­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­

Лабораторна робота №2

з курсу «**Комп’ютерні технології проектування**»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 14

Виконав студент гр. КНз-3

Чалий Михайло

­­

Львів 2014

## Мета роботи

Ознайомитись з підходом Code First в об’єктно–реляційній проекції Entity Framework для роботи з базами даних в середовищі MS SQL Server та середовищі розробки MS Visual Studio.

## Теоретичні відомості

Entity Framework (EF) – об’єктно-реляційна проекція, що дозволяє .NET розробникам працювати з реляційними базами даних (БД) використовуючи програмні об’єкти. EF дає можливість відмовитись від великої частини коду, яку розробникам потрібно було написати для роботи з БД.

### Підходи Database First і Model First.

Підхід Database First найчастіше використовують коли БД вже розроблена і необхідно розробити програмну логіку для роботи з нею. Використовуючи такі інструменти Visual Studio, як Entity Framework Designer, можна згенерувати C# чи VB.NET класи, що будуть відповідати структурі існуючої БД. Після створення проекції з метою оптимізації моделі можна змінювати зв’язки чи структуру за допомогою дизайнера (чи змінюючи XML файли проекцій). Пріоритетом в даному підході є БД – код і модель виносяться на задній план.

Підхід Model First доцільно використовувати коли ведеться розробка нового проекту (необхідно розробити структуру БД, що буде використовуватись). При цьому підході розробка починається з моделі: використовується Entity Framework Designer для побудови сутностей та зв'язків між ними. З побудованої моделі в середовищі Visual Studio генеруються C# чи VB.NET класи і БД. Пріоритетом в даному підході є модель БД; сама БД і код відходять на задній план.

### Підхід Code First.

Підхід Code First найчастіше використовується коли проект і БД розробляються з нуля. Даний підхід дає можливість відмовитися від використання схем дизайнера чи файлів XML проекцій, натомість БД створюється на основі коду. Програмісти можуть створювати програмні класи для опису потрібних об’єктів, а Entity Framework забезпечить їх використання з БД і моделлю. Entity Framework забезпечує коректну комунікацію і роботу з БД; всі класи, які необхідні для проекції, генеруються автоматично. При використанні цього підходу конфігурація проекції БД не зчитується з XML файлу, натомість використовується об’єкт DbContext, який забезпечує представлення структури БД в пам’яті.

## Завдання

1. Використовуючи підхід Code First cтворити БД згідно з індивідуальним завданням.
2. Використовуючи анотації даних чи API налаштувати валідацію даних та особливості зв’язків у структурі БД. Продемонструвати можливості валідації в коді програми.
3. Організувати аудит полів з різних таблиць в БД і продемонструвати результати в коді програми.
4. Налаштувати використання міграцій. Модифікувати POCO об’єкти і згенерувати як мінімум дві міграції, оновити з їх допомогою БД і продемонструвати роботу зі зміненою структурою.
5. Всі написані та згенеровані скрипти внести в систему контролю версій.

## Реалізація

Лістінг Program.cs

namespace Restaurants

{

using ManyConsole;

using Restaurants.Model;

using System;

using System.Data.Entity;

using System.Linq;

using System.Linq.Expressions;

class Program

{

static int Main(string[] args)

{

var commands = ConsoleCommandDispatcher.FindCommandsInSameAssemblyAs(typeof(Program));

return ConsoleCommandDispatcher.DispatchCommand(commands, args, Console.Out);

}

public class Add : ConsoleCommand

{

public Add()

{

IsCommand("add", "Додати тестовi даннi");

}

public override int Run(string[] remainingArguments)

{

using (var db = new RestaurantEntities())

{

var ua = CreateIfNotExists(db.Countries, \_ => \_.Name == "Україна", () => new Country

{

Name = "Україна",

ISOCode = "UA"

}, "Країна Україна");

var uk = CreateIfNotExists(db.Countries, \_ => \_.Name == "Велика Британiя", () => new Country

{

Name = "Велика Британiя",

ISOCode = "UK"

}, "Країна Велика Британiя");

var lviv = CreateIfNotExists(db.Cities, \_ => \_.Name == "Львiв", () => new City

{

Name = "Львiв",

Country = ua,

}, "Мiсто Львiв");

var fest = CreateIfNotExists(db.RestaurantNetworks, \_ => \_.Name == "FEST", () => new RestaurantNetwork

{

Name = "FEST",

Country = ua

}, "Мережа FEST");

var kopalnia = CreateIfNotExists(db.Restaurants, \_ => \_.Name == "Копальня кави", () => new Restaurant

{

Name = "Копальня кави",

RestaurantNetwork = fest,

Address = "Площа ринок",

City = lviv

}, "Ресторан Копальня кави");

var miasoSparv = CreateIfNotExists(db.Restaurants, \_ => \_.Name == "М'ясо i справедливiсть", () => new Restaurant

