

# Комп'ютерний практикум 11

## Створення моделі MVC ASP.NET з використанням Entity Framework.

### Мета

Ознайомлення з можливостями Entity Framework практичне застосування їх в MVC ASP.NET.

### Зміст

1. Створення нової бази даних за допомогою Entity Framework.
2. Створення моделі бази даних.
3. Використання даних отриманих з БД.
4. Створення нової сутності за допомогою Entity Framework.
5. Оновлення вже існуючої у БД сутності.
6. Валідація даних.
7. Видалення сутності з БД.

### Література

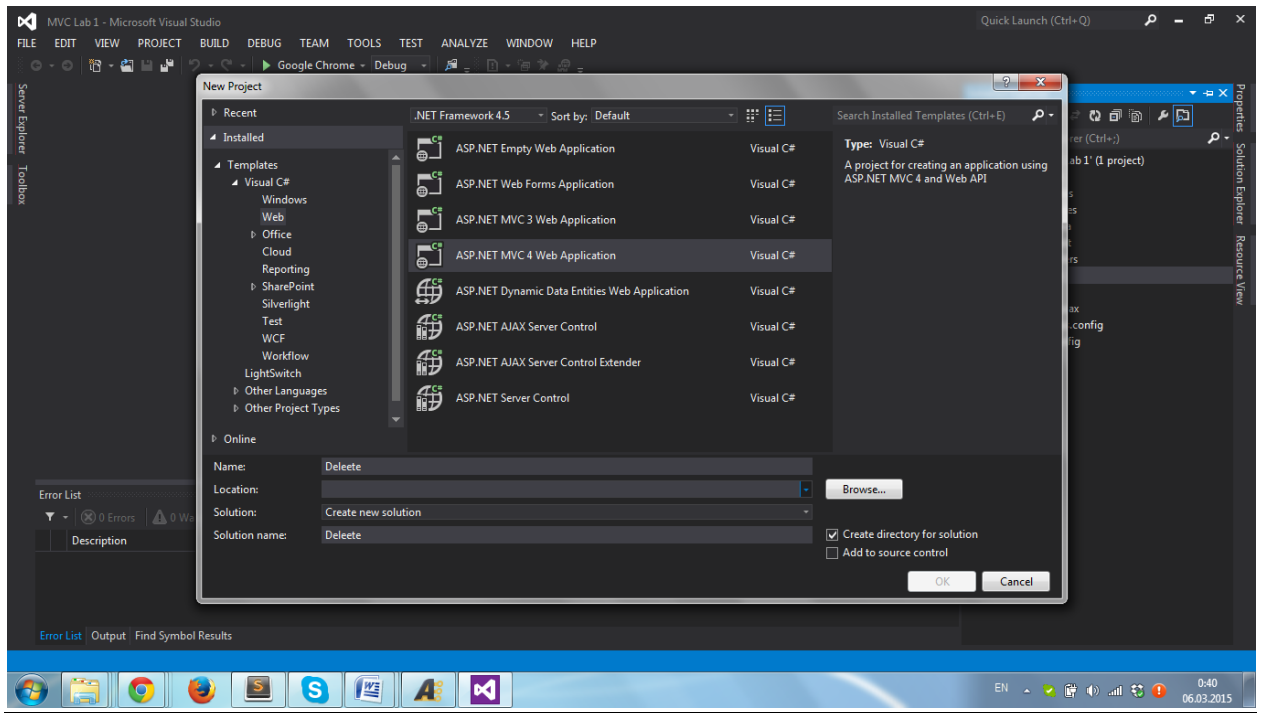
- ASP.NET MVC <http://habrahabr.ru/post/175999/>
- <http://www.asp.net/mvc>
- ASP.NET MVC <http://habrahabr.ru/post/175999/>
- <http://www.asp.net/mvc>
- Відео семпли у простій, зрозумілій формі  
<https://www.youtube.com/watch?v=KEibrCrpt6I&index=1&list=PLFE6E1A5CD2570B4C>

### Частина 1

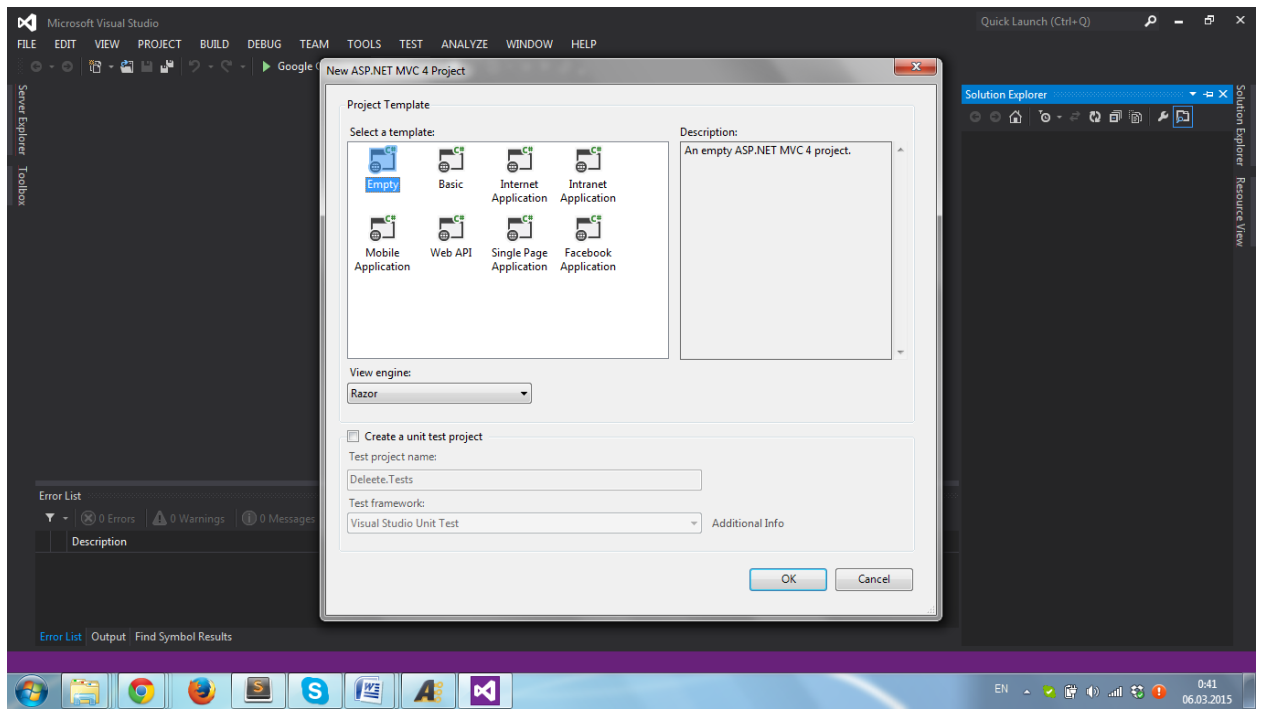
#### Виконання:

Створити 3 локальні таблиці ( Співробітник, Співробітник-Спеціалізація, Спеціалізація). Створити MVC додаток, який буде виводити весь перелік співробітників, а біля ім'я кожного буде посилання на його спеціалізацію.

