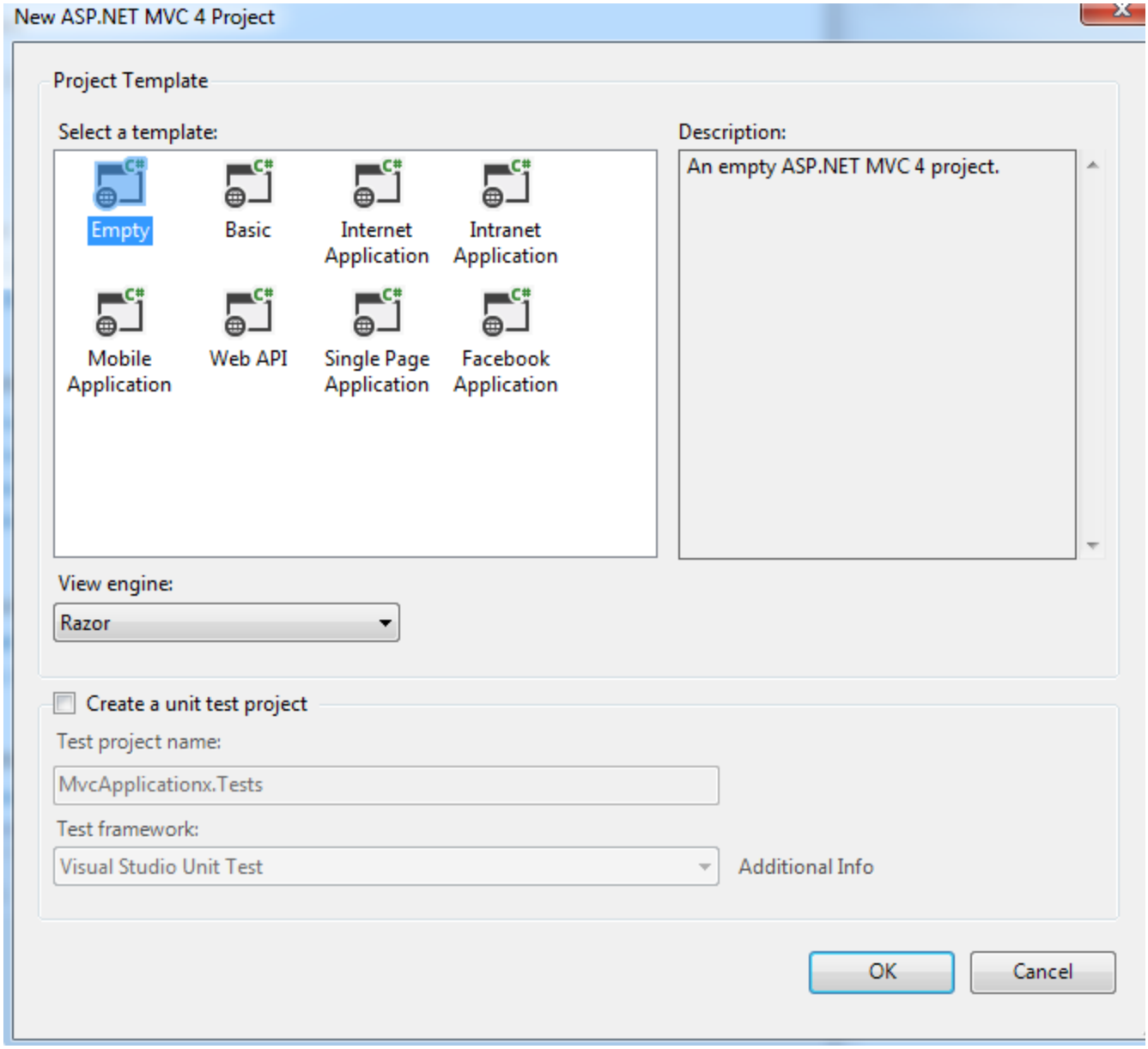
(2). Stop MSSQLSERVER service




(3). Copy NORTHWND.MDF to another file path

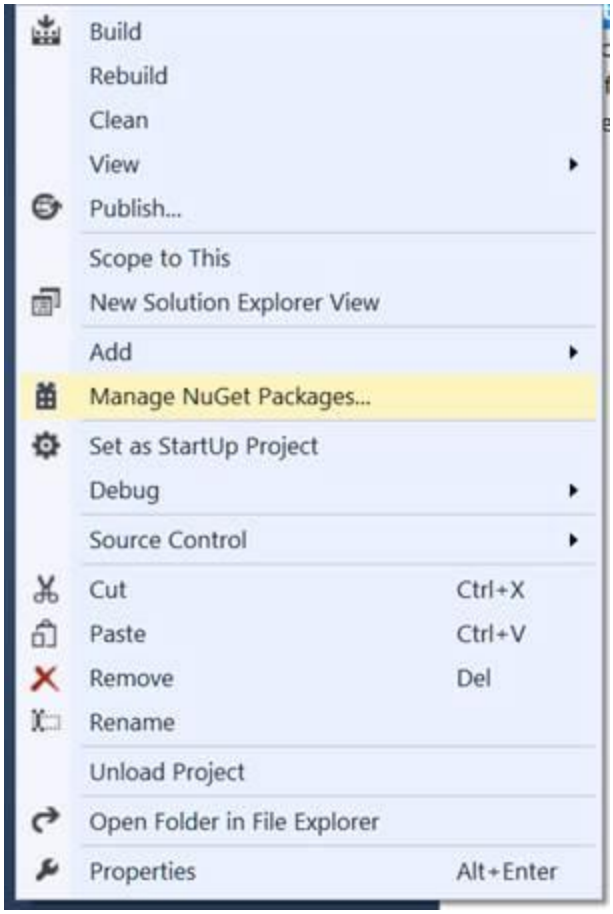
**Step 2**: Open your Visual Studio (Visual Studio Version should be greater than or equal to 12) and add an MVC Internet application.

