## Lab Work #1. Databases in web development. Data models. Description, examples of creation and use

For demonstration of working with SQL Server database, I will use IDE Visual Studio Community, since this environment already contains all the necessary tools for working with databases, creating data models, and writing the necessary code.

Let us consider how to build data models in ASP.NET MVC Framework application. The model stores data that is retrieved from database according to controller methods and displayed in the view.

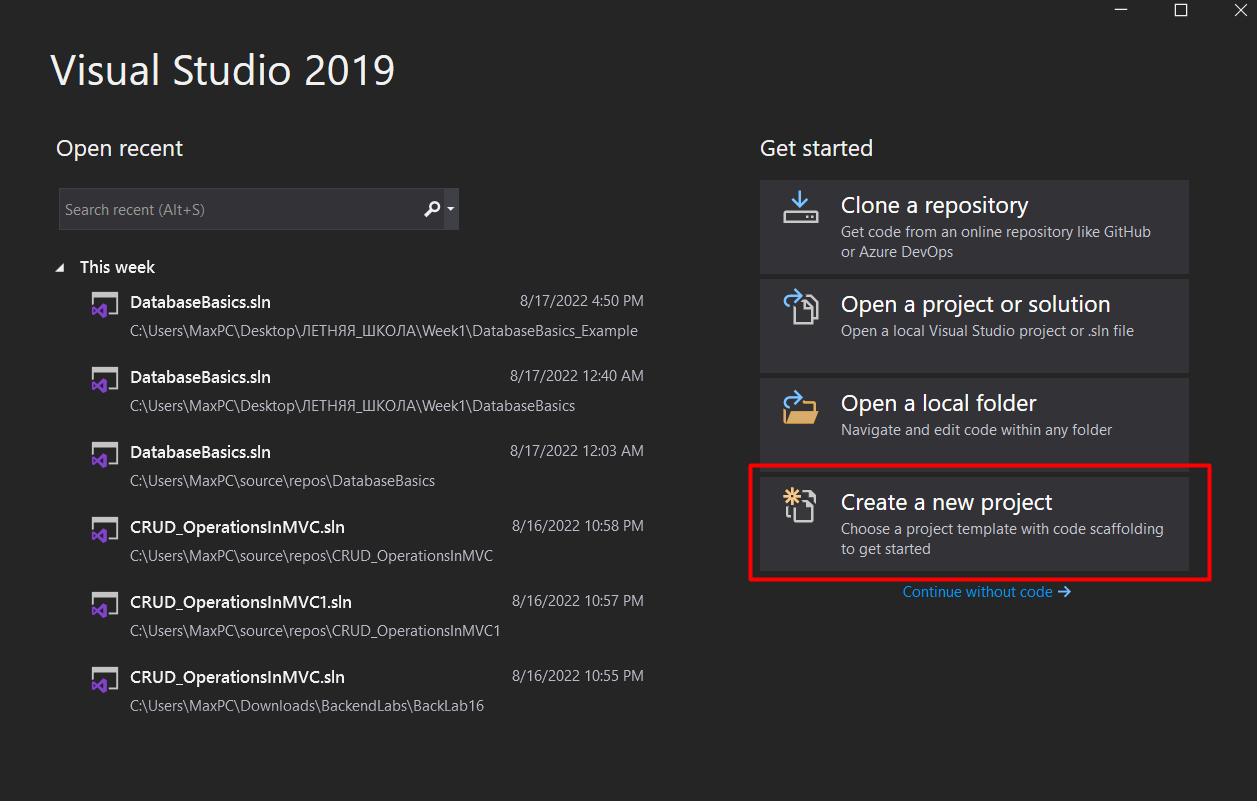
A model is a set of classes that are used to work with the application's data and business logic. Therefore, models are a kind of container for specific business domains. The model is used to interact with the database, as well as to manipulate data when implementing business logic.

In ASP.NET MVC application, the controller responds to an incoming HTTP request, then receives data from the model and passes it to the view. The view displays the data. The model can contain entities or business objects.