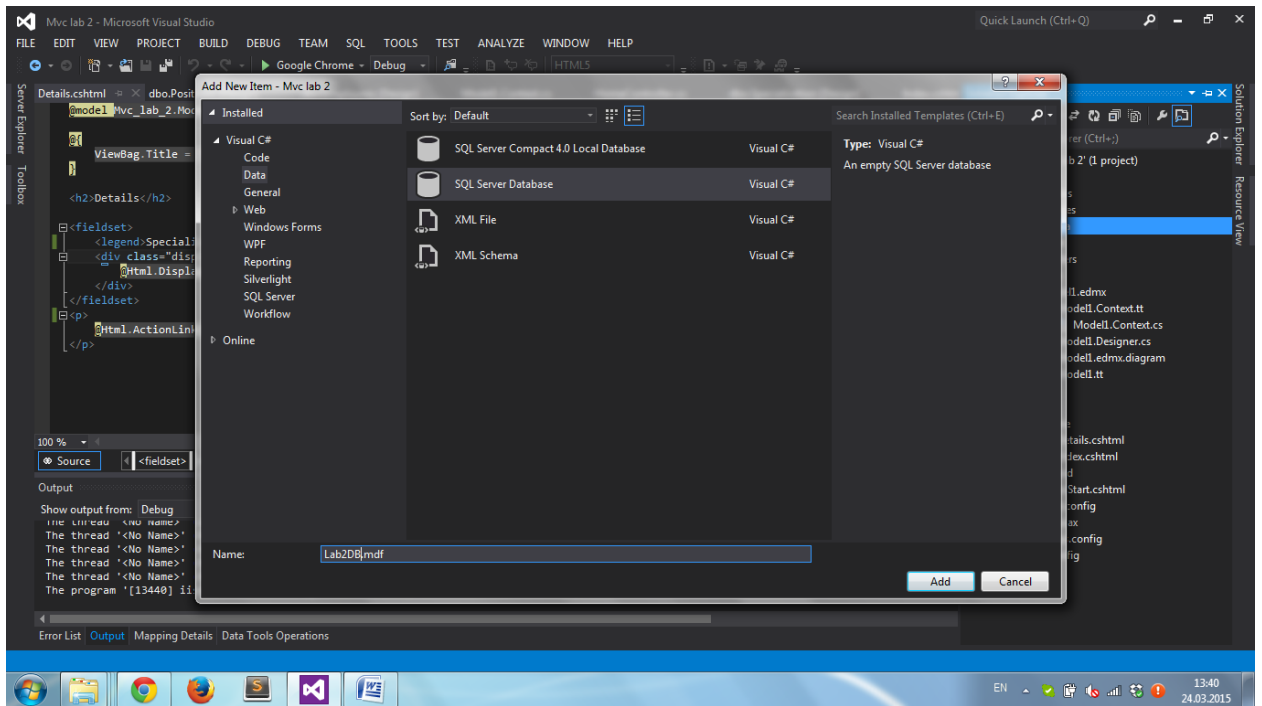
- 1.Сворити проект ASP.NET MVC



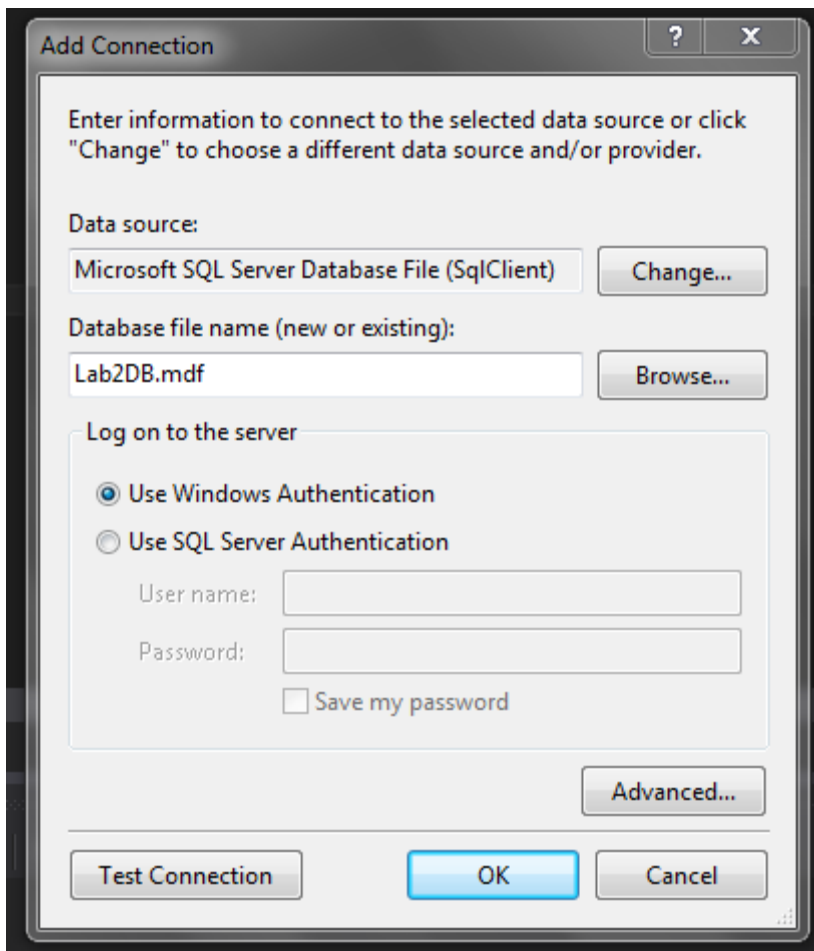
2. Обрати порожній проект. Для опису представлення буде використано синтаксис Razor.



3. У проекті створюємо нову базу даних.



4. Після створення БД переходимо у Server Explorer, де підключаємо нашу БД до проекту, якщо це не відбулося автоматично.

