

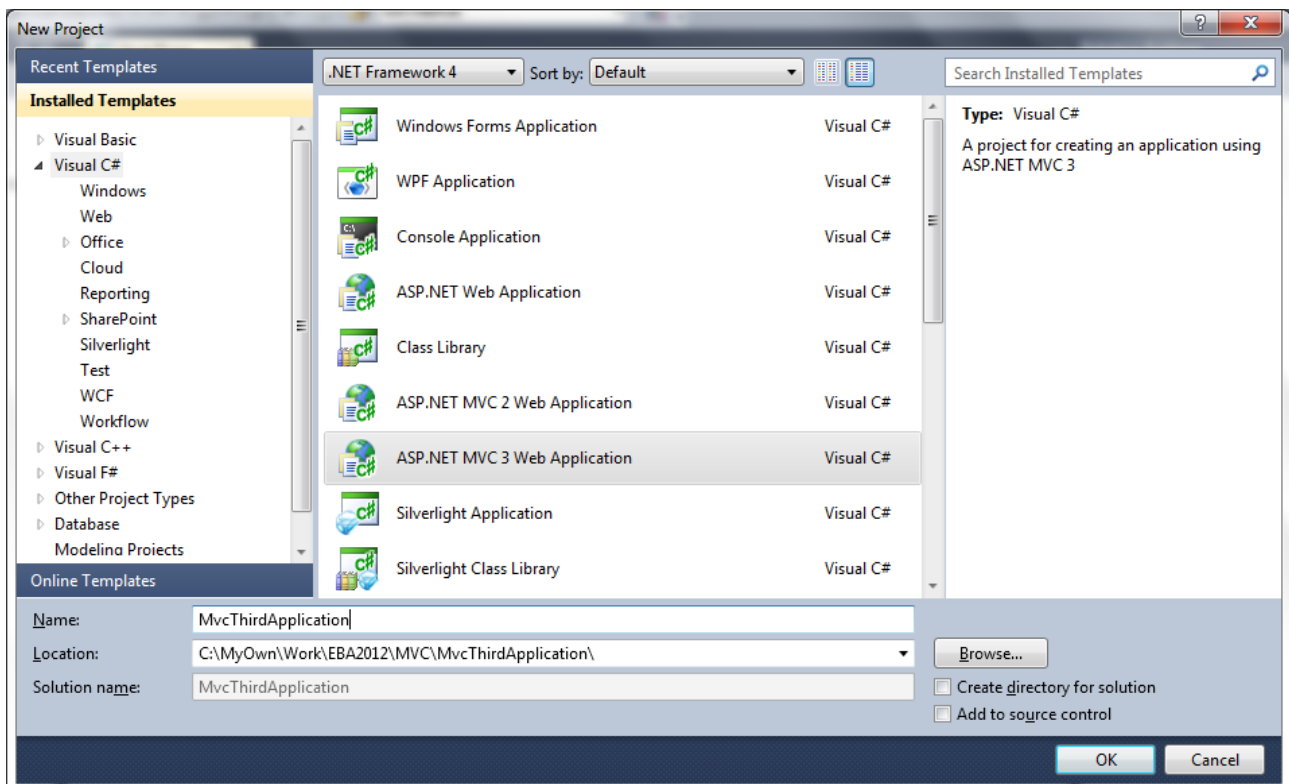
MVC 3 application with a Database and Entity Framework

ADO.NET Entity Framework gives the application developer three possibilities to define the communication between the program and the database:

1. When the **“Database first”** workflow is chosen, the developer will first create the database and then create the entity data model and entity classes from database.
2. When the **“Model first”** workflow is chosen, the developer will first create the entity data model and the database is generated from the model.
3. When the **“Code first”** workflow is chosen, the developer defines the entity classes and then attaches Entity Framework to the classes.

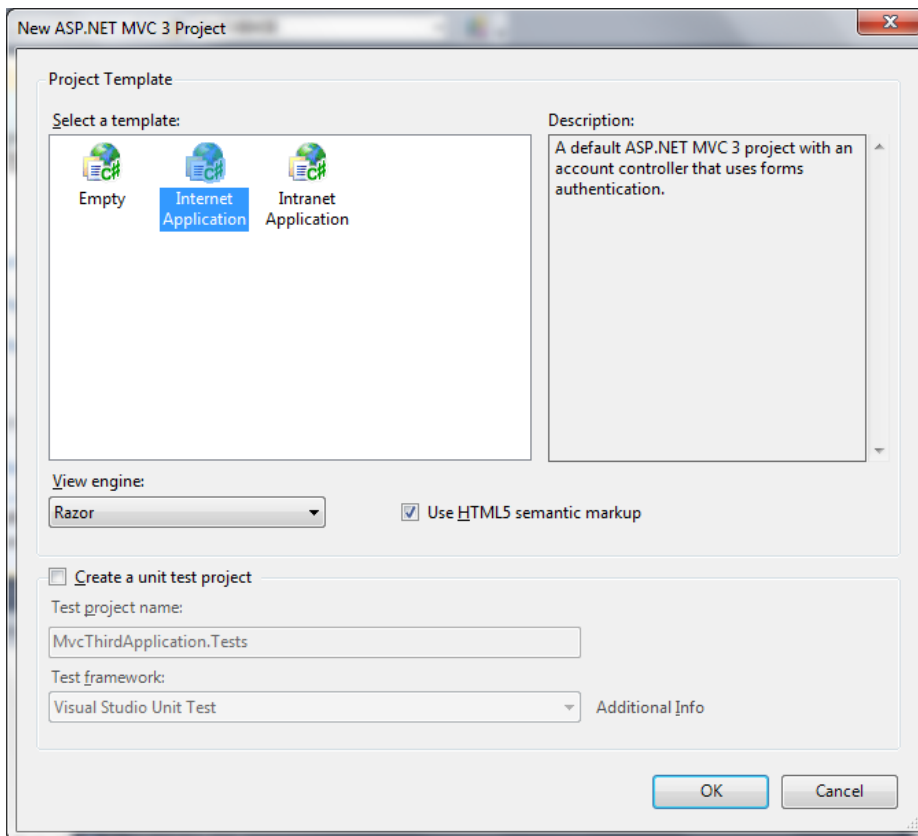
1. Using Database first workflow

MVC 3 application has to be created at first. This is the third MVC model application so it is called MvcThirdApplication. A new project is selected from File menu:



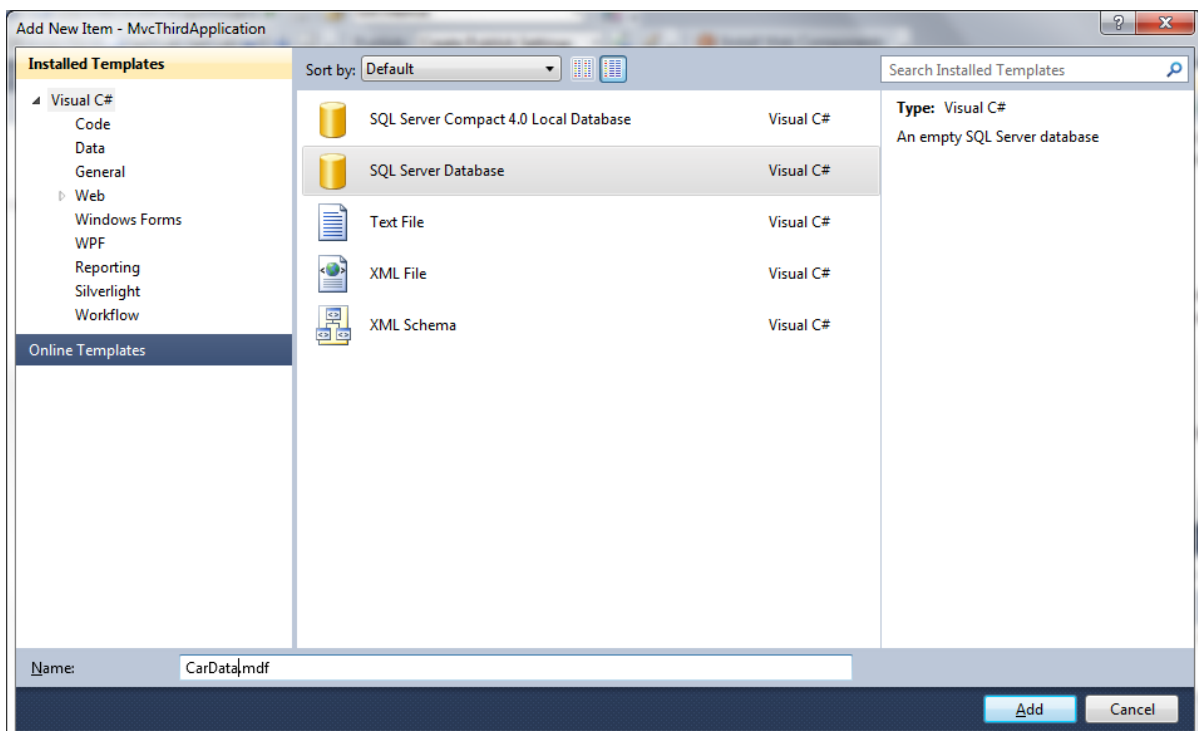
Picture: Create a new MVC 3 application.

When the ASP.NET MVC 3 Application, Location and Solution name are given it is time to select the template for the application. So far Empty template has been used but now Internet Application template is selected with Razor view engine and support for HTML 5.



Picture: Internet Application template is selected with Razor view engine and HTML 5 support

When application is created it is time to create the database. Right click App_Data –folder, select SQL Server Database template and give name **CarData** to the database.



Picture: SQL Server Database is selected

When the database has been created it is time to replace the database we created with the Data Management sample application CarDatabase:

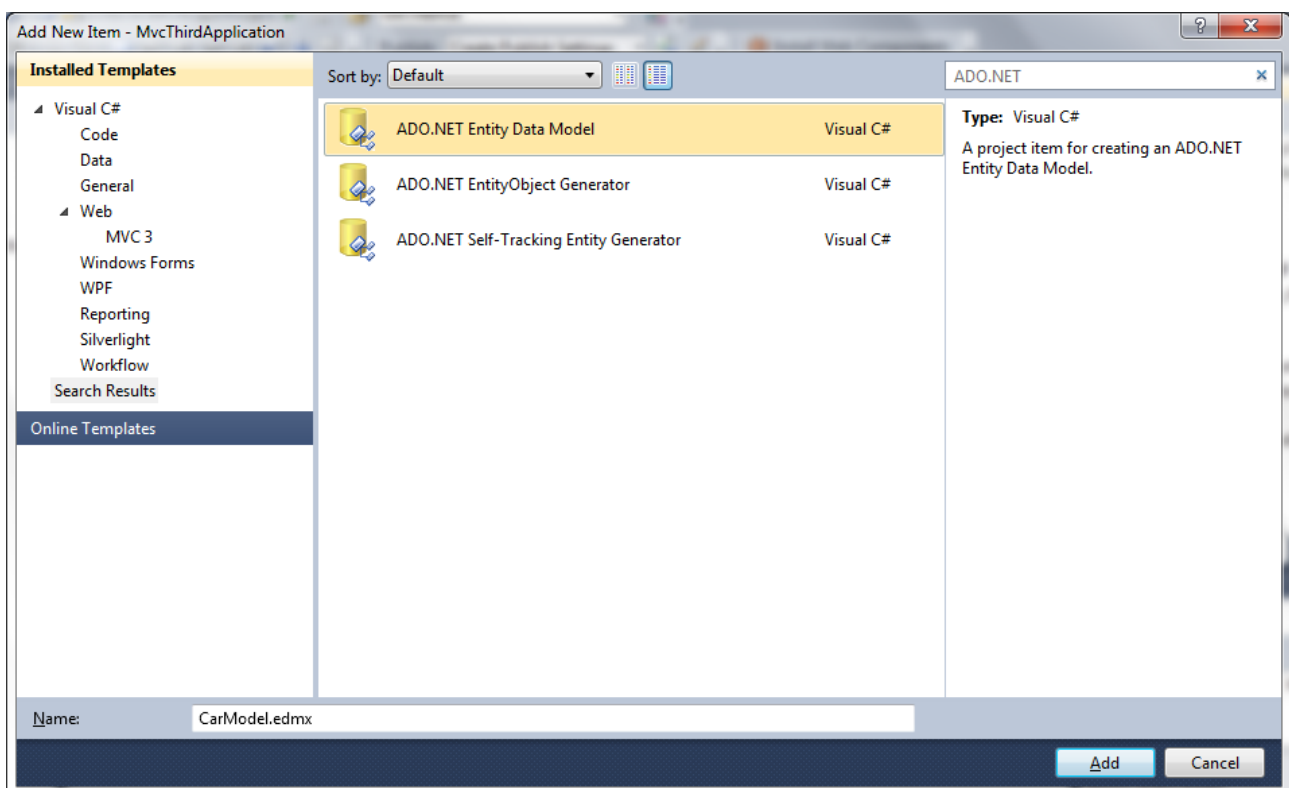
1. Close the solution (Visual Studio can be working)
2. Use File Explorer and delete the database file (**CarData.mdf**) and log file (**CarData_log.ldf**) in solution **App_Data** folder
3. Use File Explorer and copy the Car database and log files from previous Data Management programs to solution **App_Data** folder
4. Rename the files CarData.mdf and **CarData_log.ldf** .
5. Load the application to Visual Studio 2010.
6. Open the database and check that it works.

Application has been created and the Car Database has been attached to the application.

2. Generating the Entity Framework model

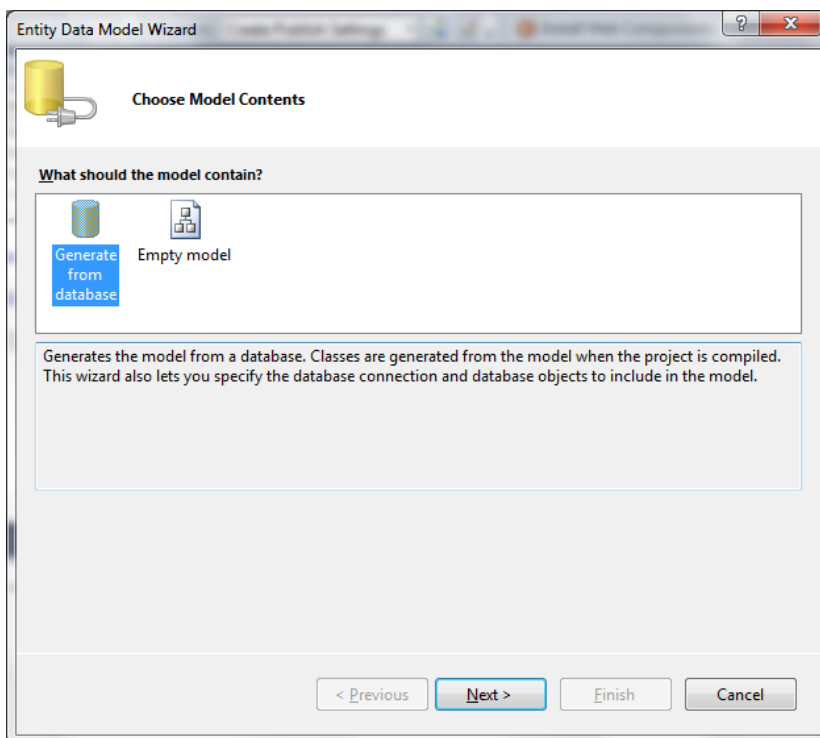
It is time to generate the Entity Framework model from the database:

- Right click the **Models** folder in Solution Explorer
- Select **Add** from the first menu and **New Item** from the second menu
- **Add New Item** dialog is opened. Type **ADO.NET** in the search box in the upper right corner
- Select **ADO.NET Entity Data Model** template
- Set name **CarModel.edmx** to the model