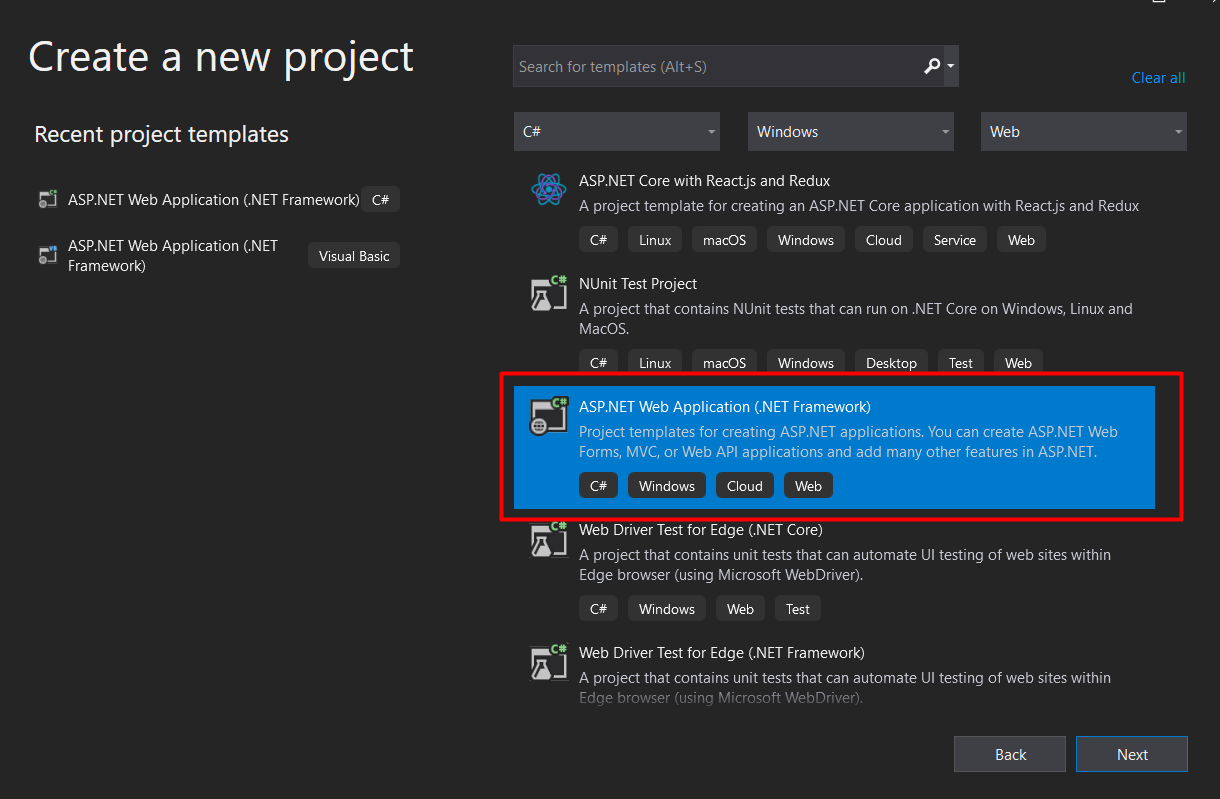
This example demonstrates how to retrieve data from SQL Server Express database using the «Database First Method» in Entity Framework (it means that a database already exists).

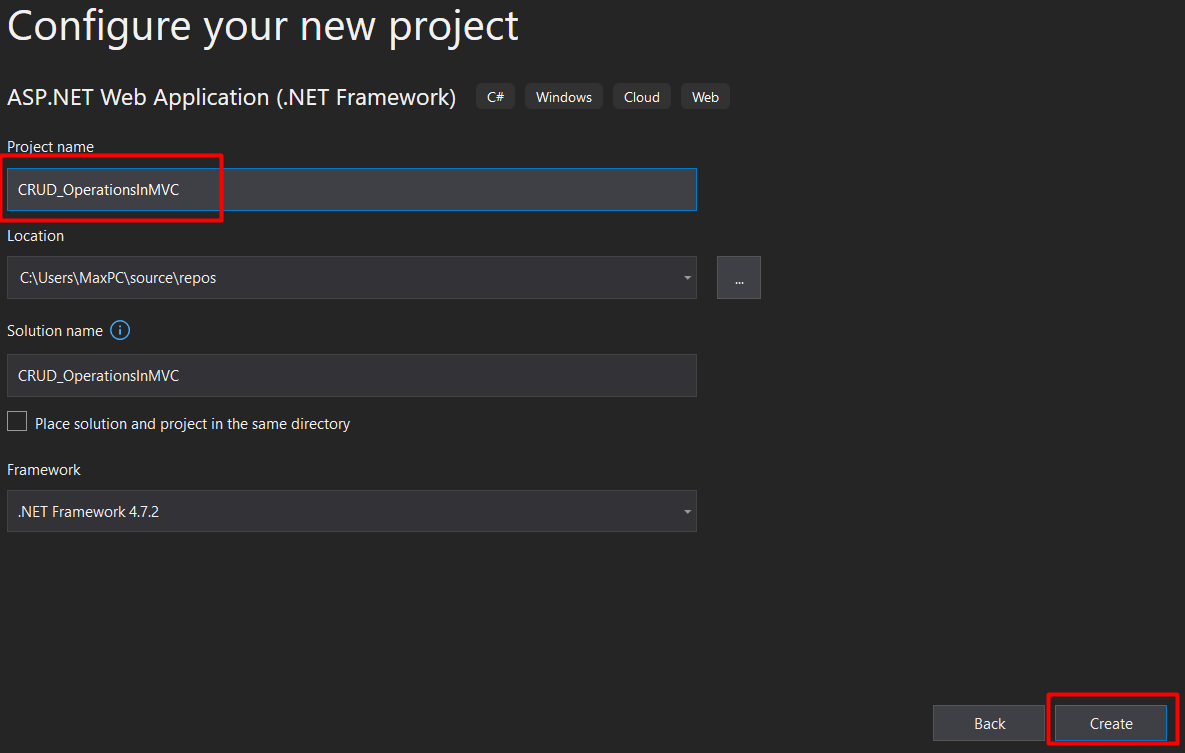
### Step 1: Creation of new application and adding a database