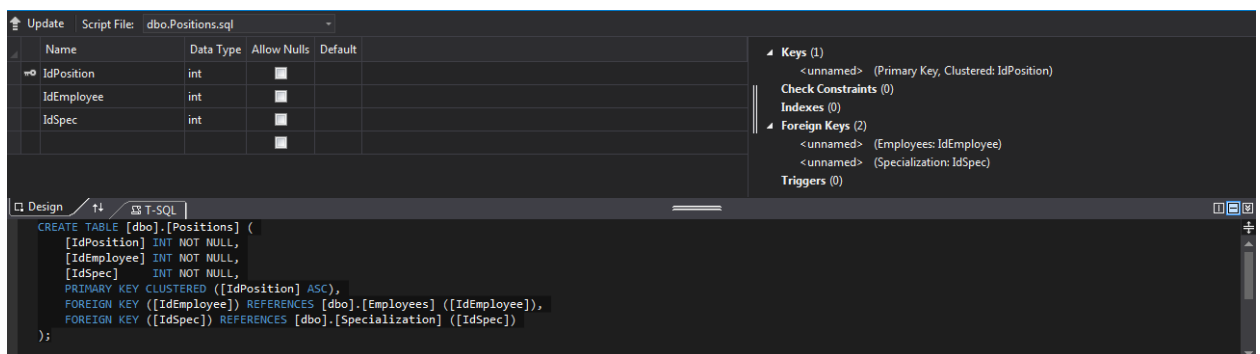
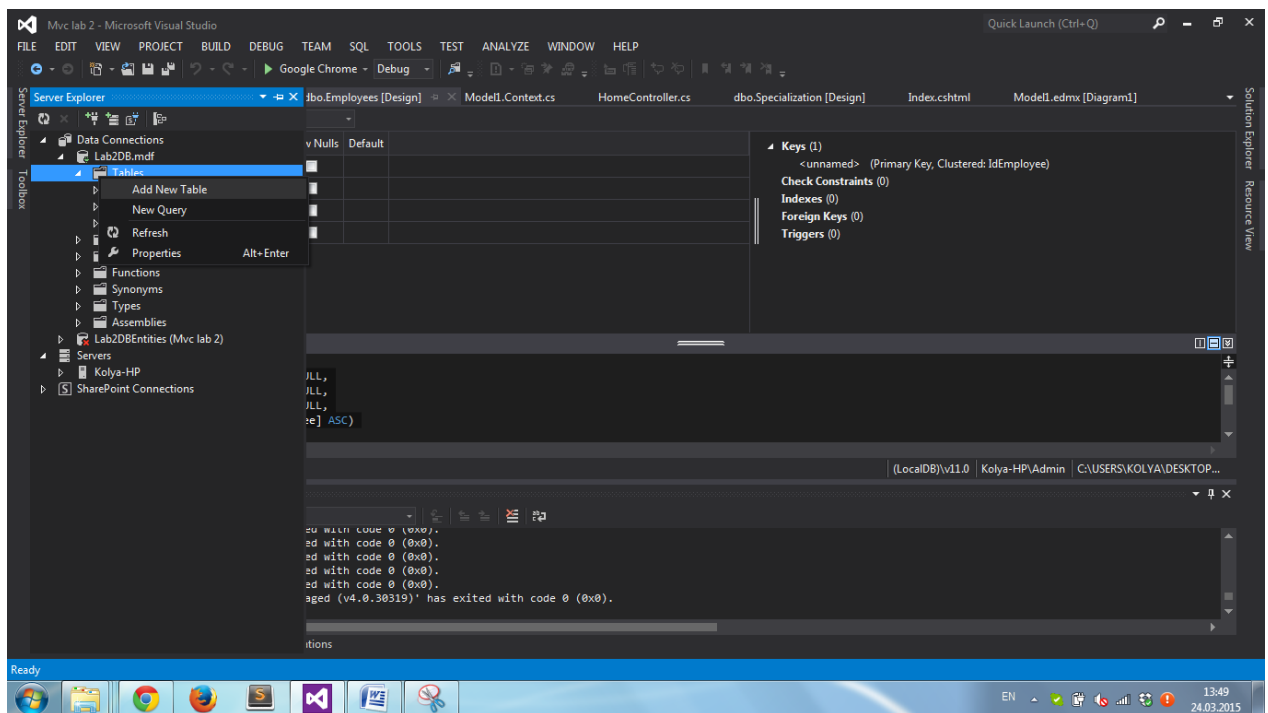


Обов'язково натискаємо кнопку Test Connection, для перевірки з'єднання.

Після цього ваше з'єднання повинно з'явитися у вкладці Server Explorer.

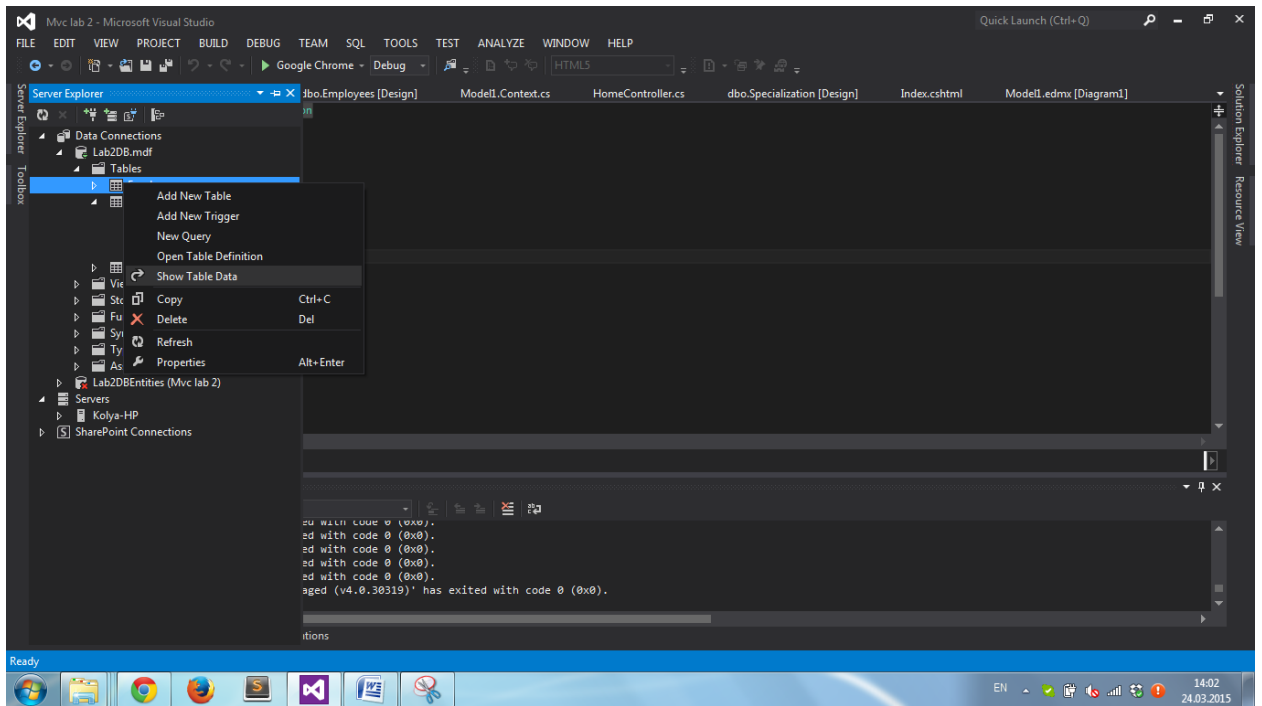
Створена база даних не буде мати таблиць, тому їх треба створити, що і буде виконано на наступному кроці.

5. Додаємо до нашої бази даних необхідні таблиці, з необхідними первинними та зовнішніми ключами.

