**ЛАБОРАТОРНА РОБОТА №4**

***Тема.*** Моделі зі складною структурою. Визначення таблиць. Створення логіки додатка. Виконання операцій з моделями - редагування, видалення, додавання. Відношення один до багатьох, багато до багатьох.

***Мета.*** Отримати навикирозробки Web - додатків зі складними моделями. При цьому необхідно враховувати значення навігаційної властивості, наявне у складній моделі також відношення один до багатьох, багато до багатьох.

***Завдання***. 2. Розробити web – додаток «Бібліотека».

**4.2 Хід роботи**

**4.2.1 Моделі зі складною структурою. Відношення один до багатьох**

Розглянемо наступні дві моделі:

public class Book

{

public int Id { get; set; }

public string Name { get; set; }

public int Year { get; set; }

public int? BookId { get; set; }

public Genre Genre { get; set; }

}

public class Genre

{

public int Id { get; set; }

public string Name { get; set; }

public IEnumerable<Book> Books { get; set; }

}

Тут клас Book має властивість Genre, яка показує приналежність книги до певного жанру.

Тепер створимо контекст даних, що використовує моделі:

public class LibraryContext: DbContext

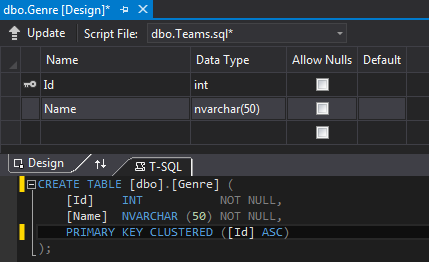
{

public DbSet<Book> Books { get; set; }

public DbSet<Genre> Genres { get; set; }

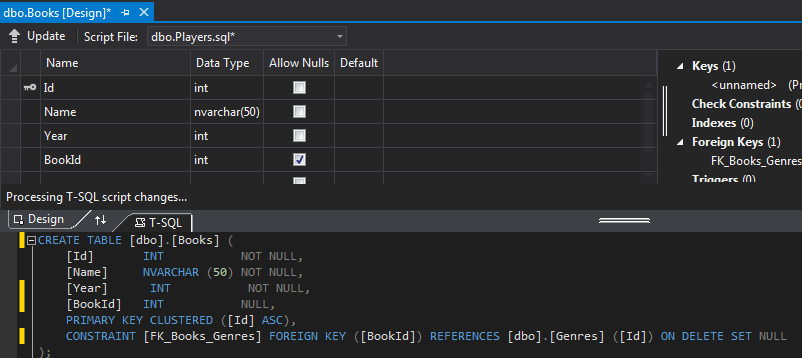
}

Тепер подивимося, як би це все розташовувалося в БД. Припустимо, у нас є деяка база даних Library\_Info.mdf. Нехай наші моделі Book і Genres розташовуються відповідно в таблицях Players і Teams. Визначення таблиці Teams, яка буде зберігати об'єкти моделі Team, виглядає наступним чином (рис.4.1).



**Рис. 4.1 - Визначення таблиці Teams**

Тоді щоб проектувати модель Book на таблицю Books в базі даних, нам треба задати наступне визначення стовпців таблиці (рис.4.2).

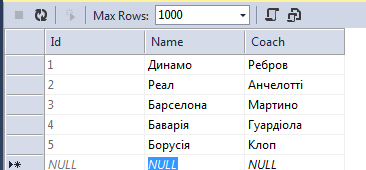


**Рис. 4.2 - Визначення таблиці Players**

Крім того, тут ми задаємо зовнішній ключ - властивість BookId тепер буде посилатися на властивість Id з таблиці Genres. Щоб задати зовнішній ключ, ми додаємо в панелі SQL внизу під дизайнером таблиці наступний рядок:

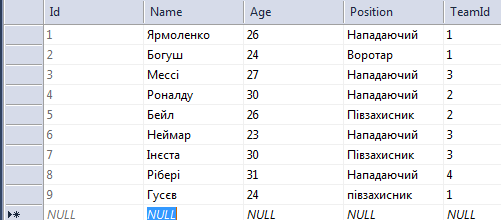
CONSTRAINT [FK\_Bookss\_Genres] FOREIGN KEY ([BookId]) REFERENCES [Genres] ([Id]) ON DELETE SET NULL

Тепер після визначення таблиць наповнимо їх деякими початковими даними. Наприклад додамо деякі дані в таблицю (рис.4.3).



**Рис. 4.3 - Заповнення таблиці Teams**

І в таблицю Players (де стовпець TeamId містить деяке існуюче значення з рядка Id таблиці Teams ) (рис.4.4).