1. Open MS Visual Studio Community development environment. Select "Create a new Project".

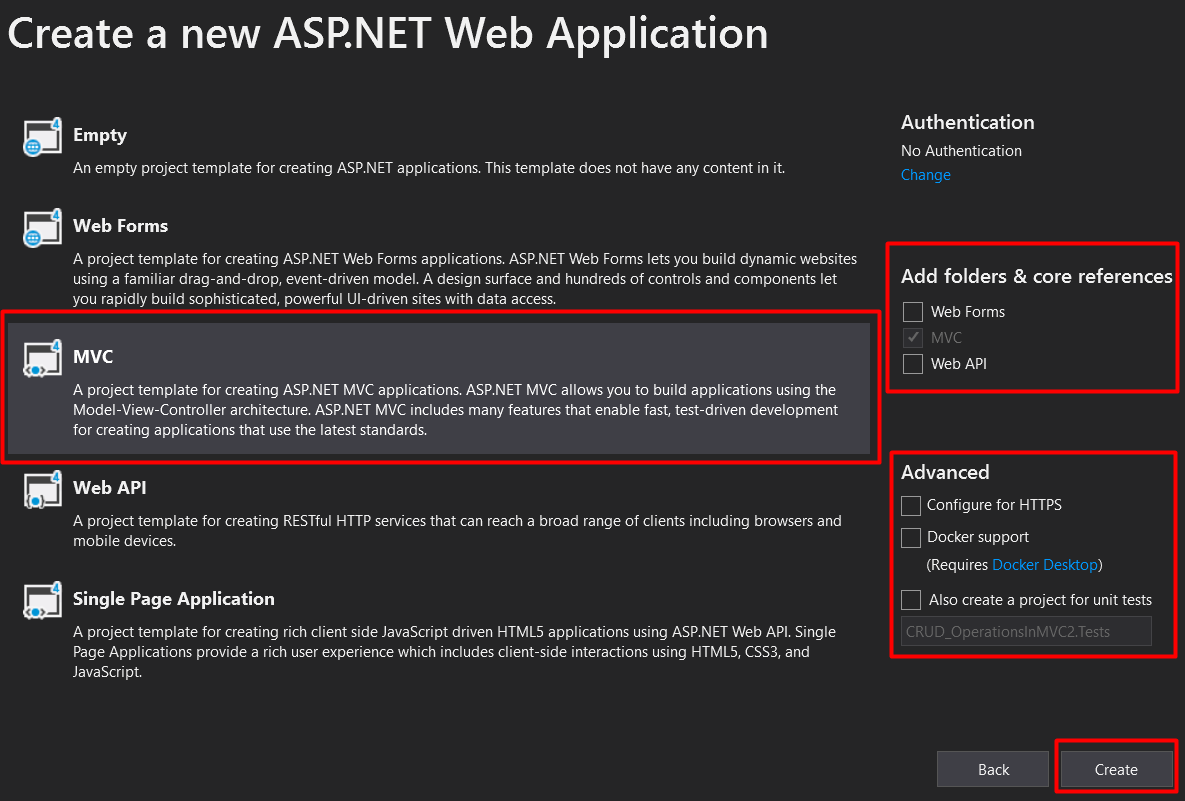


1. Create a new blank ASP.NET MVC application named "CRUD\_OperationsInMVC" with no built-in authentication mechanism (need to uncheck HTTPS).