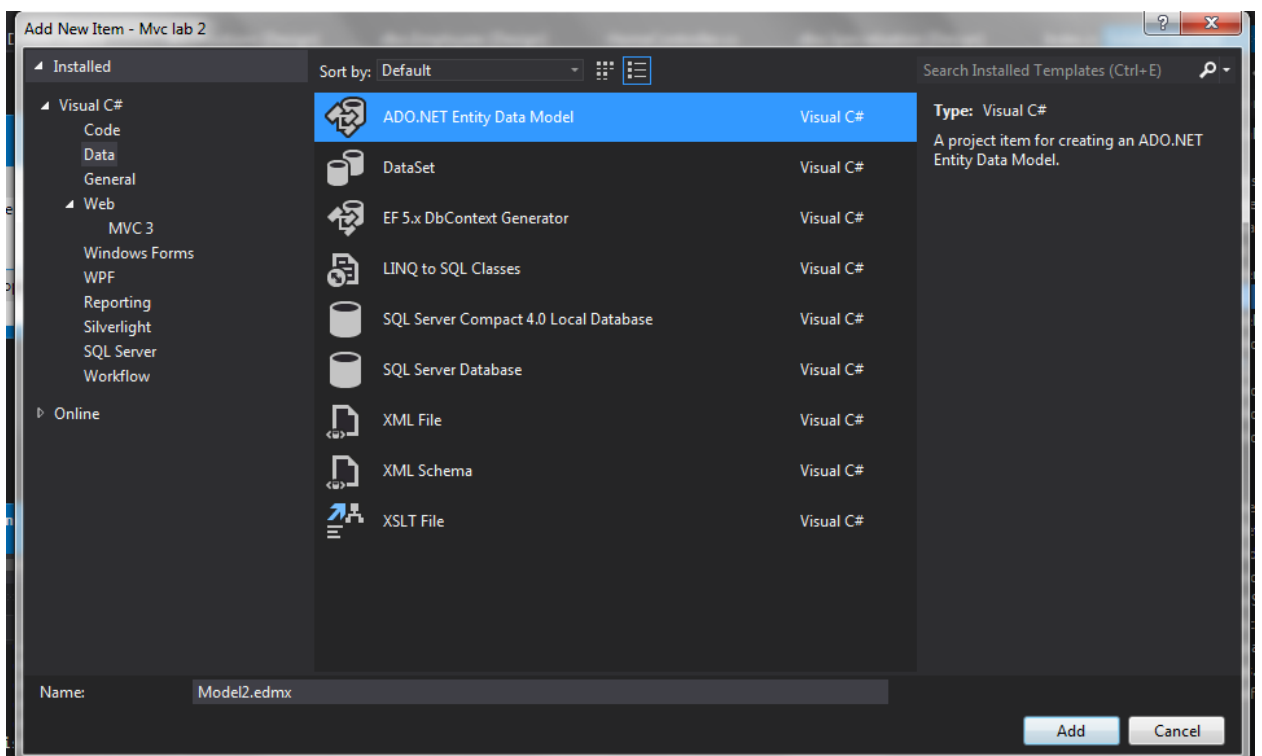


Для виконання запитів на збереження налаштувань створеної таблиці необхідно натиснути кнопку "Update", що знаходиться у верхньому лівому куті робочої області. Важливо приділити максимальну увагу при налаштуванні первинних та зовнішніх ключів у таблиці, адже від цього залежить коректність роботи вашого web- додатку.

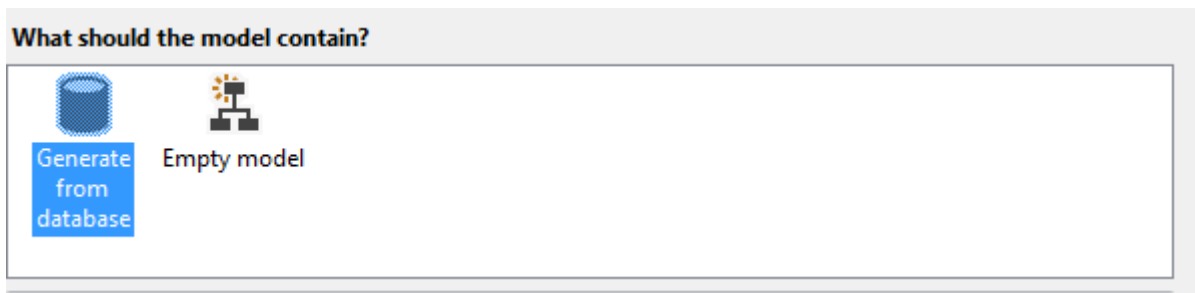
Заповнюємо створені таблиці тестовими даними за допомогою опції "Show table data".



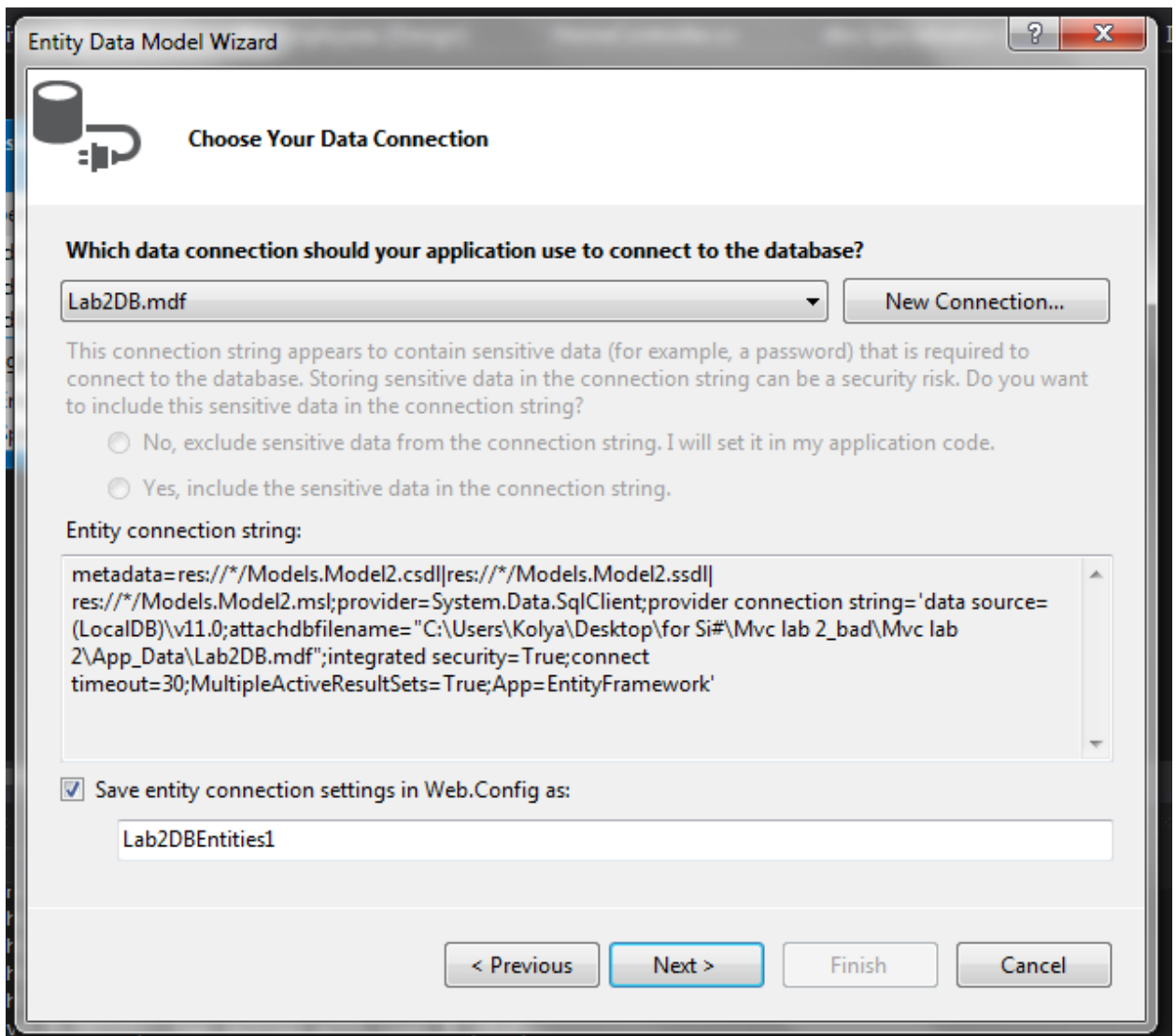
6. Для налагодження роботи між контролером та даними що зберігаються у БД необхідно створити модель, яка виступить цим зв'язуючим прошарком. Для цього на теці "Models" визиваємо контекстне меню і додаємо нову Entity Data Model.