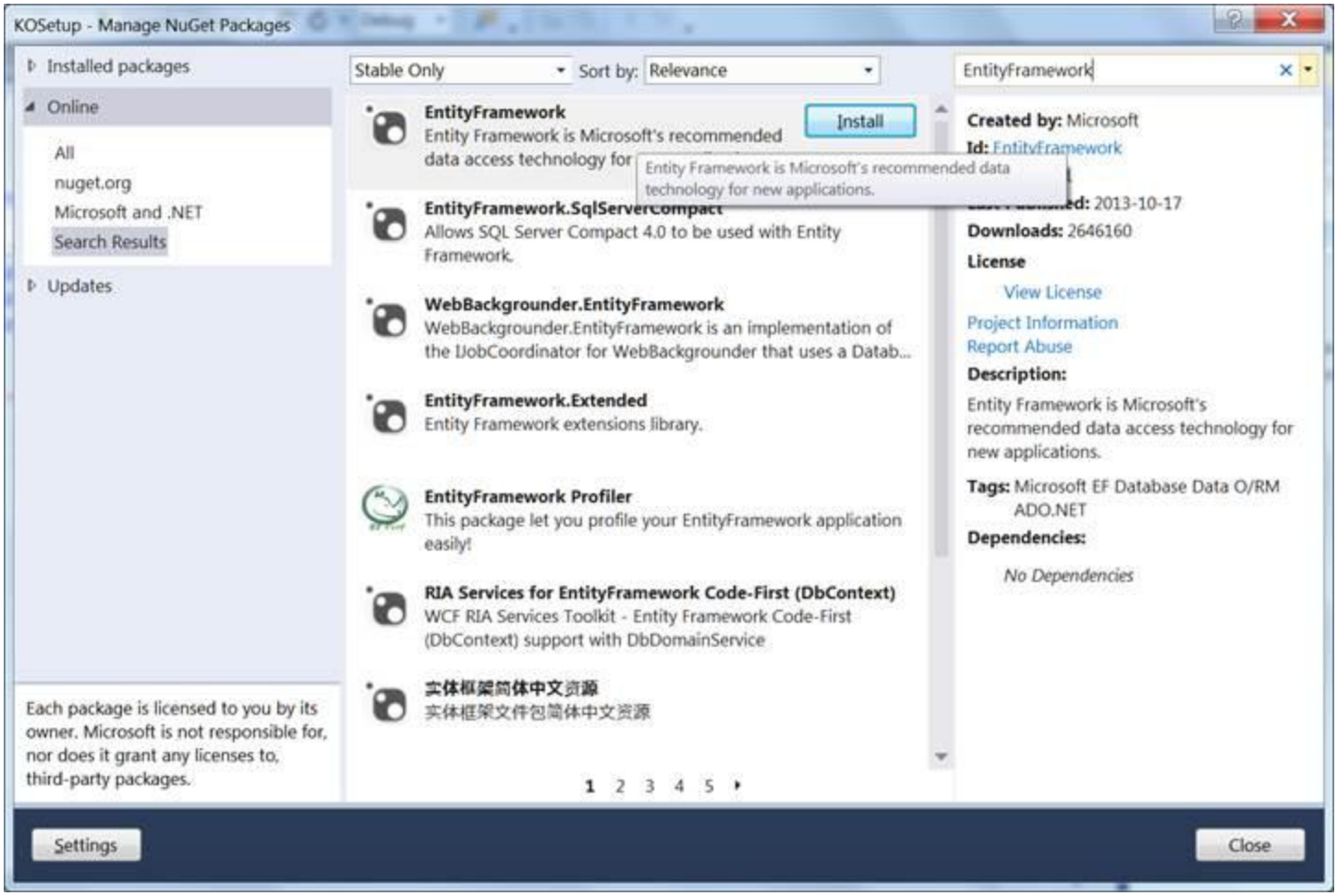

New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
SQL Server
LightSwitch
Other Project Types
Samples
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
ASP.NET Empty Web Application Visual C#
ASP.NET Web Forms Application Visual C#
ASP.NET MVC 3 Web Application Visual C#
ASP.NET MVC 4 Web Application Visual C#
ASP.NET Dynamic Data Entities Web Application Visual C#
ASP.NET AJAX Server Control Visual C#
ASP.NET AJAX Server Control Extender Visual C#
ASP.NET Server Control Visual C#
Type: Visual C#
A project for creating an application using ASP.NET MVC 4 and Web API
Name: MvcApplication1
Location: C:\york\1.Projects\34. NET\
Solution name: MvcApplication1
Browse...
Create directory for solution
Add to source control
OK
Cancel



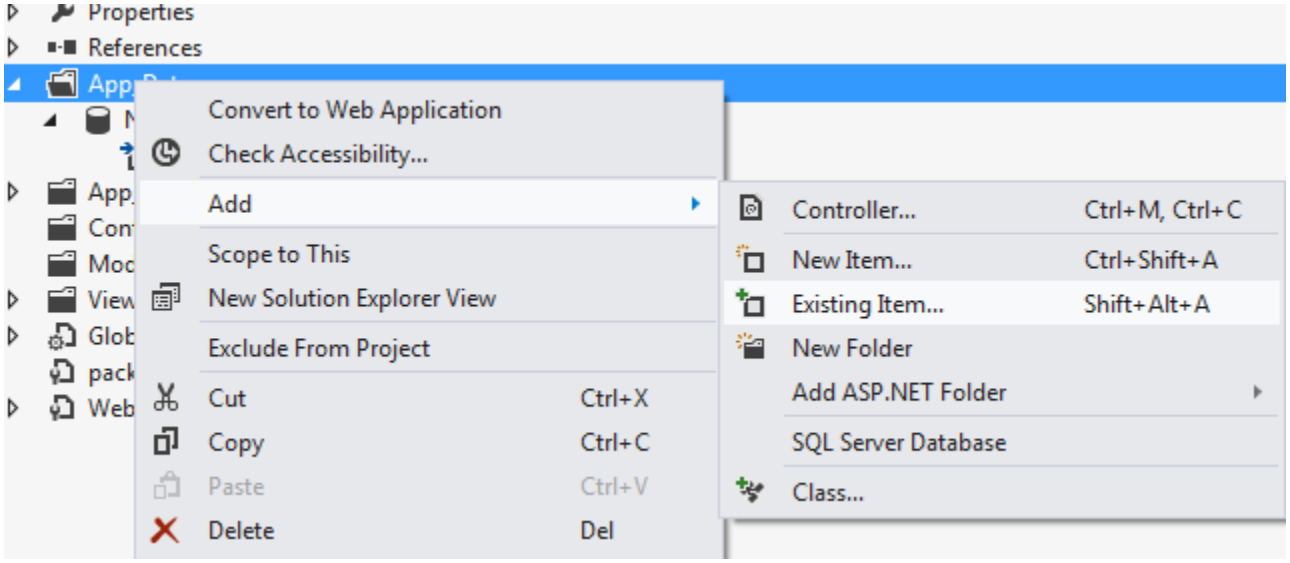

I have given it a name "`MvcApplication1`".