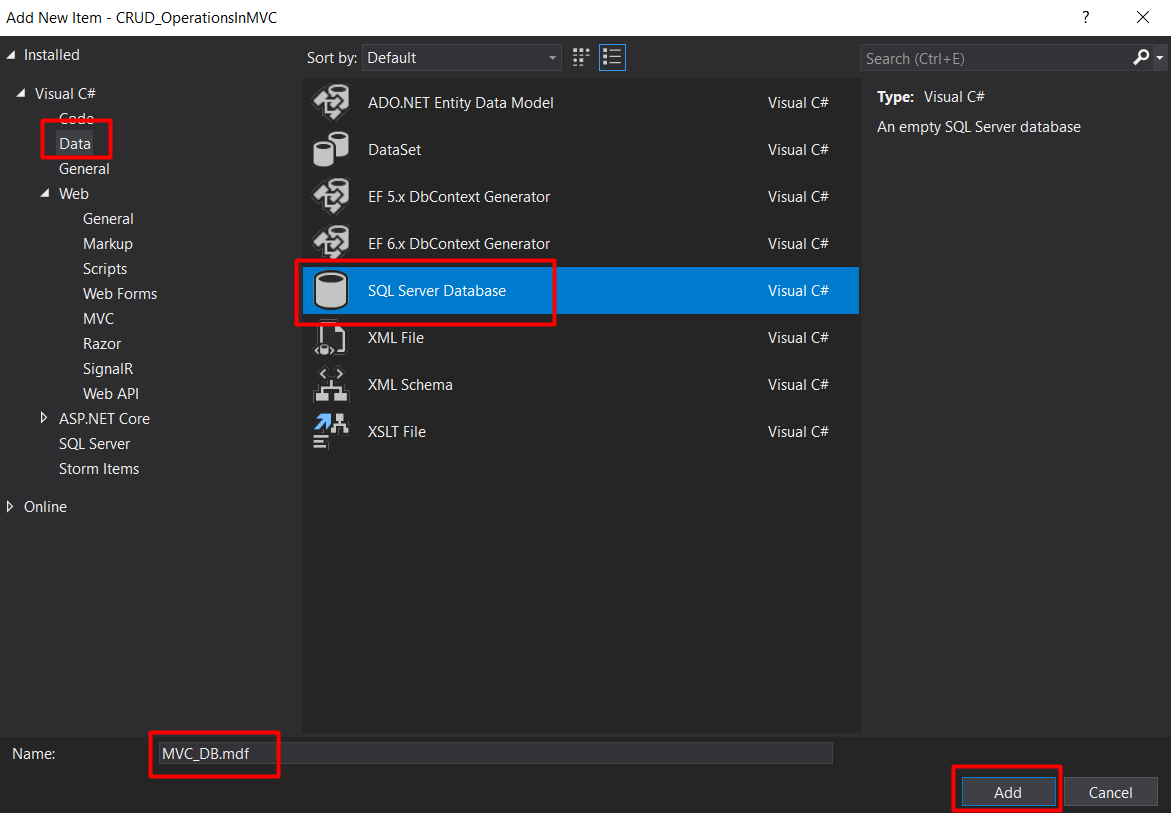




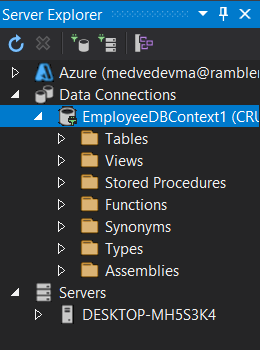
Set the project properties as shown in the figure below and click «Create»:



1. Add a new database to the application. For this:
2. Right-click on the project name, from the context menu select "Add" -> "New Item".
3. In the dialog that appears, select the “Data” tab on the left, select “SQL Server Database” in the middle part (see the figure below).
4. Enter the database name "MVC\_DB" and click the "Add" button. The development environment will create a new database in the "App\_Data" folder in the "Solution Explorer".



1. Double click on the database file "MVC\_DB.mdf" in "Solution Explorer". The Server Explorer window will open as shown below:



1. Add a new table to the database. In order to do this, right-click on the database name and select "Add New Table" from the context menu. The table designer will open.
2. Copy the following code to the bottom of the designer (T-SQL) window:

CREATE TABLE [dbo].[Employees]

(

EmployeeId int Primary Key Identity(1,1),

Name nvarchar(50),

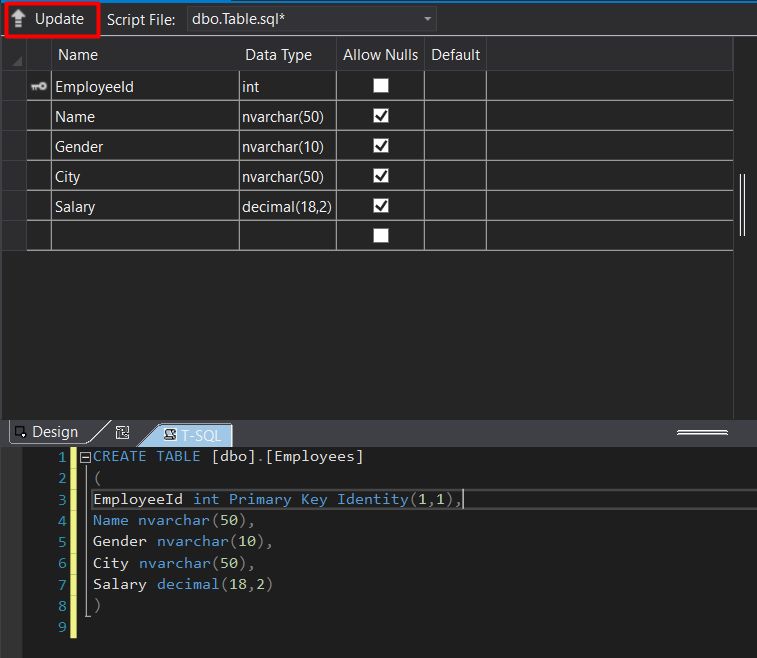
Gender nvarchar(10),

City nvarchar(50),

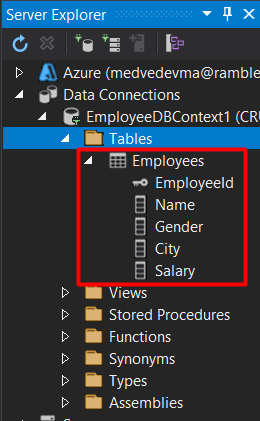
Salary decimal(18,2)

)

The table fields in the top window will automatically update.