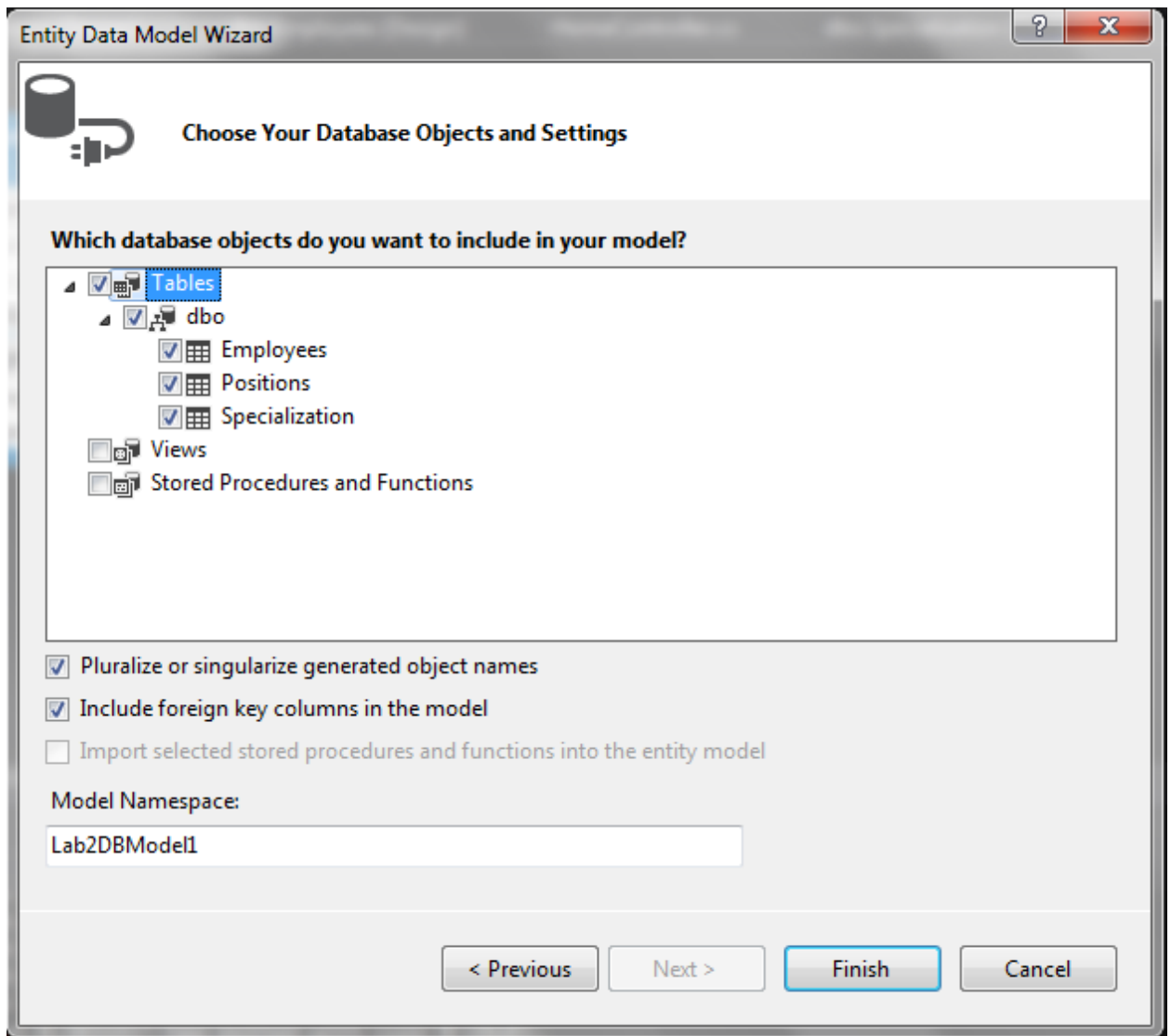
Далі виконуємо наступні дії:



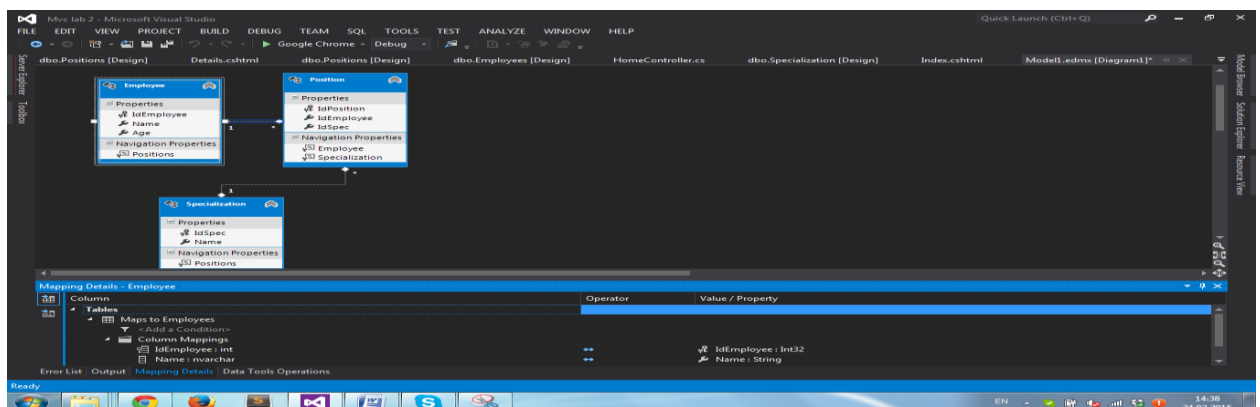
Обираємо наше підключення:



Обираємо всі таблиці, обираємо підключення зовнішніх ключів та генерацію одиничних відмінків у назвах класів.



Після натискання кнопки завершити, отримаємо наступну модель:



Обравши відповідну таблицю та натиснувши «Table mapping» можна побачити у який тип згенерується кожне поле таблиці.

Також було згенеровано класи .Conext.cs, Employee.cs, Position.cs, Specialization.cs, які описують таблиці нашої БД і за допомогою яких ми будемо взаємодіяти з нею через контролер.

```
namespace Mvc_lab_2.Models
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class Lab2DBEntities : DbContext
    {
        public Lab2DBEntities()
            : base("name=Lab2DBEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public DbSet<Employee> Employees { get; set; }
        public DbSet<Position> Positions { get; set; }
        public DbSet<Specialization> Specializations { get; set; }
    }
}
```

7. Створимо новий контролер, назвемо його HomeController та додамо туди наступний код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Mvc_lab_2.Models;

namespace Mvc_lab_2.Controllers
{
    public class HomeController : Controller
    {
        private Lab2DBEntities db = new Lab2DBEntities();
        //
        // GET: /Home/

        public ActionResult Index()
        {
            var emps = (from employees in db.Employees select employees).ToList();
            return View(emps);
        }