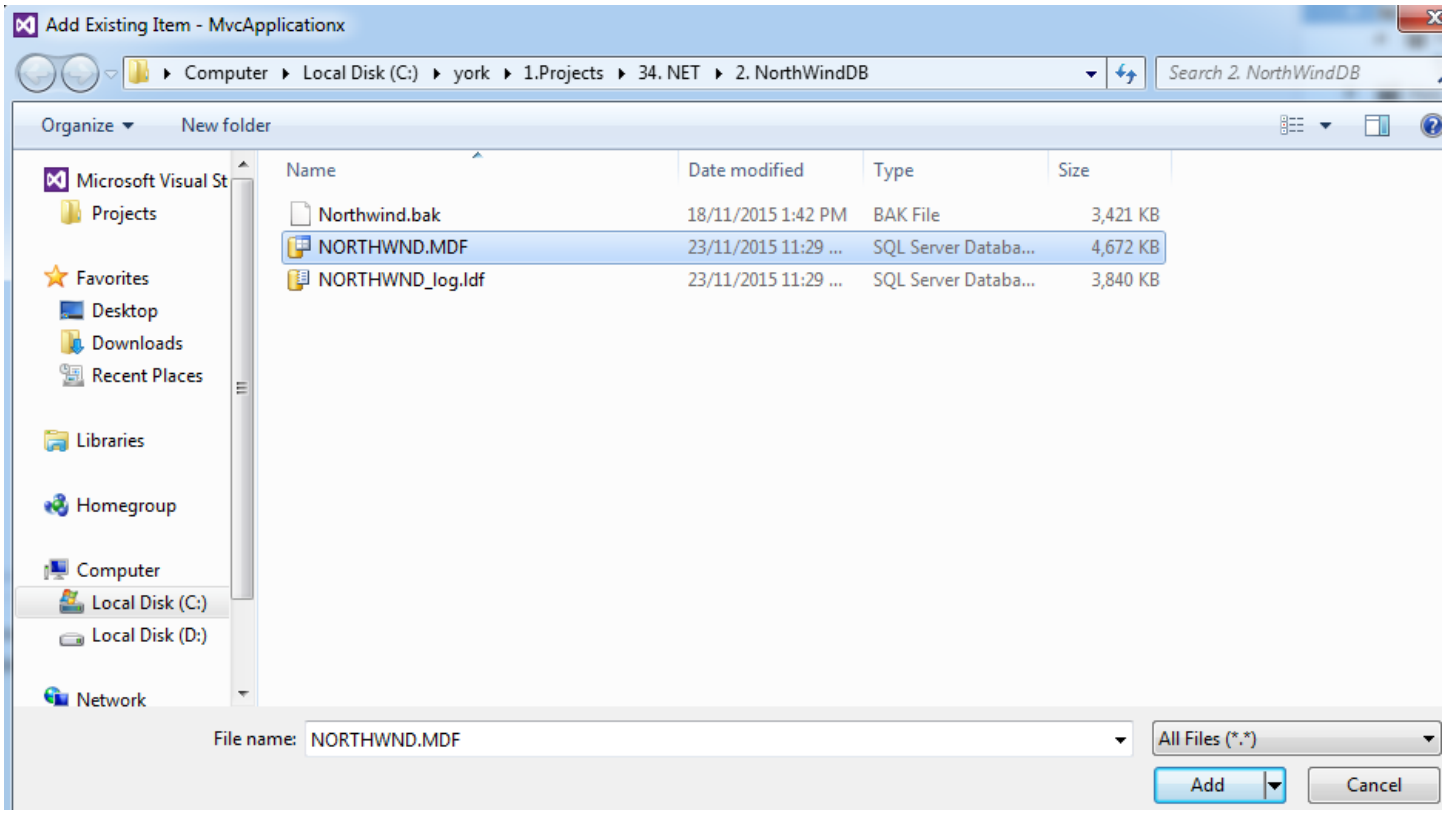
**Step 3**: You'll get a full structured MVC application with default Home controller in the *Controller* folder. By default, entity framework is downloaded as a package inside application folder but if not, you can add entity framework package by right clicking the project, select manage nugget packages and search and install Entity Framework.


Build
Rebuild
Clean
View
Publish...
Scope to This
New Solution Explorer View
Add
Manage NuGet Packages...
Set as StartUp Project
Debug
Source Control
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Open Folder in File Explorer
Properties Alt+Enter

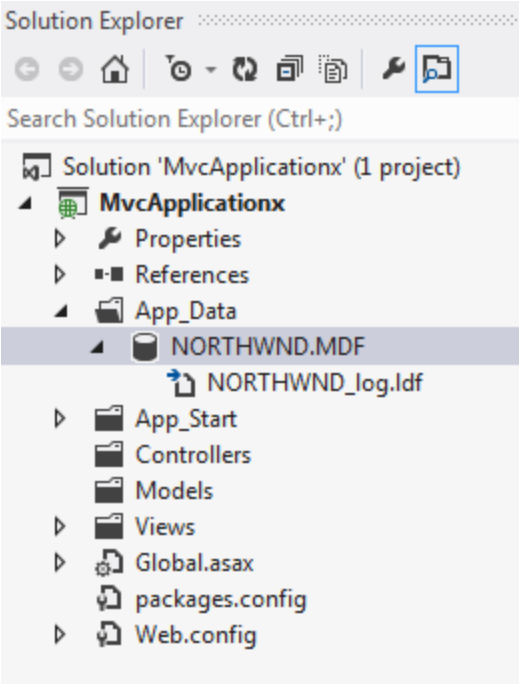

KOSetup - Manage NuGet Packages
Installed packages
Online
All
nuget.org
Microsoft and .NET
Search Results
Updates
Stable Only
Sort by: Relevance
EntityFramework
EntityFramework
Entity Framework is Microsoft's recommended data access technology for
Install
Entity Framework is Microsoft's recommended data technology for new applications.
EntityFramework.SqlServerCompact
Allows SQL Server Compact 4.0 to be used with Entity Framework.
WebBackgrounder.EntityFramework
WebBackgrounder.EntityFramework is an implementation of the IJobCoordinator for WebBackgrounder that uses a Datab...
EntityFramework.Extended
Entity Framework extensions library.
EntityFramework Profiler
This package let you profile your EntityFramework application easily!
RIA Services for EntityFramework Code-First (DbContext)
WCF RIA Services Toolkit - Entity Framework Code-First (DbContext) support with DbDomainService
实体框架简体中文资源
实体框架文件包简体中文资源
1 2 3 4 5
Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Created by: Microsoft
Id: EntityFramework
Downloads: 2646160
License
View License
Project Information
Report Abuse
Description:
Entity Framework is Microsoft's recommended data access technology for new applications.
Tags: Microsoft EF Database Data O/RM ADO.NET
Dependencies:
No Dependencies
Settings
Close