**Рис. 4.4 - Заповнення таблиці Players**

Тепер перейдемо до створення логіки додатка. Додамо в додаток контролер і визначимо в ньому вивід всіх гравців на сторінку:

public class HomeController : Controller

{

//

// GET: /Home/

SoccerContext db = new SoccerContext();

// Виводимо всіх футболістів

public ActionResult Index()

{

var players = db.Players.Include(p => p.Team);

return View(players.ToList()); }

}

За допомогою методу Include фреймворк підвантажує для кожного гравця також і команду, асоційовану з певним гравцем. А при виведенні моделі в представлення Index.cshtml фреймворк буде виводити для кожного гравця назву команди:

@model IEnumerable<NavigationProperty.Models.Player>

@{

ViewBag.Title = "Каталог гравців";

}

<h2>Каталог гравців</h2>

<p>

@Html.ActionLink("Добавити гравця", "Create")

</p>

<table>

<tr>

<th>Ім'я гравця</th>

<th>Вік</th>

<th>Позиція на полі</th>

<th>Команда</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.DisplayFor(modelItem => item.Position)

</td>

<td>

@Html.DisplayFor(modelItem => item.Team.Name)

</td>

<td>

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

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

</td>

</tr>

}

</table>

<p>

@Html.ActionLink("Каталог команд", "ListTeams")

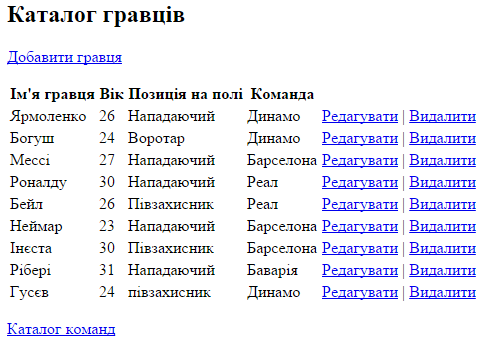
</p>

Оскільки ми в контролері за допомогою методу Include отримали пов'язаний з гравцем об'єкт Team з навігаційного властивості TeamId, то при передачі в контролер всіх об'єктів Player до них також чіпляються пов'язані з ними об'єкти Team, тому ми можемо цілком використовувати в представленні властивість item.Team.Name для отримання імені команди (рис.4.5). Рядок зєднання:

<connectionStrings><add name="SoccerContext" connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename='|DataDirectory|\Soccer\_Info.mdf';Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>



**Рис. 4.5 - Вивід для кожного гравця назву команди**

В моделі Team властивість Players, призначення якої зберігати пов'язаних з командою гравців. Використаємо її. Наприклад, виведемо всі дані про команду, в тому числі про її гравцях. Спочатку додамо в контролер наступний метод:

public ActionResult TeamDetails(int? id)

{

if (id == null)

{

return HttpNotFound();

}

Team team = db.Teams.Find(id);

if (team == null)

{

return HttpNotFound();

}

team.Players = db.Players.Where(m => m.TeamId == team.Id);

return View(team);

}

По-перше, щоб обробити введення при відсутності переданого значення, як параметр використовуємо int? id. По-друге, ми довантажили всіх гравців, пов'язаних з командою, у виразі

team.Players = db.Players.Where(m => m.TeamId == team.Id);

Представлення TeamDetails.cshtml для відображення даних про команду (рис.4.6) може виглядати наступним чином:

@using NavigationProperty.Models

@model Team

@{

ViewBag.Title = "Команда " + @Model.Name;

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<div>

<h4>Команда @Model.Name</h4>

<hr />

<dl>

<dt>Назва</dt>

<dd>

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

</dd>

<dt>Тренер</dt>

<dd>

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

</dd>

<dt>Гравці</dt>

<dd>

<ul>

@foreach (Player player in Model.Players)

{

<li>@player.Name (@player.Position)</li>

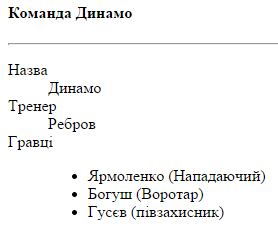
}

</ul>

</dd>

</dl>

</div>



**Рис. 4.6- Гравці пов'язані з командою**

Розглянемо, як виконати операції з моделями - редагування, видалення, додавання. Ці дії не сильно відрізняються від логіки роботи з простими моделями. Єдина відмінність - ми повинні враховувати значення навігаційне властивість, наявну у складній моделі.