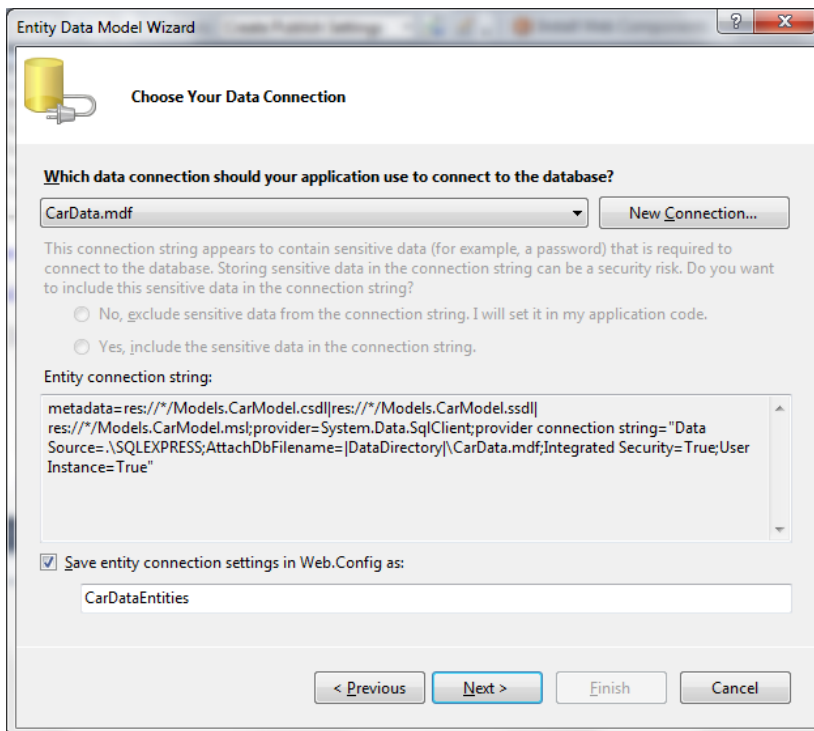
Picture: Add New Item template

- Select **Generate from database** template from the **Entity Data Model Wizard**



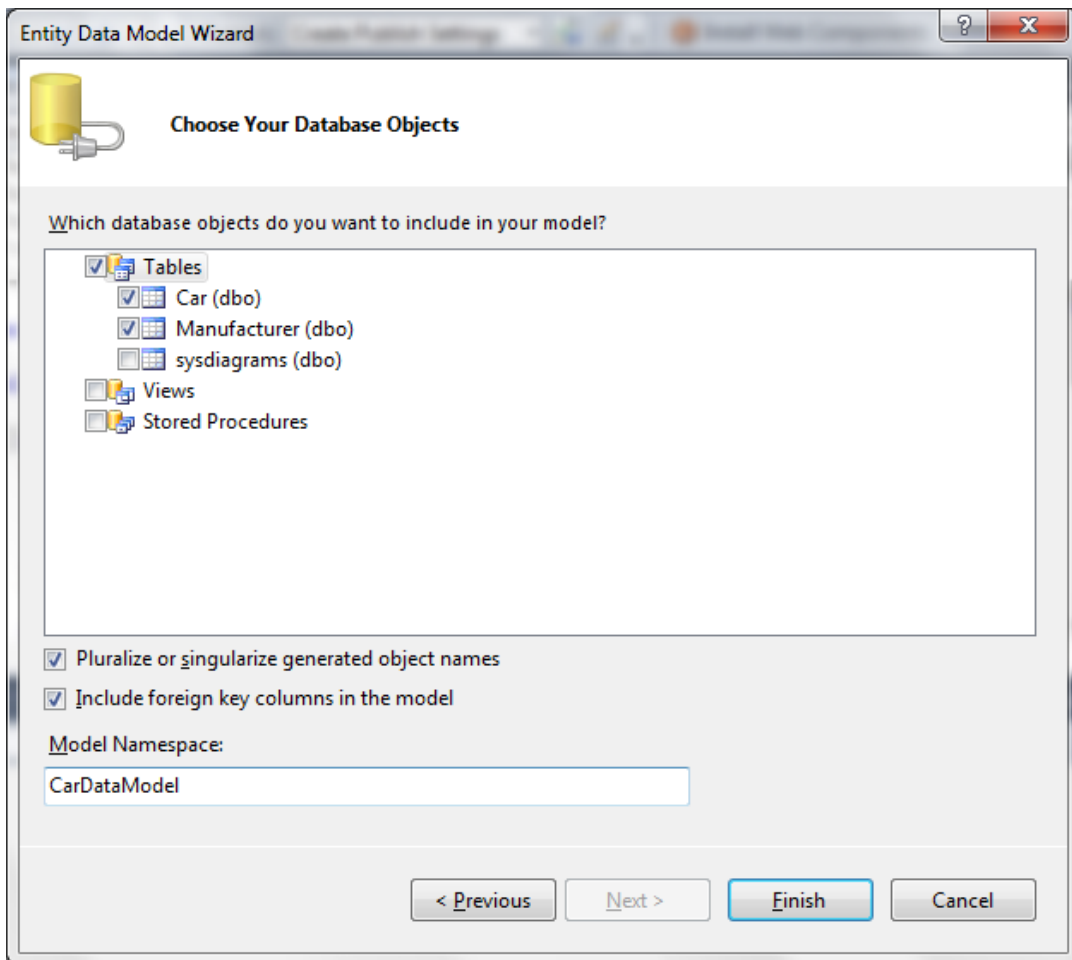
Picture: Entity Data Model Wizard model type

- Entity Data Model Wizard asks for connection string to the database. Connection string for the CarData database is given as the default.
- Entity Data Model Wizard suggests **CarDataEntities** as a name of entity connection settings



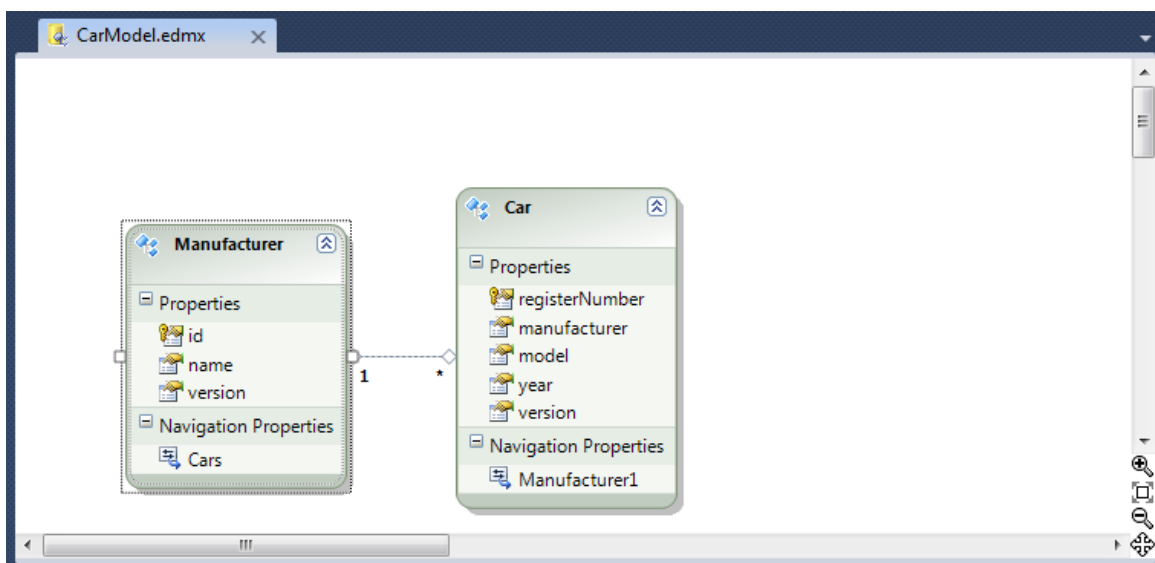
Picture: Entity Data Model Wizard for the connection string