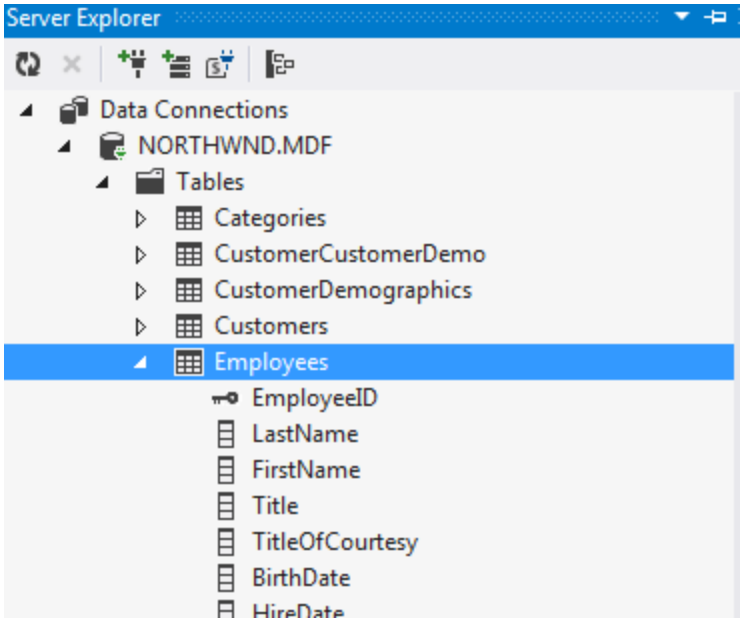
**Step 4**: Right click project App_Data folder, select add new item and add Existing Item, follow the steps in the wizard as shown below:



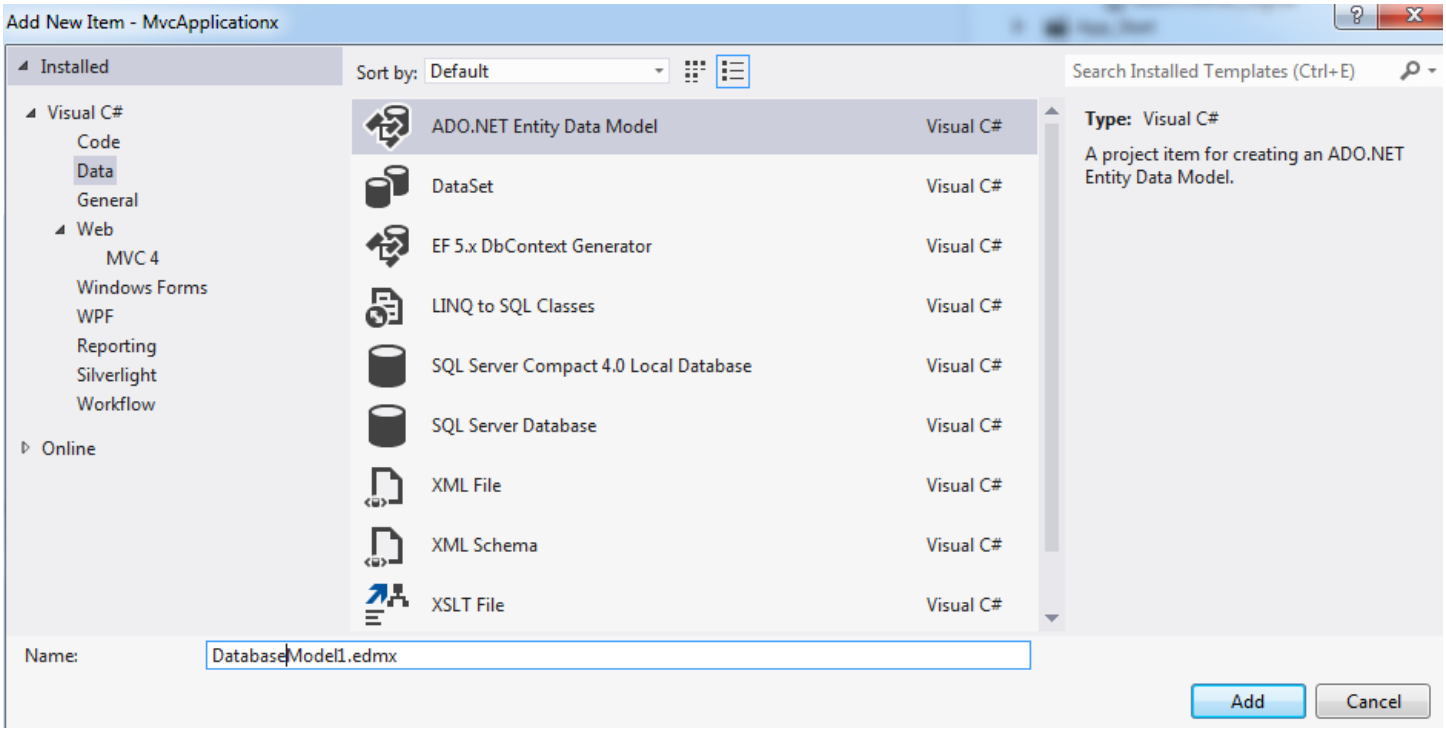



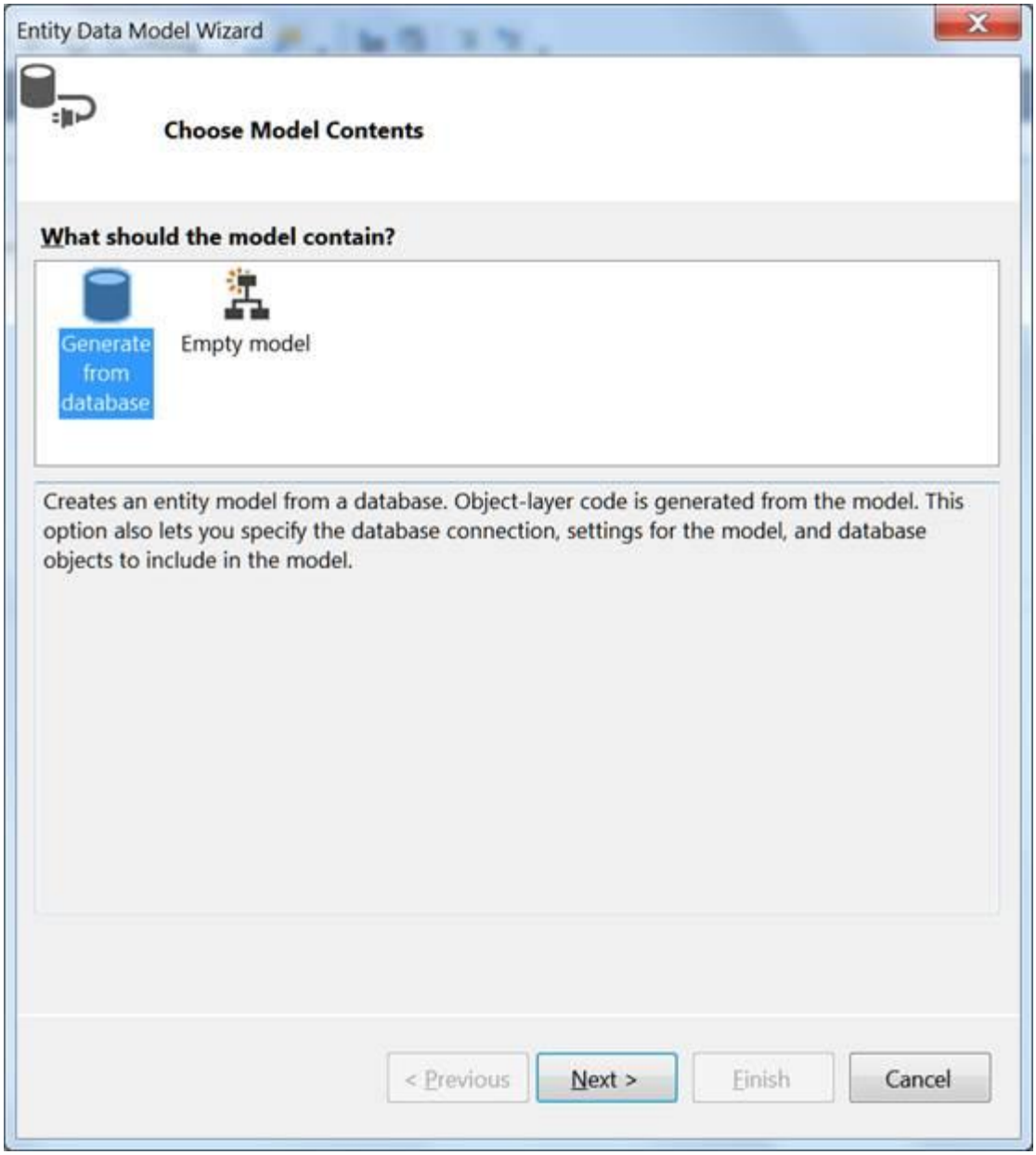

Double click NORTHWND.MDF in App_Data folder:

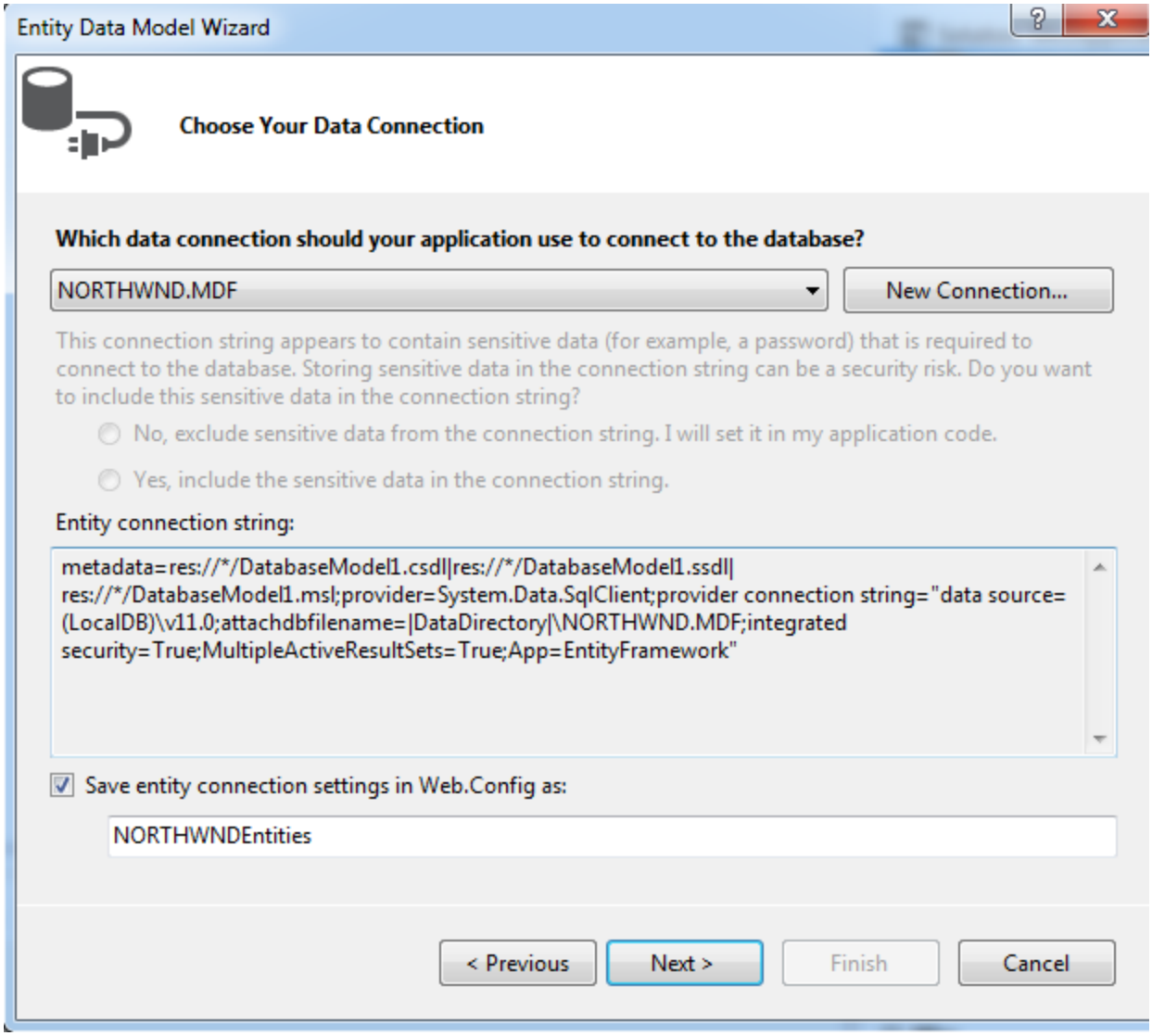


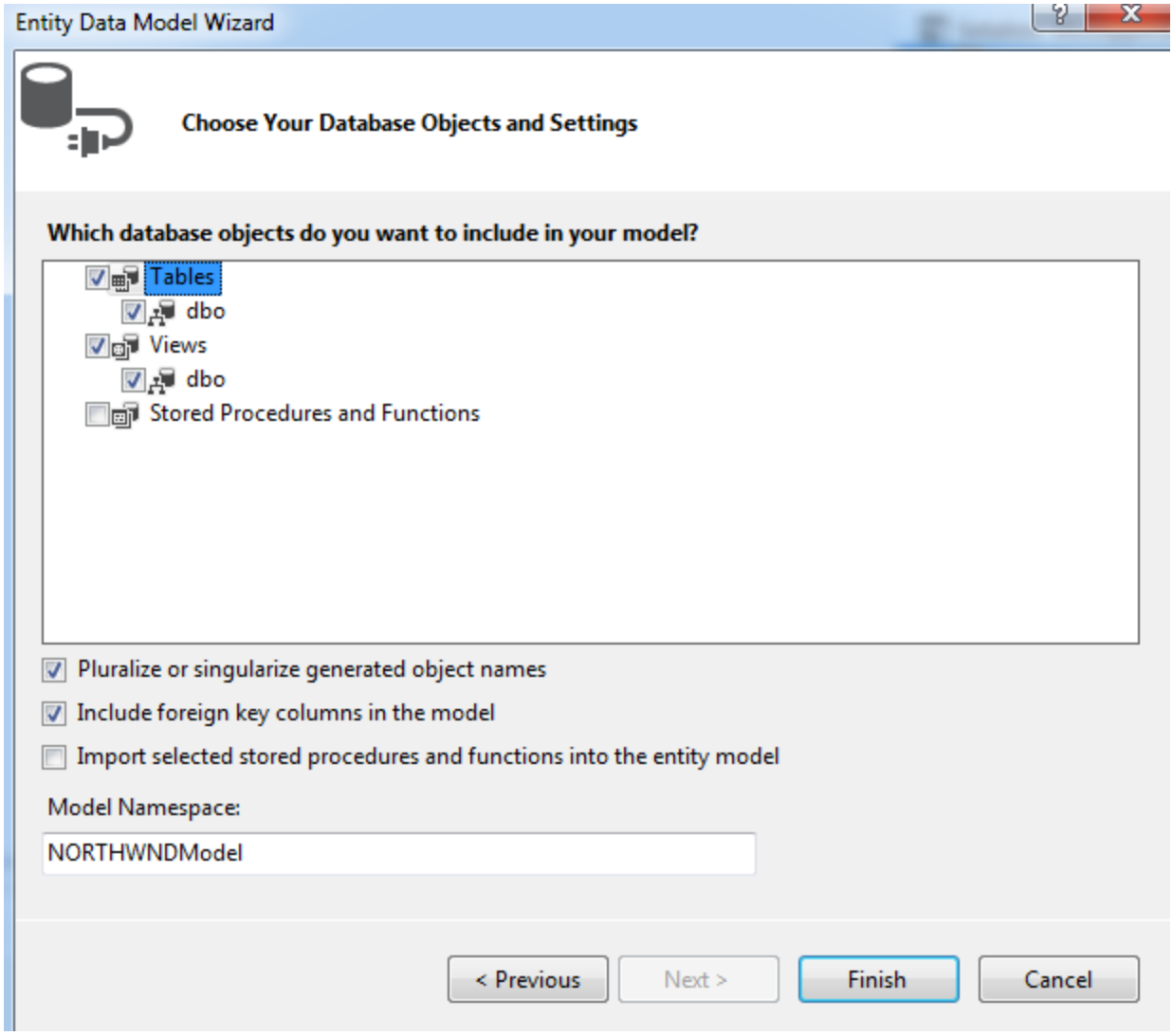

**Step 5**: Right click project file, select add new item and add ADO.NET entity data model, follow the steps in the wizard as shown below:


Add New Item - MvcApplicationx
Installed
Sort by: Default
Search Installed Templates (Ctrl+E)
Visual C#
Code
Data
General
Web
MVC 4
Windows Forms
WPF
Reporting
Silverlight
Workflow
Online
ADO.NET Entity Data Model Visual C#
DataSet Visual C#
EF 5.x DbContext Generator Visual C#
LINQ to SQL Classes Visual C#
SQL Server Compact 4.0 Local Database Visual C#
SQL Server Database Visual C#
XML File Visual C#
XML Schema Visual C#
XSLT File Visual C#
Type: Visual C#
A project item for creating an ADO.NET Entity Data Model.
Name: DatabaseModel1.edmx
Add
Cancel

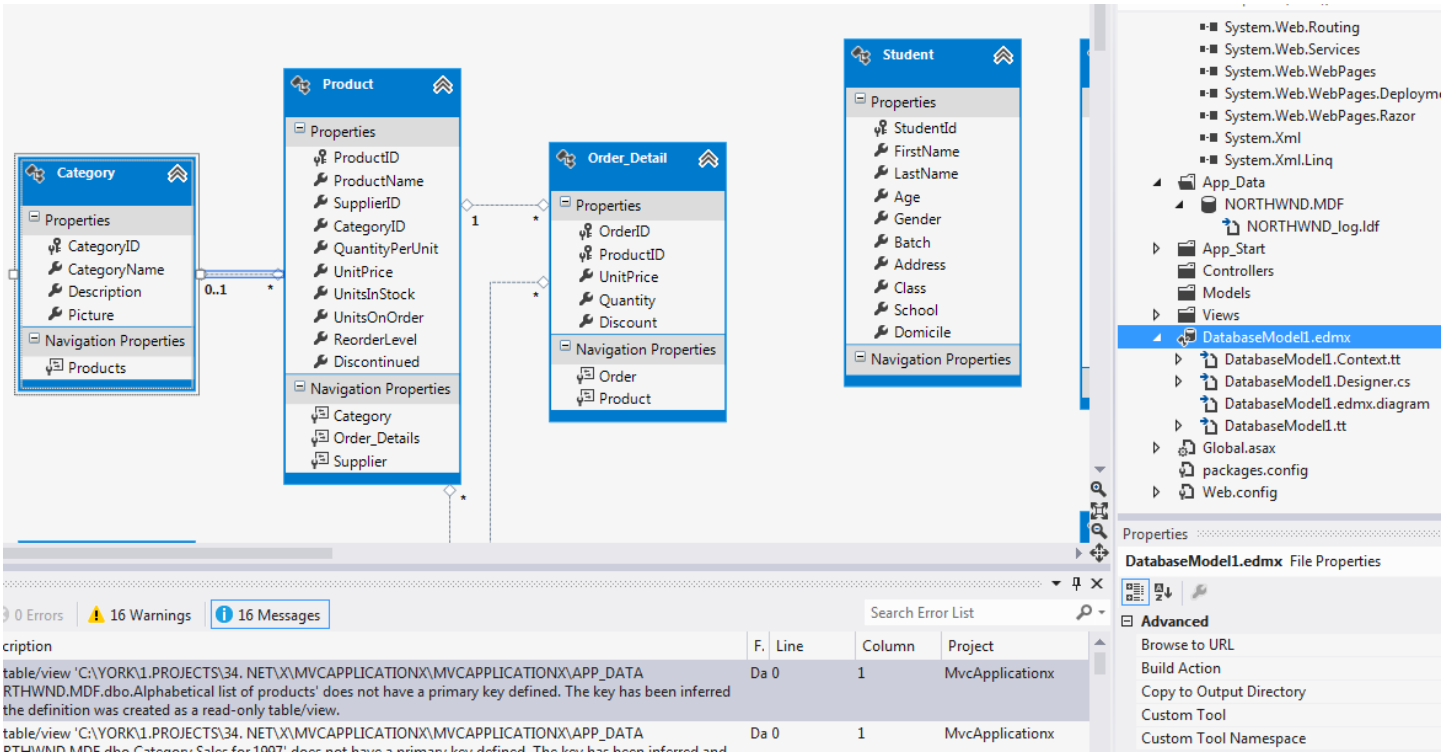

Entity Data Model Wizard
Choose Model Contents
What should the model contain?
Generate from database
Empty model
Creates an entity model from a database. Object-layer code is generated from the model. This option also lets you specify the database connection, settings for the model, and database objects to include in the model.
< Previous
Next >
Finish
Cancel




Generate model from NorthWnd.MDF database, select your server and NORTHWND database name, the connection string will automatically be added to your *Web.Config*, name that connection string as NORTHWND`Entities`.