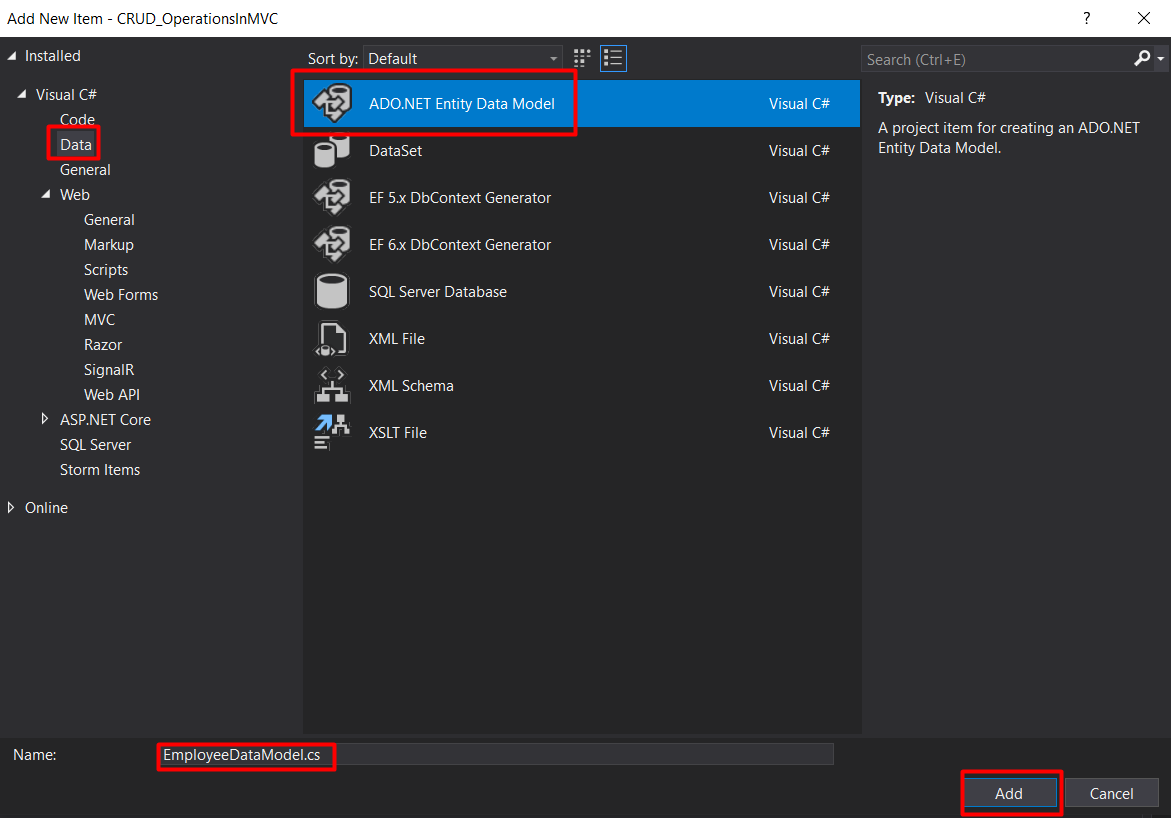
1. Click the «Update» button in the top left corner of the table builder.
2. Next, in the Preview window, click the «Update Database» button. The development environment will create a new table named "Employee" in the database.



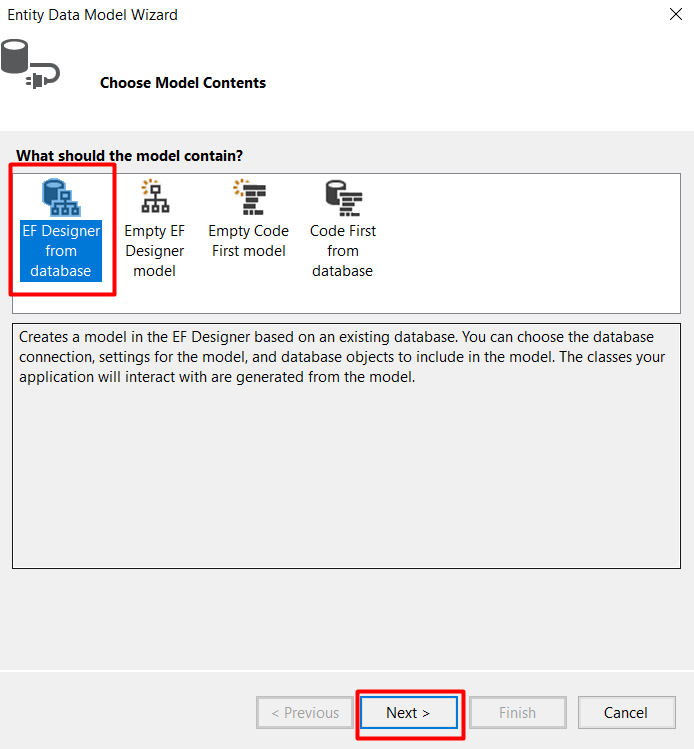
1. Add data to the created table. Right-click on the table name and select "Show Table Data" from the context menu.
2. Enter at least four lines of data (the key field is updated automatically; you do not need to enter it).

### Step 2: Adding the Entity Data Model to the Application

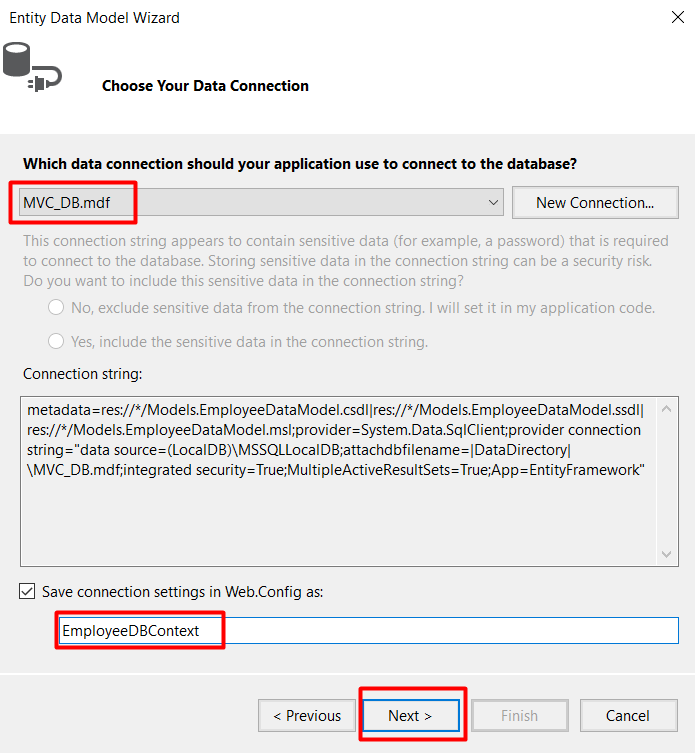
1. Right-click on the "Models" folder and select «Add» -> «New Item» from the context menu.
2. In the «Add New Item» window, in the left tab, expand «Visual C#» => «Data».
3. In the middle tab, select the «ADO.NET EDM Model» template. Provide a meaningful name for your data model, such as «EmployeeDataModel» and click the «Add» button.