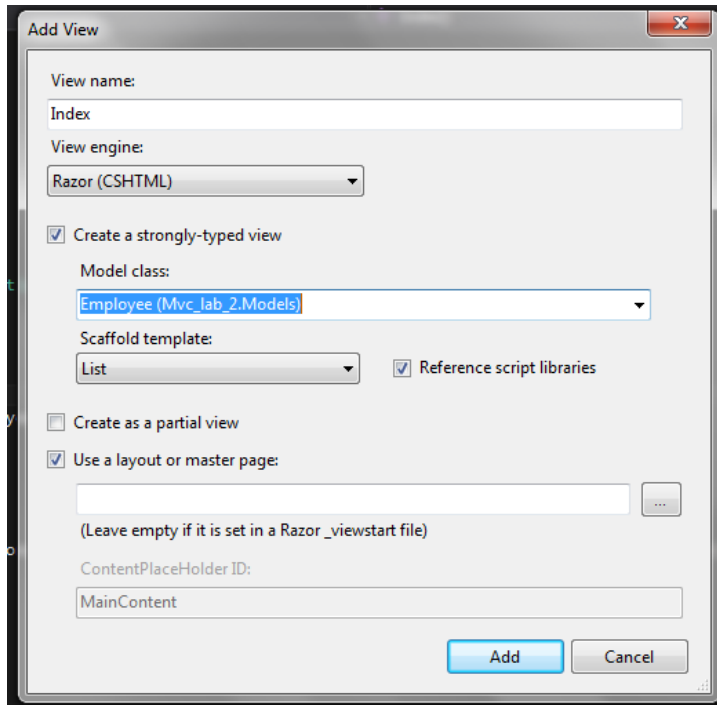
        public ActionResult Details(int id)
        {
            var emps = (from position in db.Positions where position.IdEmployee == id select position.Specialization).First();
            return View(emps);
        }
    }
}
```

В контролері два ActionResult, перший з яких робить запит на отримання всіх співробітників, а другий на спеціалізацію відповідного співробітника. Для побудови запитів використано спеціальну мову LINQ, яка з простою,



зручною, дозволяє проходити по зв'язкам у таблицях без використання Join та будувати значно коротші запити. Необхідно зробити білд проекту.

8. Для першого ActionResult згенеруємо строготипізоване представлення за шаблоном List.



Add View

View name:  
Index

View engine:  
Razor (CSHTML)

☒ Create a strongly-typed view

Model class:  
Employee (Mvc\_lab\_2.Models)

Scaffold template:  
List

☒ Reference script libraries

☐ Create as a partial view

☒ Use a layout or master page:

(Leave empty if it is set in a Razor \_viewstart file)

ContentPlaceHolder ID:  
MainContent

Add Cancel

```
@model IEnumerable<Mvc_lab_2.Models.Employee>

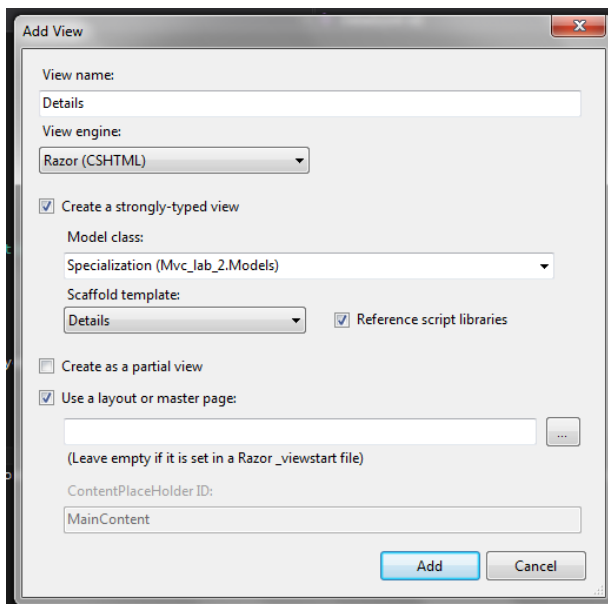
@{
    ViewBag.Title = "Index";
}

<h2>Employees</h2>

<table>
    <tr>
        <th>
            Name
        </th>
        <th>
            Age
        </th>
        <th></th>
    </tr>
    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Age)
            </td>
            <td>
                @Html.ActionLink("Specialization", "Details", new { id=item.IdEmployee })
            </td>
        </tr>
    }
</table>
```

Редагуємо його до такого вигляду.

Для другого ActionResult генеруємо строго типізоване представлення за шаблоном Details:



```
@model Mvc_lab_2.Models.Specialization

@{
    ViewBag.Title = "Details";
}

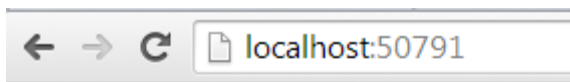
<h2>Details</h2>

<fieldset>
    <legend>Specialization</legend>
    <div class="display-field">
        @Html.DisplayFor(model => model.Name)
    </div>
</fieldset>

<p>
    @Html.ActionLink("Back to List", "Index")
</p>
```

Редагуємо його до такого вигляду.

9. Для коректного запуску даного веб-додатку необхідно зробити білд проекту, після завершення білду запустити проект адресний рядок браузера повинен мати наступний вигляд:



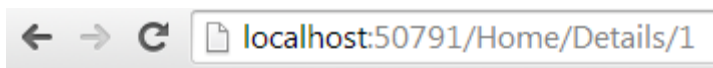
## Employees

Name Age

Peter 21 [Specialization](#)

Kate 65 [Specialization](#)

Натискаємо кнопку "Specialization"



## Details

**Specialization**

Seller

[Back to List](#)

Як бачимо web-додаток працює коректно.

### Вимоги до виконання роботи:

- Реалізувати додаток згідно варіанту
- Підготувати звіт до програми, який буде містити:
  - зміст завдання
  - текст додатку з коментарями
  - результати перевірки коректності роботи додатку на різних значеннях параметрів (у тому числі і помилкових)
  - формулювання призначення моделі, представлення та контролера.

### Варіанти завдань:

Створити 3 локальні таблиці (назви брати згідно варіанту з таблиці нижче), кожна таблиця має містити не менше 3 полів. Заповнити створені таблиці тестовими даними, мінімум по 3 записи у кожній таблиці. Створити MVC додаток, який буде виводити дані згідно варіанту (дані вказані в таблиці нижче).

Перелік варіантів:

№ варіанта	Назва таблиць	Вихідні дані
1	Товар, Товар-Група, Група	Перелік груп товарів, а біля назви кожної групи буде посилання на перелік усіх товарів даної групи
2	Телефонні номери, Номер-Контакт, Контакт	Перелік контактів, а біля назви кожного контакту буде посилання на перелік усіх телефонних номерів даного контакту
3	Модель автомобіля, Модель-Марка, Марка автомобіля	Перелік марок автомобілів, а біля назви кожної марки буде посилання на перелік усіх моделей даної марки.
4	Тип принтера, Тип принтера – Модель, Модель принтера	Вивести всі лазерні принтери однієї марки
5	Тип ПК, Марка ПК, Тип-Марка	Вивести найдорожчий ПК одного з типів (наприклад, найдорожчий ноутбук)
6	Марка телефону, Ціна телефону, Марка - Ціна	Вивести 5 найдешевших телефонів.
7	Тип корабля, Тип – Назва, Назва корабля	Вивести всі кораблі одного типу з переліком усіх назв цього типу.
8	Тип магазину, Тип – Назва, Назва магазину	Вивести назви всіх супермаркетів.
9	Тип потягу, Тип – Маршрут, Назва маршруту	Вивести всі типи потягів по одному маршруту.
10	Тип зброї, Тип – об'єм магазину, Об'єм магазину	Вивести типи зброї з об'ємом магазину більше 16.
11	Дата вильоту, Дата – Рейс, Рейси літаків	Вивести всі рейси літаків за сьогоднішню

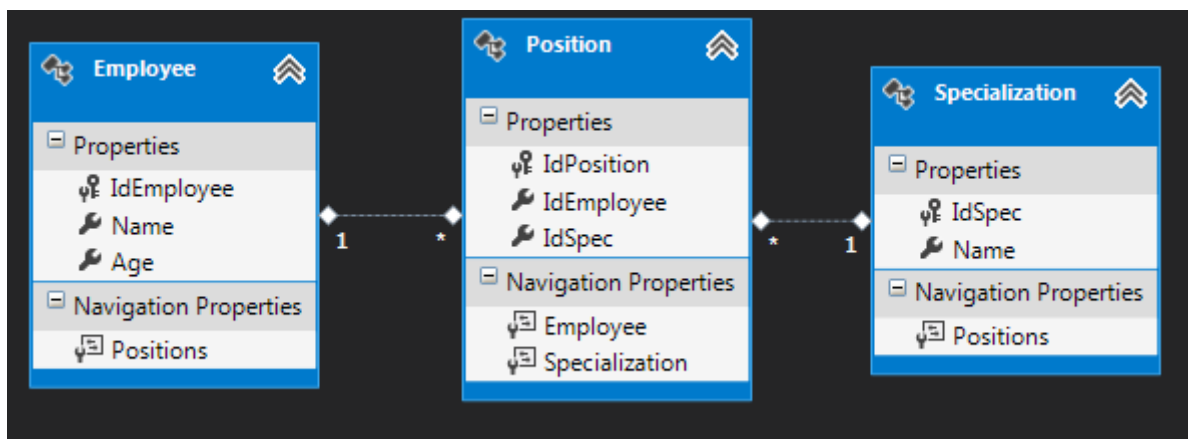
		дату.
12	Назва предмету, Предмет – Викладач, Імена Викладачів	Вивести всі предмети одного викладача.
13	Назва виконавця, Виконавець – Пісня, Назва Пісні	Вивести по 2 пісні кожного виконавця
14	Тип велосипеда, Тип – Марка, Марка велосипеда	Вивести всі марки велосипедів з посиланням на їх тип
15	Район розміщення – Район – назва, Назва ресторану	Вивести всі райони з посиланням на ресторани
16	Назва парфуму, Об'єм парфуму, Назва - Об'єм	Вивести всі назви парфумів, об'єм яких більше 50 мл.
17	Тип одягу, Ціна одягу, Ціна – тип	Вивести всі позиції одягу з посиланням на їх ціну.
18	Тип прикраси, Ціна прикраси, Тип – ціна	Вивести 5 найдорожчих прикрас кожного типу.
19	Тип годинника, Марка годинника, Марка – тип	Вивести всі марки годинників з посиланням на їх типи.
20	Автор, Автор – Назва, Назва книги	Вивести всі твори одного автора.
21	Тип фотоапарату, Модель фотоапарату, Тип - Модель	Вивести всі моделі фотоапаратів з посиланнями на їх типи.
22	Тип музичного інструменту, Ціна, Ціна-тип	Вивести 3 найдешевших інструменти.
23	Назва навчального закладу, Тип НЗ, назва - тип	Вивести всі коледжі.
24	Марка аудіосистеми, Тип аудіосистеми, марка - тип	Вивести всі аудіосистеми з посиланням на їх тип.
25	Тип квартири, Ціна квартири, тип - ціна	Вивести найдорожчу 3-кімнатну квартиру.
26	Назва міста, Площа міста, Назва - площа	Вивести найменше місто.
27	Назва телевізора, Ціна	Вивести всі назви

	телевізора, Назва - ціна	телевізорів з посиланням на їх ціни.
28	Назва курорту, Місце розташування, Назва - місце	Вивести місце з найбільшою кількістю курортів.
29	Тип рослини, Назва рослини, назва - тип	Вивести всі назви рослин з посиланням на їх тип.
30	Група тварин, порода тварин, група - порода	Вивести всі породи однієї групи.

## Частина 2

На основі таблиці, яка містить данні про Співробітників, створити MVC додаток, який буде виводити весь перелік співробітників, дозволить створювати нових співробітників, редагувати існуючих, видаляти співробітників та здійснювати валідацію даних при виконанні операцій створення та редагування сутностей.

1. Відкриємо проект-приклад, що був створений. В цьому проекті було згенеровано модель, по заздалегідь створеній БД, та реалізовано необхідні екшини та представлення для відображення даних, що зберігаються у таблицях БД.



Структура БД.