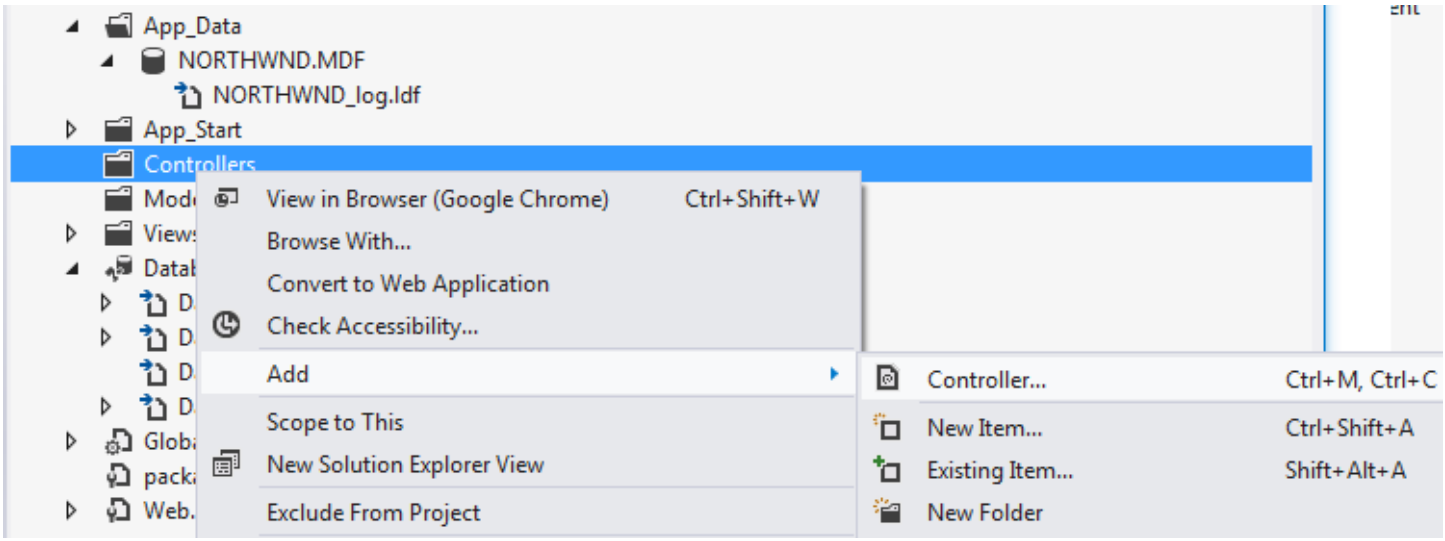


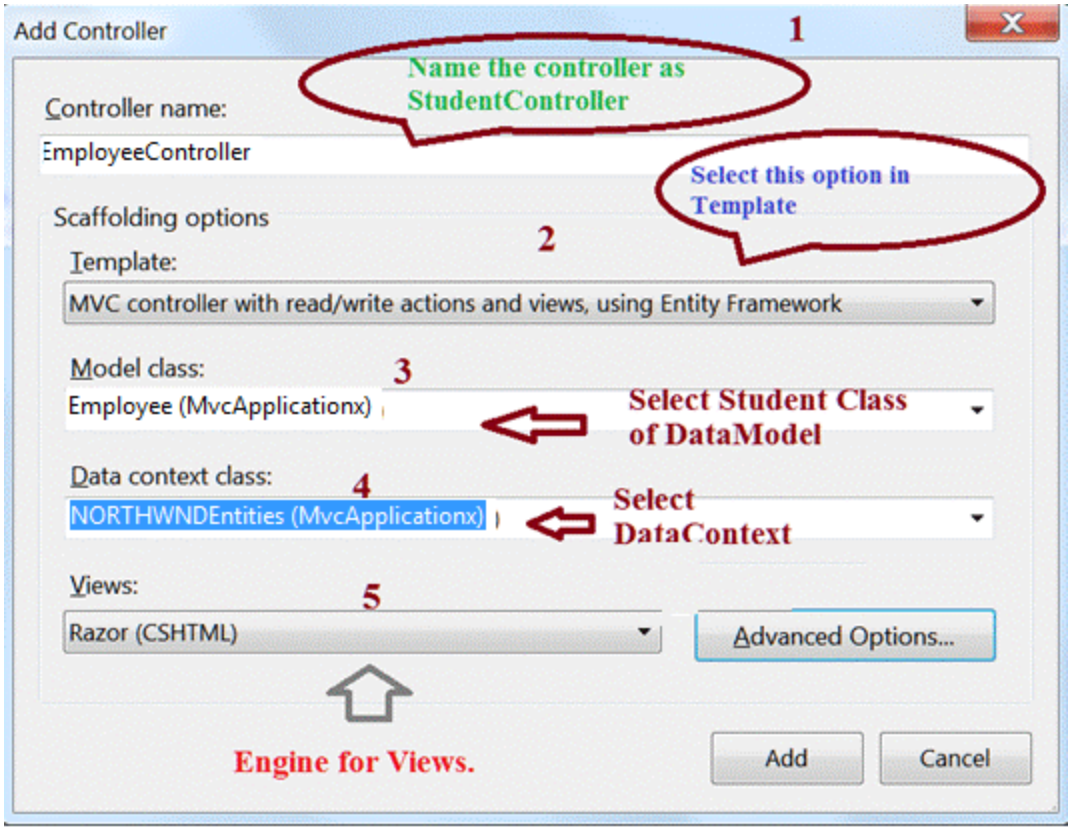

Select all tables and views to be added to the model.