{

Name = "М'ясо i справедливiсть",

RestaurantNetwork = fest,

Address = "Площа Данили Галицького",

City = lviv

}, "Ресторан М'ясо i справедливiсть");

db.SaveChanges();

}

Console.WriteLine("Тестовi даннi створенi...");

return 0;

}

private static T CreateIfNotExists<T>(DbSet<T> set,

Expression<Func<T, bool>> predicate,

Func<T> factory, string description)

where T : class

{

var entity = set.FirstOrDefault(predicate);

if (entity == null)

{

entity = set.Add(factory());

Console.WriteLine("Створено сутнiсть " + description);

}

else

{

Console.WriteLine("Сутнiсть " + description + " вже iснує");

}

return entity;

}

}

public class View : ConsoleCommand

{

public View()

{

IsCommand("view", "Дивитись даннi");

}

public override int Run(string[] remainingArguments)

{

using (var db = new RestaurantEntities())

{

Console.WriteLine("Ресторани Львова");

var lviv = db.Cities.First(\_ => \_.Name == "Львiв");

foreach (var restoraunt in lviv.Restaurants)

{

Console.WriteLine("Ресторан " + restoraunt.Name);

}

}

return 0;

}

}

public class Validate : ConsoleCommand

{

public Validate()

{

IsCommand("validate", "Тестова валiдацiя данних");

}

public override int Run(string[] remainingArguments)

{

using (var db = new RestaurantEntities())

{

var gvadelupa = db.Countries.Add(new Country

{

Name = "Гваделупа",

ISOCode = "Гваделупа"

});

var validationErrors = db.GetValidationErrors()

.Where(vr => !vr.IsValid)

.SelectMany(vr => vr.ValidationErrors);

foreach (var error in validationErrors)

{

Console.WriteLine(error.ErrorMessage);

}

}

return 0;

}

}

public class Audit : ConsoleCommand

{

public Audit()

{

IsCommand("audit", "Тестовий аудит данних");

}

public override int Run(string[] remainingArguments)

{

using (var db = new RestaurantEntities())

{

var miasoSparv = db.Restaurants.First(\_ => \_.Name == "М'ясо i справедливiсть");

var tmpAddress = miasoSparv.Address;

// Change value directly in the DB

using (var db2 = new RestaurantEntities())

{

db.Database.ExecuteSqlCommand("update Restaurants set Address = 'Нова адресса в БД' where Name = 'М''ясо i справедливiсть'");

}

// Change the current value in memory

miasoSparv.Address = "Нова адресса в пам'ятi";

// Dump values

Console.WriteLine("Оригiнальне значення: " + db.Entry(miasoSparv).Property(m => m.Address).OriginalValue);

Console.WriteLine("Поточне значення: " + db.Entry(miasoSparv).Property(m => m.Address).CurrentValue);

Console.WriteLine("Значення з базиданних: " + db.Entry(miasoSparv).GetDatabaseValues().GetValue<string>("Address"));

// Revert back value

miasoSparv.Address = tmpAddress;

db.SaveChanges();

}

return 0;

}

}

#if HASOWNER

public class Owner : ConsoleCommand

{

public Owner()

{

IsCommand("owner", "Дивитись власника мережi");

}

public override int Run(string[] remainingArguments)

{

using (var db = new RestaurantEntities())

{

Console.WriteLine("Власники мережi");

foreach (var rn in db.RestaurantNetworks)

{

Console.WriteLine("Мережа " + rn.Name + ", власник " + rn.Owner);

}

}

return 0;

}

}

#endif

}

}

Лістінг класів моделі

namespace Restaurants.Model

{

using System;

using System.Collections.Generic;

public partial class City

{

public City()

{

this.Restaurants = new HashSet<Restaurant>();

}

public System.Guid Id { get; set; }

public string Name { get; set; }

public virtual Country Country { get; set; }

public virtual ICollection<Restaurant> Restaurants { get; set; }

}

}

namespace Restaurants.Model

{

using System;

using System.Collections.Generic;

public partial class RestaurantNetwork

{

public RestaurantNetwork()

{

this.Restaurants = new HashSet<Restaurant>();

}

public System.Guid Id { get; set; }

public string Name { get; set; }

#if HASOWNER

public string Owner { get; set; }

#endif

public virtual ICollection<Restaurant> Restaurants { get; set; }

public virtual Country Country { get; set; }

}

}

namespace Restaurants.Model

{

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity;

using System.Data.Entity.Migrations;

public class RestaurantEntities : DbContext

{

public RestaurantEntities()

: base("name=RestaurantDatabase")

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Properties()

.Where(p => p.Name == "Id")

.Configure(p => p.IsKey().HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity));

modelBuilder.Properties()

.Where(p => p.PropertyType == typeof(string))