- When default connection string is accepted it is possible to select tables for the model. Select the tables **Car** and **Manufacturer**

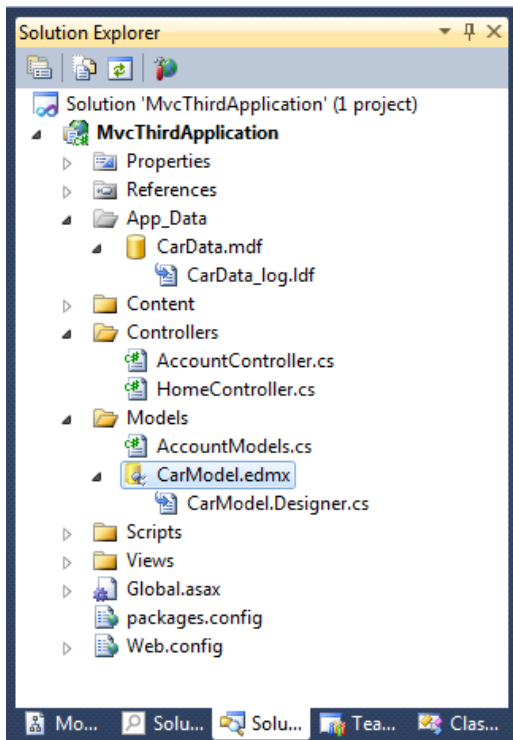


Picture: Entity Data Model Wizard for the tables

- When tables are selected the model is created



Picture: Created model called CarModel.edmx

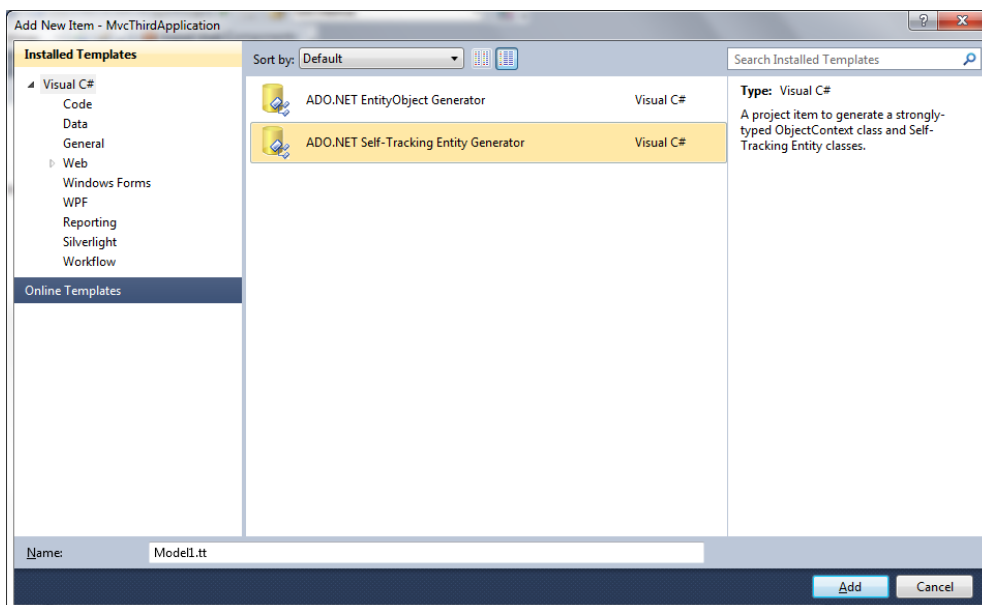


Picture Solution Explorer when the model has been added to the solution

3. Generating the code from Entity Framework model

When the model has been created it is time to generate strongly typed entity classes from the model.

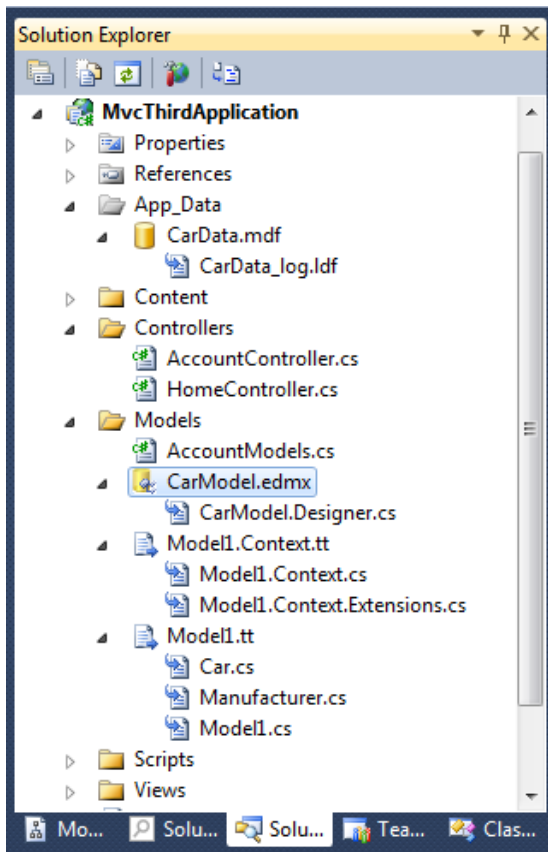
- Move the cursor over the model's designer surface and right-click
- Select **Add Code Generation Item** from the menu