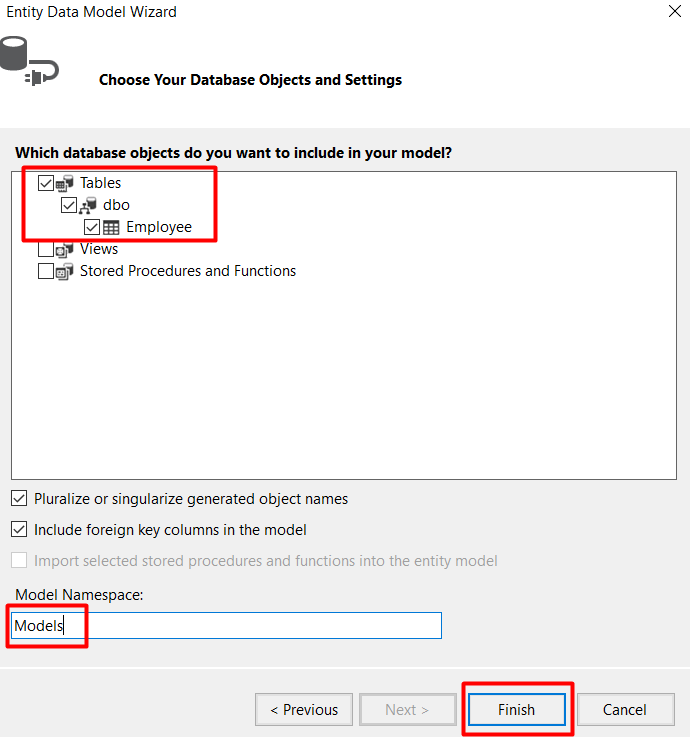
1. Since we are going to use the Entity Framework Database First approach, select EF Designer from Database in the next dialog and click «Next».



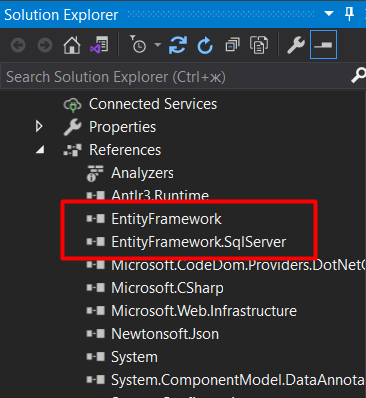
1. Specify a meaningful name for the connection string that will be saved in the web.config file (in this work, it will be named "EmployeeDBContext") and click the "Next" button, as shown below.



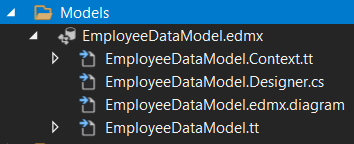
1. In this project, we will use "Entity Framework version 6.x", so in the next version selection window, select "Entity Framework 6.x" and click the «Next» button.
2. Next, we need to select the database objects for our application. Since we have one table in our database, we need to select this table «Employee». Enter a meaningful name for the namespace of your EDMX file and click the «Finish» button as shown below.



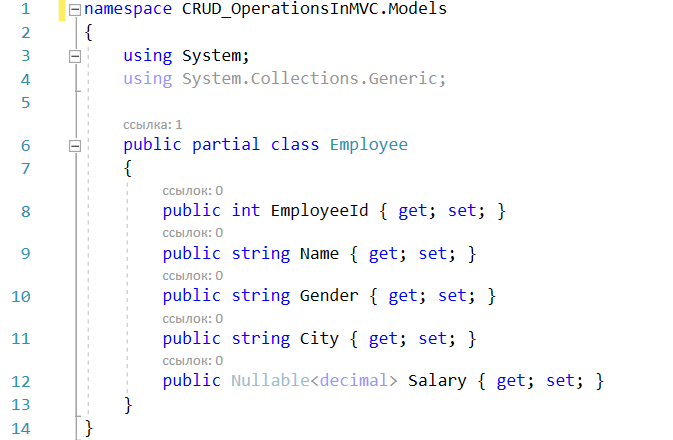
After you click the Finish button, the development environment will add a reference to the Entity Framework to the References folder.