.Configure(p => p.IsVariableLength().HasMaxLength(455));

modelBuilder.Entity<Country>().Property(\_ => \_.ISOCode).IsFixedLength().HasMaxLength(3);

}

public virtual DbSet<City> Cities { get; set; }

public virtual DbSet<Country> Countries { get; set; }

public virtual DbSet<Restaurant> Restaurants { get; set; }

public virtual DbSet<RestaurantNetwork> RestaurantNetworks { get; set; }

}

}

namespace Restaurants.Model

{

using System.Linq;

using System.Data.Entity.Migrations;

public class RestaurantMigrationConfiguration : DbMigrationsConfiguration<RestaurantEntities>

{

public RestaurantMigrationConfiguration()

{

AutomaticMigrationsEnabled = true;

}

protected override void Seed(RestaurantEntities db)

{

#if HASOWNER

foreach (var rn in from rn in db.RestaurantNetworks

where rn.Name == "FEST" && rn.Owner == null

select rn)

{

rn.Owner = "Andriy Khudo";

}

#endif

base.Seed(db);

}

}

}

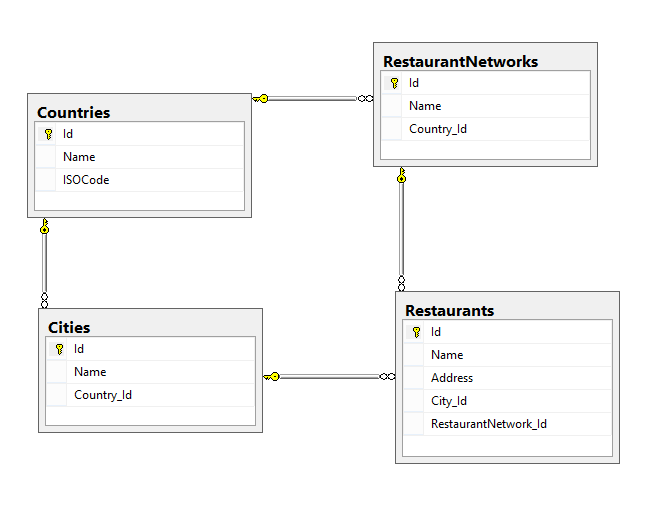


Рисунок . Модель бази данних до міграції

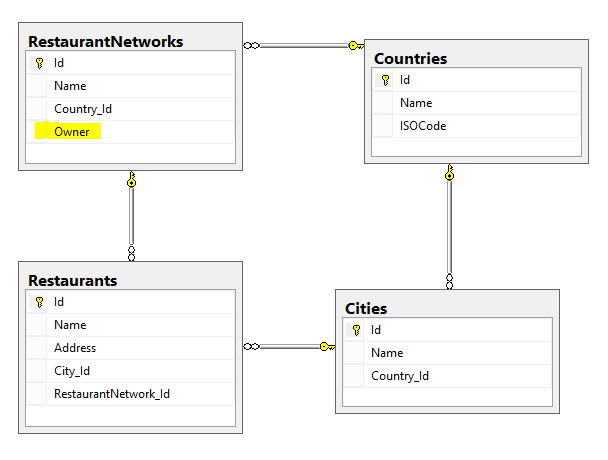