Picture: Add New Item dialog

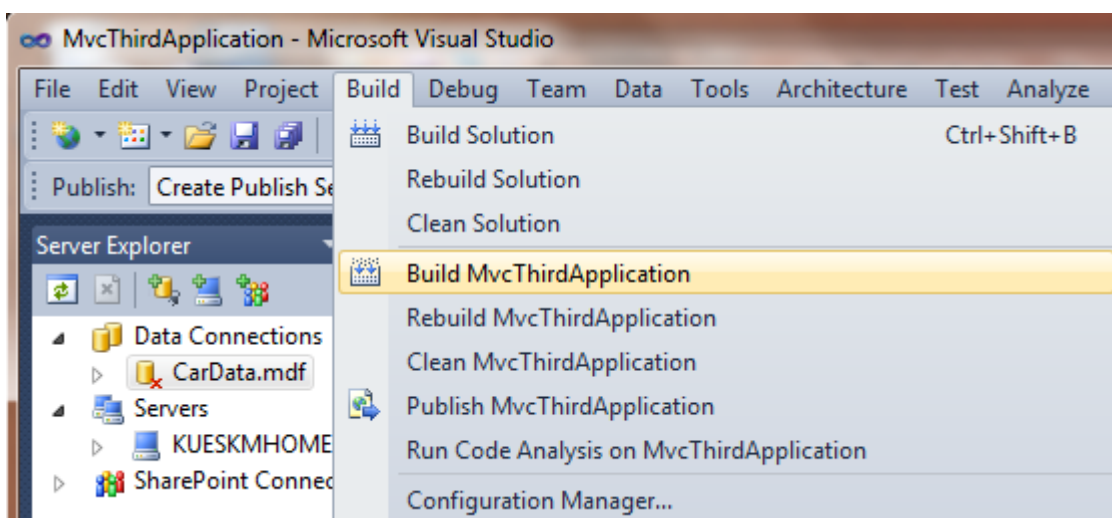
- Select Code in the **Add New Item dialog** types on the left
- Select Choose ADO.NET Self Tracking Entity Generator in the middle

You will get the entity classes as follows:



Picture: Solution Explorer when Entity generation is done.

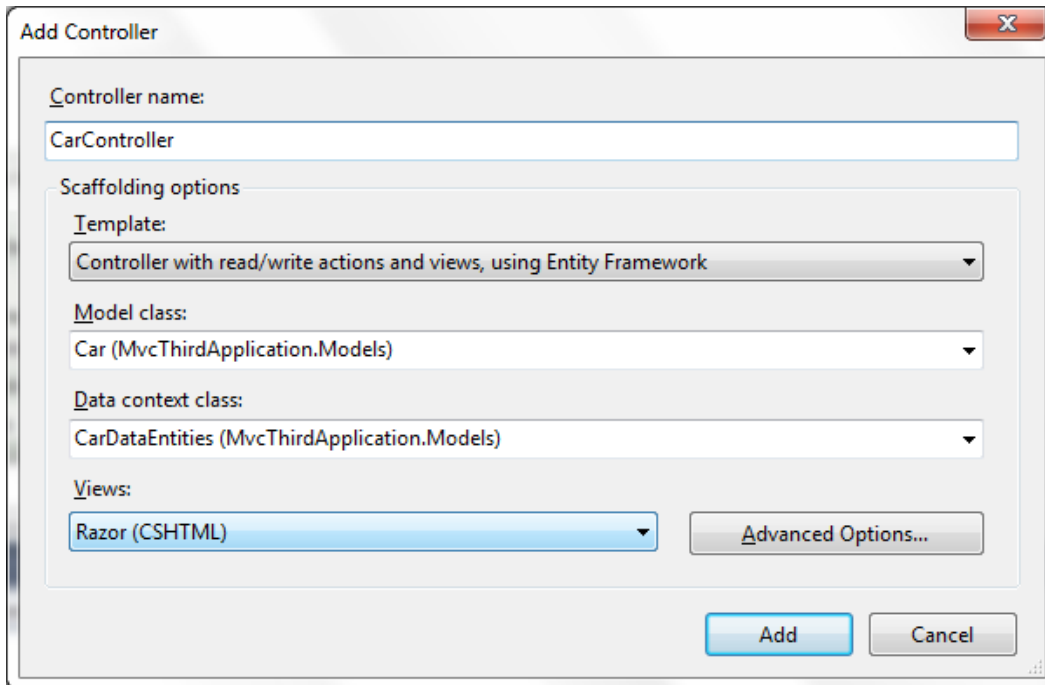
- When **model** has been added to the solution build the project: Choose command **Build** from the Visual Studio main menu and then choose **Build MvcThirdApplication** from the drop-down menu



Picture: Building the application

4. The Model has been created. Now it is time to add the controllers.

- In the Solution Explorer, right click the **Controllers** folder
- Select **Add** command from the menu. Select **Controller** from the next menu
- Add Controller dialog is opened:



Picture Add Controller dialog

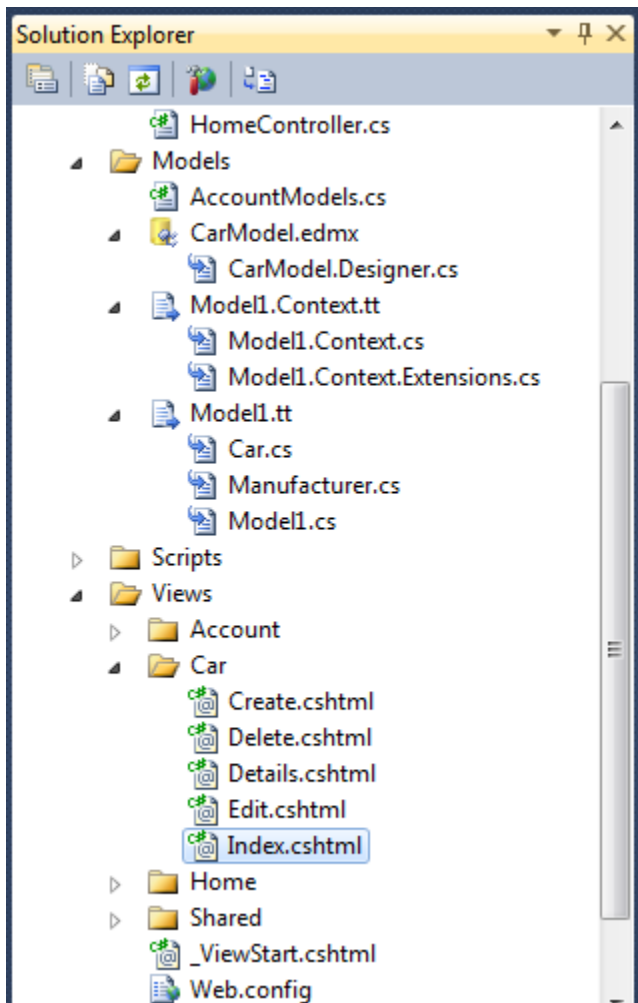
- Set **CarController** to Controller name
- Select **Controller with read/write actions and views, using Entity Framework** to Template
- Select **Car** to Model class
- Select **CarDataEntities** to data Context Class
- Select **Razor** to Views

When the controller is auto-created it will have the following methods:

- Index()
- Details()
- Create()
- Create(Car car)
- Edit(string id)
- Edit(Car car)
- Delete(string id)
- DeleteConfirmed(string id)

At the same time **Car** folder has been automatically added to **Views** folder. Car folder contains the following views:

- Create.cshtml
- Delete.cshtml
- Details.cshtml
- Edit.cshtml
- Index.cshtml



Picture: Views folder in Solution Explorer when **CarController** has been added to the solution

When **CarController** has been implemented the application already has some functionality. For instance the Car controller Index method creates the following user interface:

Index

[Create New](#)

registerNumber	Manufacturer1	model	year	version	
AAA-111	Ferrari	458 Spider	2010	1	Edit Details Delete
ABC-123	Alfa Romeo	Giulietta	2008	7	Edit Details Delete
BBB-222	Ford	Mondeo	2006	3	Edit Details Delete
CBA-888	Chevrolet	Aveo	2006	1	Edit Details Delete
DDD-333	Ford	Focus	2009	1	Edit Details Delete

Picture: Part of the user interface for Car controller Index method

In the user interface there is available one line for each and every table row in the database. At the end of the line are three links to load pages for editing, details or deleting the selected item. The code in **Index.cshtml** View for the links is following:

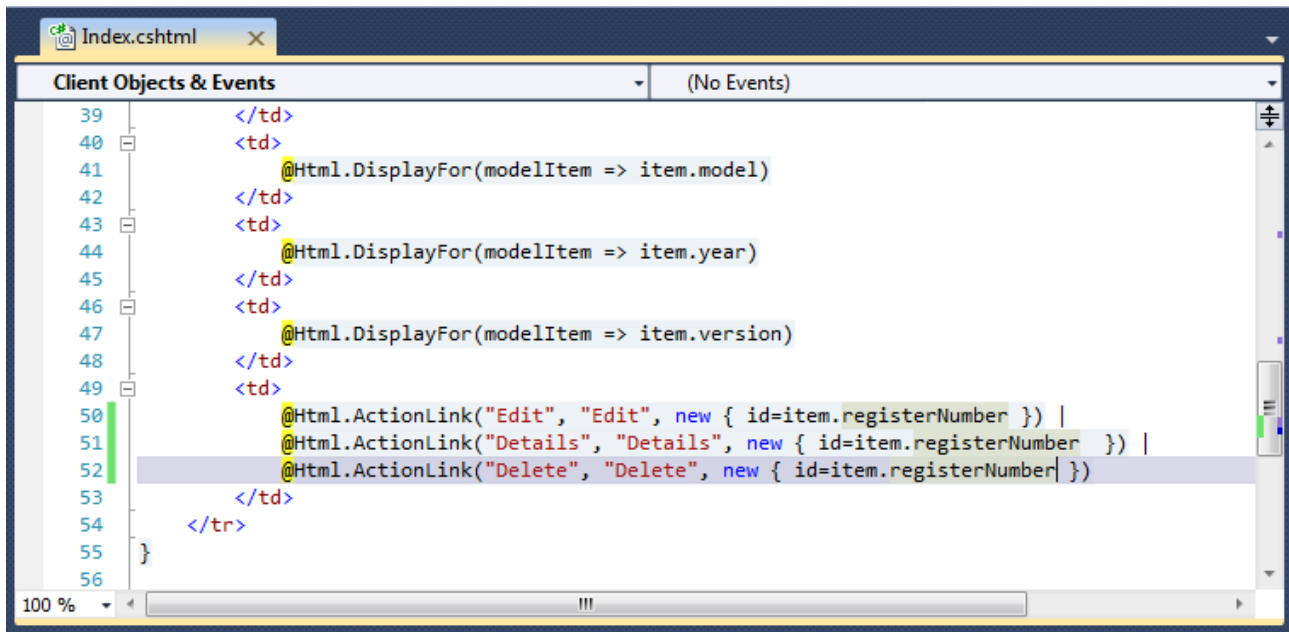
```
39 </td>
40 <td>
41     @Html.DisplayFor(modelItem => item.model)
42 </td>
43 <td>
44     @Html.DisplayFor(modelItem => item.year)
45 </td>
46 <td>
47     @Html.DisplayFor(modelItem => item.version)
48 </td>
49 <td>
50     @Html.ActionLink("Edit", "Edit", new { /* id=item.PrimaryKey */ }) |
51     @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
52     @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
53 </td>
54 </tr>
55 }
56
```