```

namespace Mvc_lab_2.Models
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class Lab2DBEntities : DbContext
    {
        public Lab2DBEntities()
            : base("name=Lab2DBEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public DbSet<Employee> Employees { get; set; }
        public DbSet<Position> Positions { get; set; }
        public DbSet<Specialization> Specializations { get; set; }
    }
}

```

Контекст БД.

```

public ActionResult Index()
{
    var emps = (from employees in db.Employees select employees).ToList();
    return View(emps);
}

public ActionResult Details(int id)
{
    var emps = (from position in db.Positions where position.IdEmployee == id select position.Specialization).First();
    return View(emps);
}

```

Екшини HomeController для відображення даних БД.

```

@model IEnumerable<Mvc_lab_2.Models.Employee>

@{
    ViewBag.Title = "Index";
}

<h2>Employees</h2>

<table>
<tr>
<th>@Html.ActionLink("+", "Create", new { })</th>
<th>
        Name
    </th>
<th>
        Age
    </th>
</tr>

@foreach (var item in Model) {
<tr>
<td>
        @Html.DisplayFor(modelItem => item.Name)
    </td>
<td>
        @Html.DisplayFor(modelItem => item.Age)
    </td>
<td>
        @Html.ActionLink("Specialization", "Details", new { id=item.IdEmployee })
    </td>
</tr>
}
</table>

```

Представлення для відображення даних таблиці «Співробітники»

Далі саме над даними таблиці «Співробітники» будуть проводитися операції створення, редагування, видалення (CRUD) та валідації вводу даних.

2. Додамо у наш раніше створений HomeController екшини для створення нової сутності «Співробітник». Цих екшинів повинно бути два, перший для відображення форми вводу даних, який буде викликатися при запиті GET, другий для відправки даних форми на сервер, який буде викликатися при запиті POST.

Код екшину Create для GET:

```

[HttpGet]
public ActionResult Create()
{
    Employee emp = new Employee();
    return View(emp);
}

```



Код екшину Create для POST:

```
[HttpPost]
public ActionResult Create(Employee emp)
{
    try
    {
        if (ModelState.IsValid)
        {
            db.Employees.Add(emp);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(String.Empty, ex);
    }
    return View(emp);
}
```

Створимо строго типізоване представлення типу 'Create' для відображення форми створення нової сутності.

View name: Create

View engine: Razor (CSHTML)

☒ Create a strongly-typed view

Model class: Employee (Mvc\_lab\_2.Models)

Scaffold template: Create ☒ Reference script libraries

☐ Create as a partial view

☒ Use a layout or master page:

(Leave empty if it is set in a Razor \_viewstart file)

ContentPlaceHolder ID: MainContent

Add Cancel

Код представлення Create:

```
@model Mvc_lab_2.Models.Employee

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")" type="text/javascript"></script>

<using (Html.BeginForm()) {
    <Html.ValidationSummary(true)
    <fieldset>
        <legend>Employee</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Age)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Age)
            @Html.ValidationMessageFor(model => model.Age)
        </div>

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

3. Додамо до контролеру екшини та представлення для редагування вже існуючих сутностей. Для цього буде створено також два екшини для запитів GET і POST відповідно.

Код екшину Edit для GET:

```
[HttpGet]
public ActionResult Edit(int id)
{
    var empEdit = (from emp in db.Employees where emp.IdEmployee == id select emp).First();
    return View(empEdit);
}
```

Код екшину Edit для POST:

```

[HttpPost]
public ActionResult Edit(int id, FormCollection collection)
{
    var empEdit = (from emp in db.Employees where emp.IdEmployee == id select emp).First();

    try
    {
        UpdateModel(empEdit);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    catch
    {
        return View(empEdit);
    }
}

```

Створимо строго типізоване представлення типу 'Edit' для відображення форми редагування сутності.