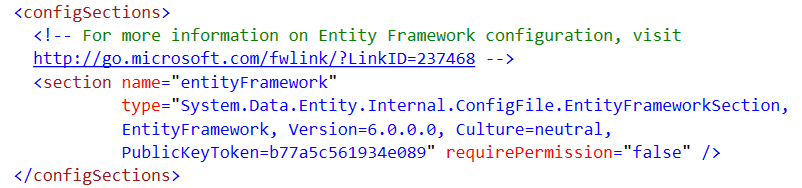
An 'EDMX' file will also be created in the «Models»folder:



To see the created employee model file, double click on the «Employee.cs file» (it is located inside the «EmployeeDataModel.tt» file):

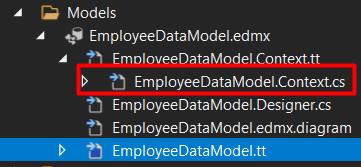


The connection string will also be created in the «web.config» file as shown below:

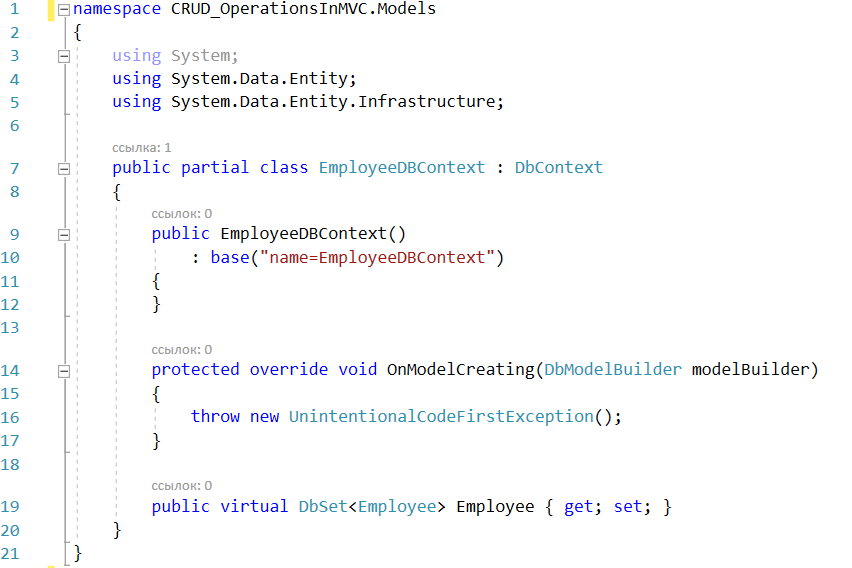


The class «DBContext» will be created, the name of which will be the same as the name of the connection string «EmployeeDBContext». The «EmployeeDBContext» class is derived from the "DbContext" class and is responsible for establishing a connection to the database.

The data context class «EmployeeDataModel.Context.cs» is inside «EmployeedataModel.Context.tt” as shown below:

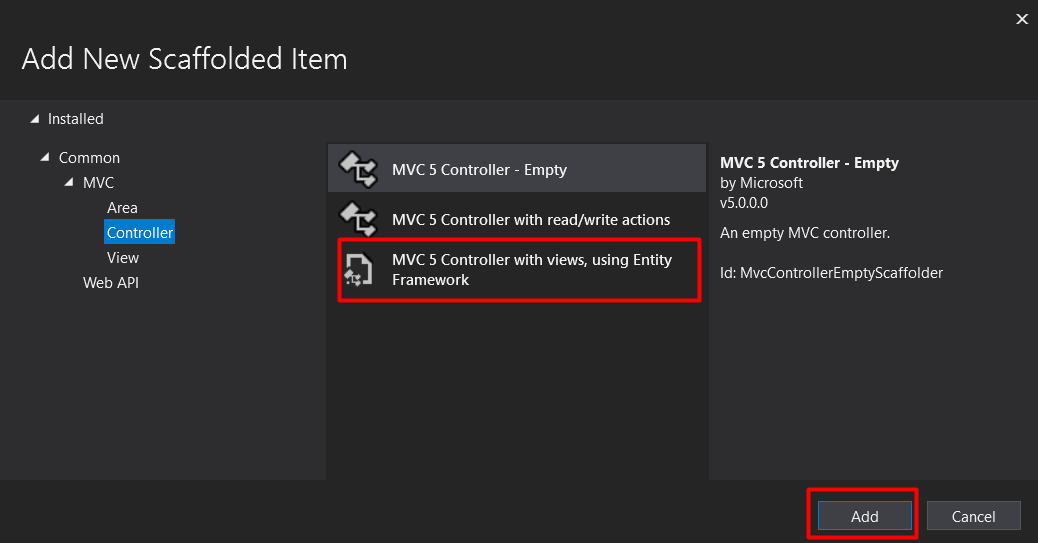


This is how the automatically generated code for the «EmployeeDBContext» class looks like:

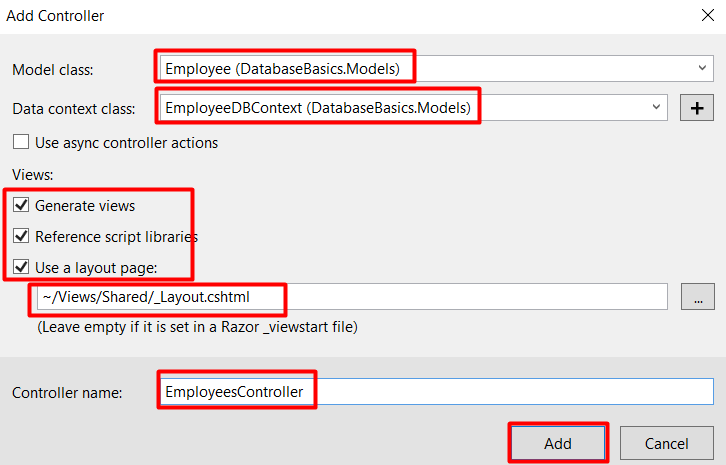


We have successfully created the Entity Data Model. Now we can use it in the application.