Picture: The **Index.cshtml** View code when code has been generated

@Html.ActionLink helper creates each link. Parameters for the helper are:

- "Edit"(first one) is the text displayed on the link
- "Edit"(second one) is the method name in the controller code: **Edit()**
- new { /* id=item.PrimaryKey */ } is the parameter for the method

The parameter for the method is not created because it is a comment so far. The comment must be replaced by the actual parameter setting:



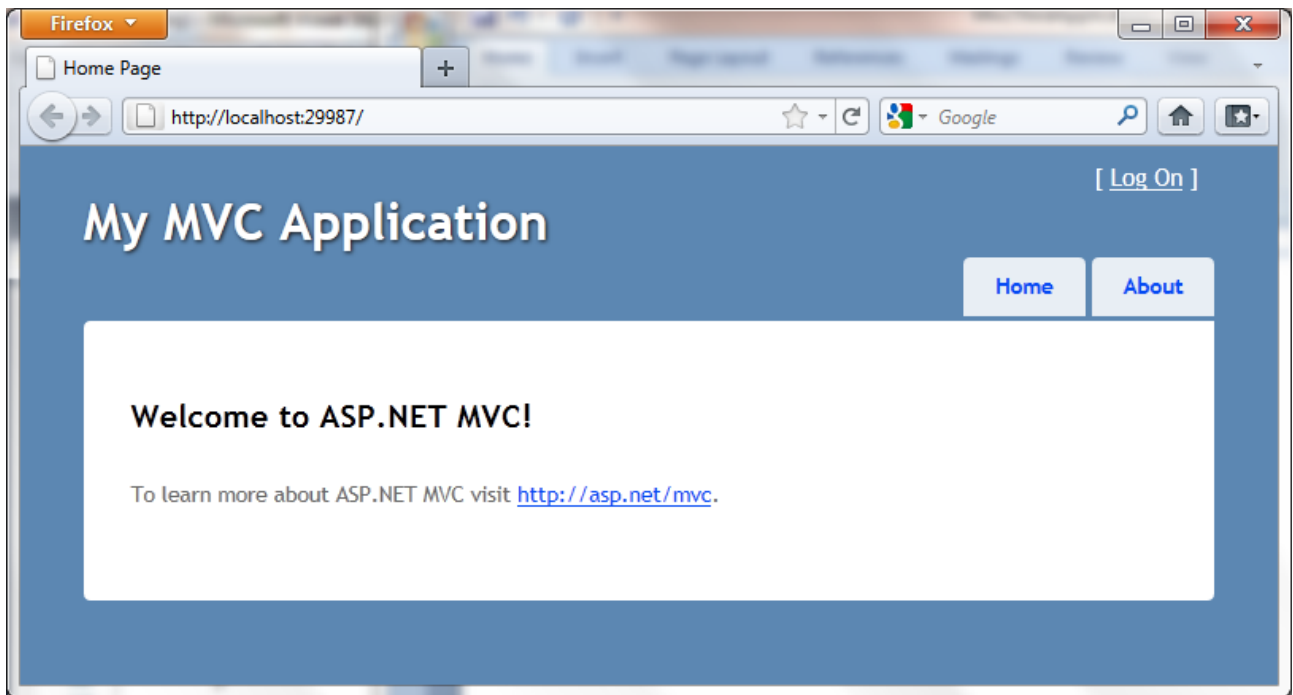
```
39      </td>
40      <td>
41          @Html.DisplayFor(modelItem => item.model)
42      </td>
43      <td>
44          @Html.DisplayFor(modelItem => item.year)
45      </td>
46      <td>
47          @Html.DisplayFor(modelItem => item.version)
48      </td>
49      <td>
50          @Html.ActionLink("Edit", "Edit", new { id=item.registerNumber }) |
51          @Html.ActionLink("Details", "Details", new { id=item.registerNumber }) |
52          @Html.ActionLink("Delete", "Delete", new { id=item.registerNumber })
53      </td>
54  </tr>
55  }
56  }
```

Picture: The **Index.cshtml** View code when the parameters for links are created

When these corrections are made it is possible to also update information in the Car table.

5. Updating the links in the home page

When the application is started after the code generation the Home page is loaded first. No changes have been made to the Home page and it does not have links to Car controller methods.



Picture: Home page for the site when no updates are done

All the pages in the site have the same master page. The master page code can be found from the View `_Layout.cshtml`. The links in the page are created with the code below:

```
<nav>
  <ul id="menu">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("About", "About", "Home")</li>
  </ul>
</nav>
```

Html.ActionLink helper is used again to create the links. It is possible to create the links for Car controller and Manufacturer controller using the code below:

```
<nav>
  <ul id="menu">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("About", "About", "Home")</li>
    <li>@Html.ActionLink("Car", "Index", "Car")</li>
    <li>@Html.ActionLink("Manufacturer", "Index", "Manufacturer")</li>
  </ul>
</nav>
```

When the links in the Home page are updated it is time to test the application.

AFTER FINISHING THE TASK, GO THROUGH ALL THE STEPS AT LEAST ONE EXTRA TIME! You will only learn by understanding. Just following the instructions is not enough. So: Think, discuss and ask!