The 'Add View' dialog box is shown with the following configuration:

- View name: Edit
- View engine: Razor (CSHTML)
- ☒ Create a strongly-typed view
  - Model class: Employee (Mvc\_Jab\_2.Models)
  - Scaffold template: Edit
  - ☒ Reference script libraries
- ☐ Create as a partial view
- ☒ Use a layout or master page:
  - ContentPlaceHolder ID: MainContent

The 'Add' button is highlighted in blue.

Код представлення Edit:

```

@model Mvc_lab_2.Models.Employee

@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>

<script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
<script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")" type="text/javascript"></script>

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Employee</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Age)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Age)
            @Html.ValidationMessageFor(model => model.Age)
        </div>

        <p>
            <input type="submit" value="Save" />
        </p>
    </fieldset>
}

```

4. При створенні нових та редагуванні вже існуючих сутностей користувач може ввести в форму некоректні данні або не зрозуміти, які данні потрібно вводити, такі дії можливо призведуть до помилки та виникненню виключної ситуації, для попередження таких дій користувача необхідно накласти певні обмеження на данні, що вводяться користувачем, тобто перевіряти валідність введених даних.

Для реалізації валідації необхідно до папки Models додати новий клас з назвою «EmployeeValidation». В цьому файлі буде реалізований частковий клас Employee, на поля якого будуть накладені обмеження за допомогою спеціальних атрибутів.

Код EmployeeValidation.cs:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
using System.Web.Mvc;
using System.ComponentModel;

namespace Mvc_lab_2.Models
{
    [MetadataType(typeof(EmployeeMetaData))]
    public partial class Employee
    {
        [Bind(Exclude="IdEmployee")]
        public class EmployeeMetaData {

            [ScaffoldColumn(false)]
            public int IdEmployee { get; set; }

            [DisplayName("Name")]
            [Required(ErrorMessage = "Employee name is required.")]
            [DisplayFormat(ConvertEmptyStringToNull = false)]
            [StringLength(50)]
            public string Name { get; set; }

            [DisplayName("Age")]
            [Required(ErrorMessage = "Age is required.")]
            [DisplayFormat(ConvertEmptyStringToNull = false)]
            [Range(18, 70, ErrorMessage = "Employee age should be in range 18-70 years")]
            public int Age { get; set; }
        }
    }
}

```

За допомогою цих атрибутів можливо задавати певний формат введення, обробляти порожні рядки, контролювати довжину певного рядка та накладати інші необхідні обмеження.

5. Додамо можливість видалення вже існуючих сутностей. Для цього у HomeController буде створено два екзини 'Delete' для запитів GET і POST відповідно.

Код екшини Delete для GET:

```

[HttpGet]
public ActionResult Delete(int id)
{
    var empDelete = (from emp in db.Employees where emp.IdEmployee == id select emp).First();
    return View(empDelete);
}

```

Код екшини Delete для POST:

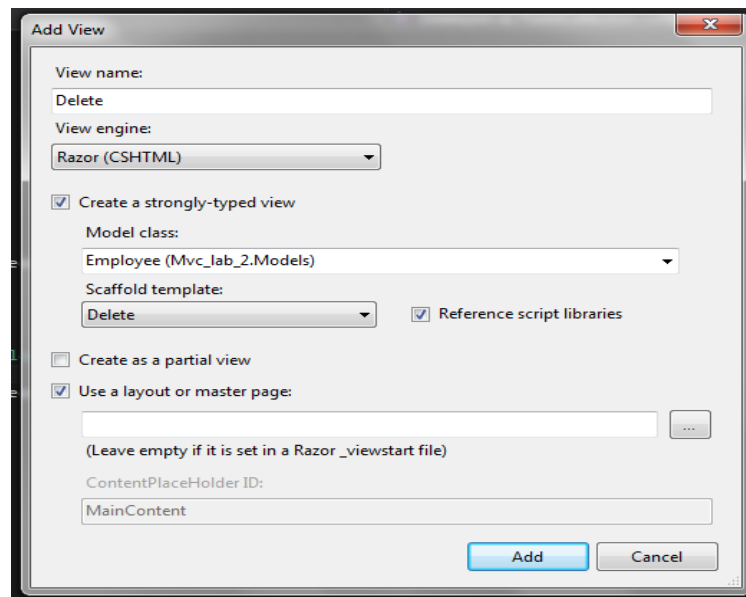
```

[HttpPost]
public ActionResult Delete(int id, FormCollection collection)
{
    var empDelete = (from emp in db.Employees where emp.IdEmployee == id select emp).First();

    try
    {
        db.Employees.Remove(empDelete);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    catch
    {
        return View(empDelete);
    }
}

```

Створимо строго типізоване представлення типу 'Delete' для відображення форми видалення сутності.



Код представлення Delete:

```

@model Mvc_lab_2.Models.Employee

@{
    ViewBag.Title = "Delete";
}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>
<fieldset>
    <legend>Employee</legend>

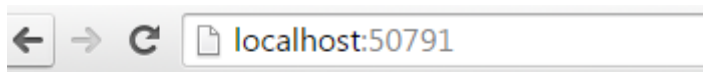
    <div class="display-label">Name</div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Name)
    </div>

    <div class="display-label">Age</div>
    <div class="display-field">
        @Html.DisplayFor(model => model.Age)
    </div>
</fieldset>
@using (Html.BeginForm()) {
    <p>
        <input type="submit" value="Delete" /> |
        @Html.ActionLink("Back to List", "Index")
    </p>
}

```

6. Зробимо білд проекту та запустимо наш додаток.

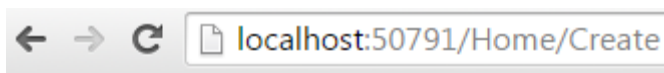
Створимо нового співробітника:



## Employees

[Create new Employee](#)

	Name	Age	
Peter		21	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>
Kate		65	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>



## Create

**Employee**

Name

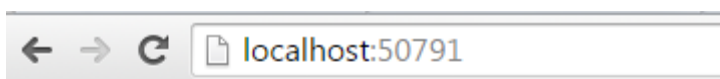
John Uik

Age

51

Create

[Back to List](#)



## Employees

[Create new Employee](#)

Name	Age	
John Uik	51	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>
Peter	21	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>
Kate	65	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>

Нова сутність створена успішно.

Виконаємо редагування існуючої сутності:

## Edit

**Employee**

Name

John Uik

Age

12

Employee age should be in range 18-70 years

Save

[Back to List](#)

Валідація спрацювала на некоректні данні.



← → ↻ localhost:50791

## Employees

[Create new Employee](#)

Name	Age	
John Uik	67	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>
Peter	21	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>
Kate	65	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>

Редагування виконано успішно.

Виконаємо видалення існуючої сутності:

← → ↻ localhost:50791/Home/Delete/0

## Delete

Are you sure you want to delete this?

### Employee

Name  
John Uik  
Age  
67

| [Back to List](#)

← → ↻ localhost:50791

## Employees

[Create new Employee](#)

Name	Age	
Peter	21	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>
Kate	65	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">Specialization</a>

Видалення виконано успішно.

Всі CRUD операції виконуються успішно. Додаток працює коректно.

### **Вимоги до виконання роботи:**

- Реалізувати додаток згідно варіанту
- Підготувати звіт до програми, який буде містити:
  - зміст завдання
  - текст додатку з коментарями
  - результати перевірки коректності роботи додатку на різних значеннях параметрів (у тому числі і помилкових)
  - формулювання призначення моделі, представлення та контролера.

### **Завдання:**

Створити MVC додаток, на основі створеної БД, який окрім виводу, буде давати можливість створювати нові записи до БД, редагувати існуючі та видаляти їх. Також реалізувати валідацію при створенні нових та редагуванні існуючих сутностей.