Рисунок . Можель базиданних після міграції

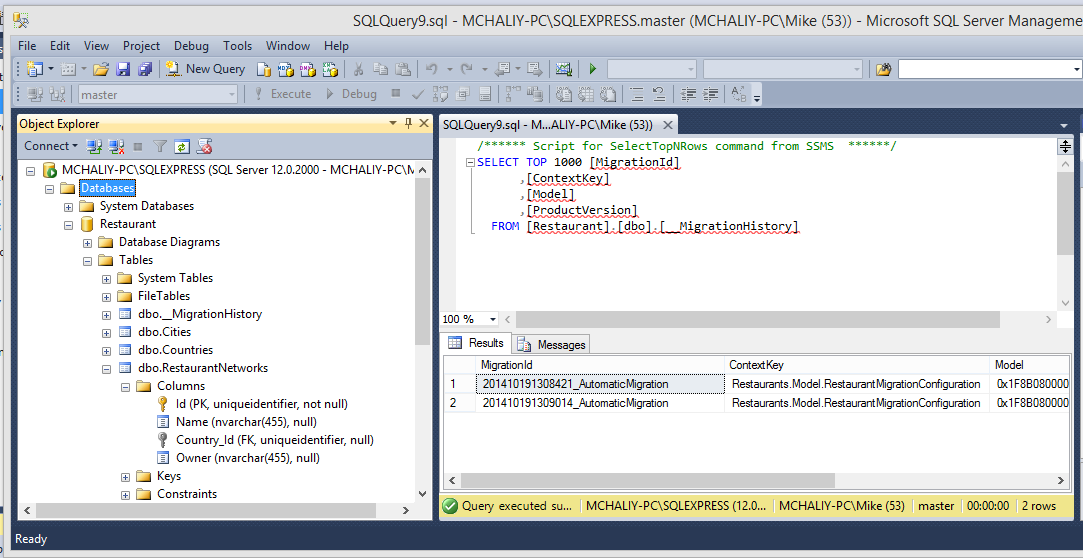


Рисунок . Записи в таблиці міграцій

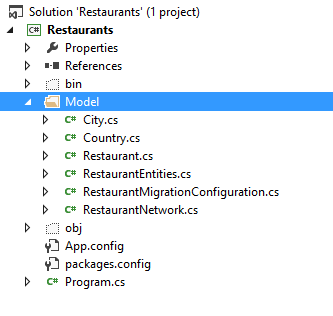


Рисунок . Вигляд солюшен екслорера для проекта

Лістінг компіляції і виконання

git init

msbuild Restoraunts.sln

Restoraunts\bin\debug\Restoraunts.exe add

Restoraunts\bin\debug\Restoraunts.exe view

Restoraunts\bin\debug\Restoraunts.exe validate

Restoraunts\bin\debug\Restoraunts.exe audit

git add -A

git commit –m “Added implementation of the Restoraunts application”

Код на https://github.com/chaliy/studies-octo-adventure/tree/master/lp/c3\_1/trp/l2/

## Результат

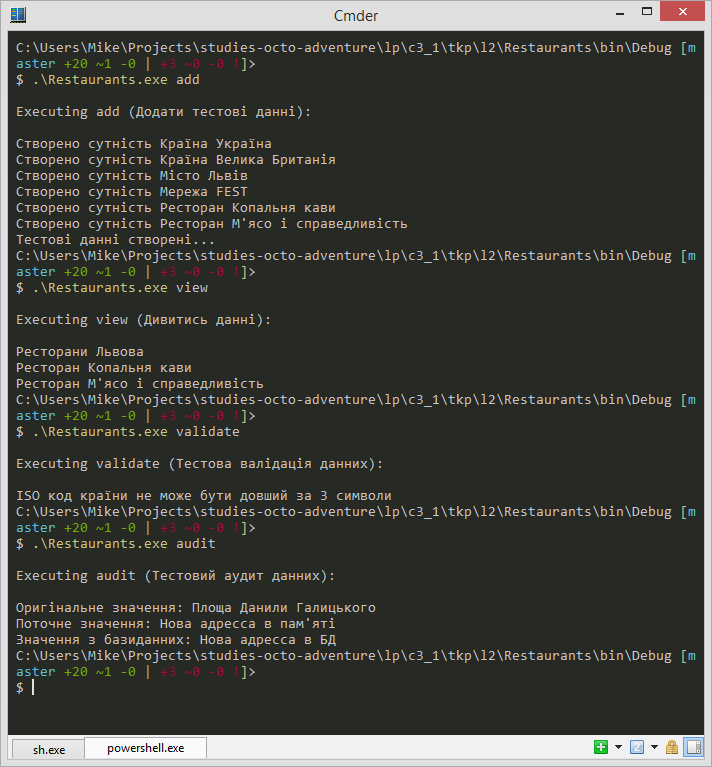


Рисунок . Послідовне виконная комад add, view, validate, audit

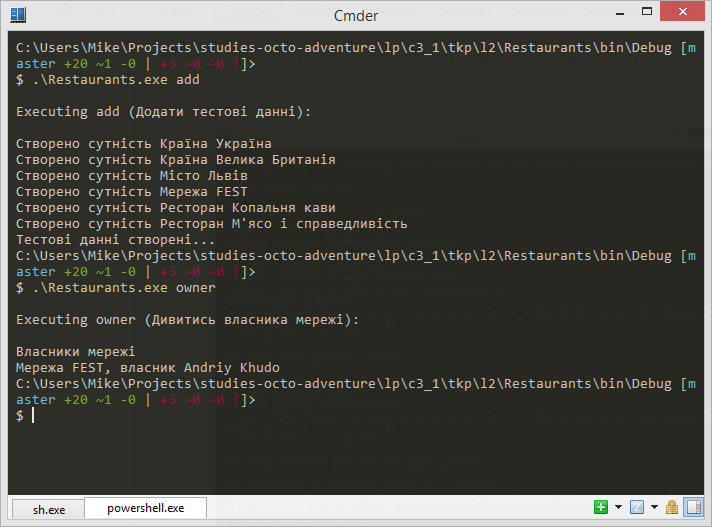


Рисунок . Виконання команд add до міграції, і owner після міграції

## Висновки

Ознайомився з підходом Code First в об’єктно–реляційній проекції Entity Framework для роботи з базами даних в середовищі MS SQL Server та середовищі розробки MS Visual Studio.