1. Right click on the «Controllers» folder. Select «Add» => «MVC5 controller with Views, using Entity Framework» and click the «Add» button.



1. In the next «Add Controller» dialog, specify all the necessary settings, as shown in the figure below:



and click the «Add» button.

The development environment will create a new controller named «EmployeesController» with all the necessary methods for working with the database, as well as the corresponding views. The views will be placed in the «Views» folder. The complete controller code is shown below:

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using DatabaseBasics.Models;

namespace DatabaseBasics.Controllers

{

public class EmployeesController : Controller

{

private EmployeeDBContext db = new EmployeeDBContext();

// GET: Employees

public ActionResult Index()

{

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

}

// GET: Employees/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Employee employee = db.Employee.Find(id);

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

// GET: Employees/Create

public ActionResult Create()

{

return View();

}

// POST: Employees/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "EmployeeId,Name,Gender,City,Salary")] Employee employee)

{

if (ModelState.IsValid)

{

db.Employee.Add(employee);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(employee);

}

// GET: Employees/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Employee employee = db.Employee.Find(id);

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

// POST: Employees/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "EmployeeId,Name,Gender,City,Salary")] Employee employee)

{

if (ModelState.IsValid)

{

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

db.SaveChanges();

return RedirectToAction("Index");

}

return View(employee);

}

// GET: Employees/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Employee employee = db.Employee.Find(id);

if (employee == null)

{

return HttpNotFound();

}

return View(employee);

}

// POST: Employees/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

Employee employee = db.Employee.Find(id);

db.Employee.Remove(employee);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

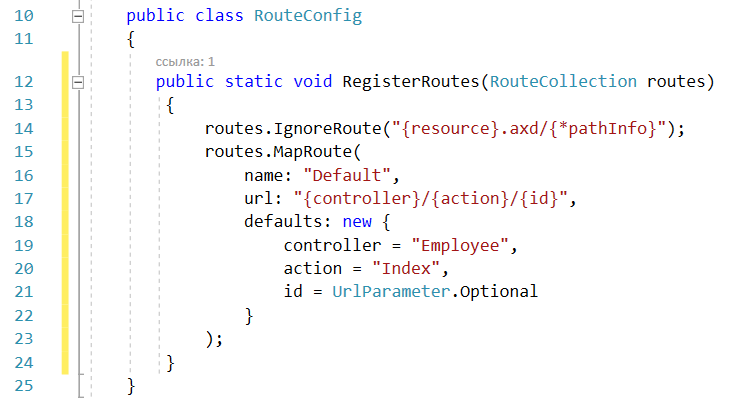
base.Dispose(disposing);

}

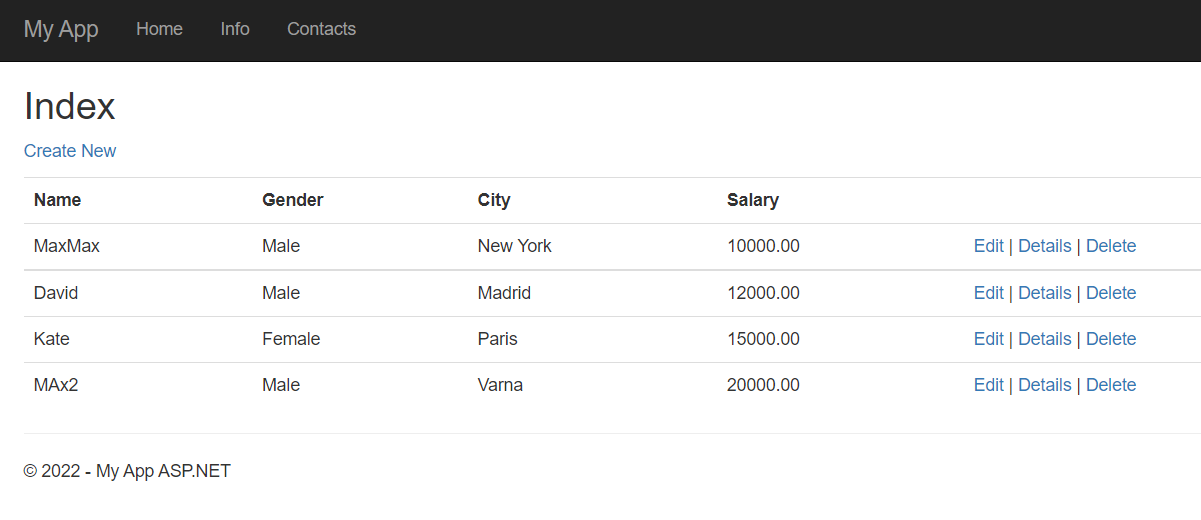
}

}

1. Let us change the default route in the «Route.Config» class so that the first time the application is launched, we will be redirected to the «Index” action method in the «EmployeeController». After changing the «RegisterRouts» class, the code will look like this:



1. Run the application (Ctrl+F5). You will see the following page:



Check all links from the right in the table. The development environment and the Entity Framework have created automatically all the necessary methods and program code for working with the data in the «Employees» table.