Додавання моделі. Розглянемо логіку додавання моделі Player в таблицю Players. Спочатку пропишемо в контролері наступну дію Create:

[HttpGet]

public ActionResult Create()

{

//Формуємо список команд для передачі в представлення

SelectList teams = new SelectList(db.Teams, "Id", "Name");

ViewBag.Teams = teams;

return View();

}

[HttpPost]

public ActionResult Create(Player player)

{

// Додаємо гравця в таблицю

db.Players.Add(player);

db.SaveChanges();

// перенаправляємо на головну сторінку

return RedirectToAction("Index");

}

Перший варіант дії Create обробляє Get-запит і видає представлення, передаючи в нього об'єкт SelectList - список всіх команд. Другий варіант дії Create обробляє Post-запит отримує введену користувачем у представленні модель і додає її в БД (рис.4.7). Розглянемо саме представлення Create.cshtml:

@model NavigationProperty.Models.Player

@{

ViewBag.Title = " Додавання гравця ";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2> Додавання нового гравця </h2>

@using (Html.BeginForm())

{

<fieldset>

<legend>Футболіст</legend>

<p>

Ім’я гравця <br />

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

</p>

<p>

Вік <br />

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

</p>

<p>

Позиція на полі <br />

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

</p>

<p>

Команда <br />

@Html.DropDownListFor(model => model.TeamId, ViewBag.Teams as SelectList)

</p>

<p>

<input type="submit" value="Добавити гравця " />

</p>

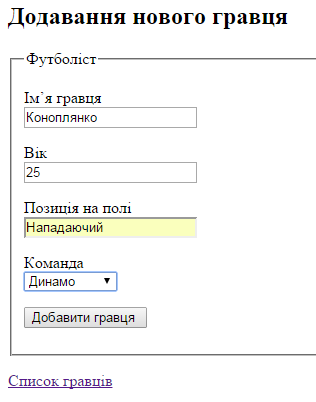
</fieldset>

}

<div>

@Html.ActionLink("Список гравців", "Index")

</div>



**Рис. 4.7- Додавання нового гравця**

Редагування моделі. Редагування працює подібним чином. Визначимо в контролері дії, які відповідають за редагування:

[HttpGet]

public ActionResult Edit(int? id)

{

if (id == null)

{

return HttpNotFound();

}

// Знаходимо в БД футболіста

Player player = db.Players.Find(id);

if (player != null)

{

// Створюємо список команд для передачі в представлення

SelectList teams = new SelectList(db.Teams, "Id", "Name", player.TeamId);

ViewBag.Teams = teams;

return View(player);

}

return RedirectToAction("Index");

}

[HttpPost]

public ActionResult Edit(Player player)

{

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

db.SaveChanges();

return RedirectToAction("Index");

}

Тут ми також формуємо список команд, які ми отримуємо з БД, в об'єкті SelectList. І після отримання запиту на редагування певної моделі Player контролер передає цю модель і список команд в представлення Edit.cshtml:

@model NavigationProperty.Models.Player

@{

ViewBag.Title = "Edit";

}

<h2>Редагування гравця </h2>

@using (Html.BeginForm())

{

<fieldset>

<legend>Футболіст</legend>

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

<p>

Ім’я гравця <br />

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

</p>

<p>

Вік <br />

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

</p>

<p>

Позиція на полі <br />

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

</p>

<p>

Команда <br />

@Html.DropDownListFor(model => model.TeamId, ViewBag.Teams as SelectList)

</p>

<p>

<input type="submit" value="Зберегти" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Повернутися до списку футболістів", "Index")

</div>

Видалення проводиться так, як і у випадку із звичайною моделлю.

**4.4. Контрольні питання**

1. Призначення контролера
2. Звернення до контролера з веб-браузера
3. Методи дій
4. Звернення до методу дій з веб-браузера
5. Опціональний параметр
6. Значення параметрів за замовчуванням
7. Отримання даних з контексту запиту
8. Об'єкт Request
9. Клас ActionResult
10. Створення власних результатів дій
11. Навігаційна властивість
12. Звичайна властивість
13. Зовнішній ключ в моделі зі складною структурою
14. Метод Include
15. Рядок зєднання connectionStrings
16. Призначення властивості Players в моделі Team
17. Створення моделі
18. Видалення моделі
19. Редагування моделі
20. Призначення класу SelectList
21. Віртуальні властивості - Students і Courses
22. Призначення public StudentsContext() : base("DefaultConnection") { }
23. Призначення Database.SetInitializer(new CourseDbInitializer());

**Література:** [1,3,5,7,8,9].