**Step6**: Build the project. Make sure no error

**Step7**:Now add a new controller to the *Controller* folder, right click *controller* folder and add a controller named Employee. Since we have already created our `Datamodel`, we can choose for an option where CRUD actions are created by chosen Entity Framework `Datamodel`:






- Name your controller as Employee`Controller`.
- From Scaffolding Options, select "MVC controller with read/write actions and views, using Entity Framework".
- Select `Model` class as Employee, that lies in our solution.
- Select Data context class as NORTHWND`Entities` that is added to our solution when we added EF data model.
- Select Razor as rendering engine for views.

**Step 8**: We see our `student` controller prepared with all the CRUD operation actions 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 MvcApplication1.Controllers
{
    public class EmployeeController : Controller
    {
        private NORTHWNDEntities db = new NORTHWNDEntities();

        //
        // GET: /Employee/

        public ActionResult Index()
        {
            var employees = from e in db.Employees
                            where e.EmployeeID > -1
                            select e;
            //return View(db.Employees.ToList());
            return View(employees.ToArray());
```

```
        }

        //
        // GET: /Employee/Details/5

        public ActionResult Details(int employeeID = 0)
        {
            var employee = from e in db.Employees
                           where e.EmployeeID == employeeID
                           select e;

            //Employee employee = db.Employees.Find(employeeID);
            if (employee == null)
            {
                return HttpNotFound();
            }

            return View(employee.First());
        }

        //
        // GET: /Employee/Create

        public ActionResult Create()
        {
            return View();
        }

        //
        // POST: /Employee/Create

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create(Employee employee)
        {
            if (ModelState.IsValid)
            {
                db.Employees.Add(employee);
                db.SaveChanges();
                return RedirectToAction("Index");
            }

            return View(employee);
        }

        //
        // GET: /Employee/Edit/5

        public ActionResult Edit(int employeeID = 0)
        {
            var employee = from e in db.Employees
                           where e.EmployeeID == employeeID
                           select e;
            //Employee employee = db.Employees.Find(employeeID);
            if (employee == null)
            {
                return HttpNotFound();
            }
            return View(employee.First());
        }

        //
        // POST: /Employee/Edit/5

        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        public ActionResult Edit(Employee employee)
        {
            if (ModelState.IsValid)
            {
                db.Entry(employee).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(employee);
        }

        //
        // GET: /Employee/Delete/5

        public ActionResult Delete(int employeeID = 0)
        {
            Employee employee = db.Employees.Find(employeeID);
            if (employee == null)
            {
                return HttpNotFound();
            }
            return View(employee);
        }

        //
        // POST: /Employee/Delete/5

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public ActionResult DeleteConfirmed(int employeeID)
        {
            Employee employee = db.Employees.Find(employeeID);
            db.Employees.Remove(employee);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        //
        // GET: /Employee/Test/5

        public ActionResult Test(string employeeID)
        {
            ViewBag.employeeID = employeeID;
            return View();
        }

        protected override void Dispose(bool disposing)
        {
            db.Dispose();
            base.Dispose(disposing);
        }
    }
}
```

**Step 9**: Open *App_Start* folder and, change `RouteConfig.cs as following:`

```
public static void RegisterRoutes(RouteCollection routes)
{
        public static void RegisterRoutes(RouteCollection routes)
        {
            //routes.Add(new Route("{resource}.axd/{*pathInfo}", new StopRoutingHandler()));
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            /*
            routes.MapRoute(
                name: "Default",
```

```
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        );
         */
        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{employeeID}",
            defaults: new { controller = "Employee", action = "Index", employeeID = UrlParameter.Optional
        );
      }
}
```

**Step 10**: Update the links

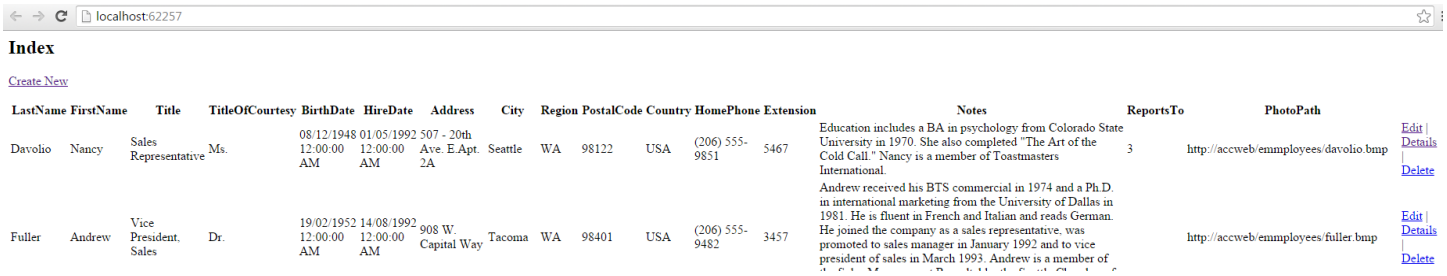
Index.CSHTML:

```
    <td>
        @Html.ActionLink("Edit", "Edit", new { employeeID=item.EmployeeID  }) |
        @Html.ActionLink("Details", "Details", new {employeeID=item.EmployeeID }) |
        @Html.ActionLink("Delete", "Delete", new { employeeID=item.EmployeeID })
    </td>
```

Edit.CSHTML:

```
@Html.ActionLink("Edit", "Edit", new { EmployeeID=Model.EmployeeID }) |
@Html.ActionLink("Back to List", "Index")
```

**Step 10**: Now press F5 to run the application, and you'll see the list of all students we added into table `Student`while creating it is displayed. Since the CRUD operations are automatically written, we have action results for display list and other Edit, Delete and Create operations. Note that views for all the operations are created in*Views* Folder under *Student* Folder name.




Now you can perform all the operations on this list.

Since I have not provided any validation checks on model or creating an existing student id, the code may break, so I am calling Edit Action in create when we find that id already exists.

Now create new employee.

Employee

LastName
York
FirstName
Chen
Title
Title1
TitleOfCourtesy
T1
BirthDate

HireDate

Address
address
City
Toronto
Region
Ontario
PostalCode
M2H3H5
Country
Canada
HomePhone
4167996678
Extension
123
Notes
note
ReportsTo
1
PhotoPath

Save

Back to List

We see that the employee is created successfully and added to the list.

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dodsworth | Anne | Sales Representative | Ms. | 27/01/1966 12:00:00 AM | 15/11/1994 12:00:00 AM | 7 | Houndstooth Rd. | London | | WG2 7LT | UK | (71) 555-4444 | 452 | Anne has a BA degree in English from St. Lawrence College. She is fluent in French and German. | 5 | http://accweb/emmployees/davolio.bmp | Edit \| Details \| Delete |
| York | Chen | Title1 | T1 | | | | address | Toronto | Ontario | M2H3H5 | Canada | 4167996678 | 123 | note | 1 | | Edit \| Details \| Delete |

You can expand the application by adding multiple Controllers, Models and Views.