# Sprawozdanie z Entity Framework

Magdalena Nowak czwartek 14.40 B

Część z zajęć:

1. Stworzyłam projekt zgodnie z wytycznymi, poniżej zamieszczono kod, który pozwala wygenerować bazę. Od razu zawiera encje potrzebne w realizacji dalszej części projektu – Customers i Orders.

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

using (var db = new ProdContext())

{

var wasCreated = db.Database.CreateIfNotExists();

db.Database.Connection.Open();

}

}

}

public class Category

{

[Key]

public int CategoryId { get; set; }

public string Name { get; set; }

public string Description { get; set; }

public virtual List<Product> Products { get; set; }

}

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

public int UnitsInStock { get; set; }

public int CategoryId { get; set; }

public decimal UnitPrice { get; set; }

}

public class Customer

{

[Key]

public string CompanyName { get; set; }

public string Description { get; set; }

}

public class Order

{

[Key]

public int OrderId {get; set; }

public Customer Customer { get; set; }

public Product Product { get; set; }

public int Quantity { get; set; }

public char Status { get; set; }

}

public class ProdContext : DbContext

{

public DbSet<Category> Categories { get; set; }

public DbSet<Product> Products { get; set; }

public DbSet<Customer> Customers { get; set; }

public DbSet<Order> Orders { get; set; }

}

}

1. Poniżej znajduje się kod będący częścią klasy main, który realizuje dodanie nowego rekordu do tabeli Category

var wasCreated = db.Database.CreateIfNotExists();

db.Database.Connection.Open();

//Create and save a new Category

Console.Write("Enter a name for a new Category: ");

var name = Console.ReadLine();

//zaistancjonuj kategorię o podanej nazwie

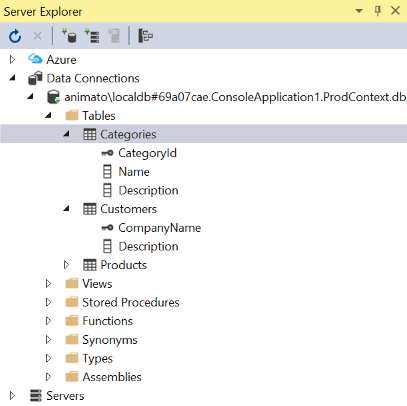
var category = new Category { Name = name };

//dodanie zaistancjonowanego obiektu do kontekstowej kolekcji kategorii

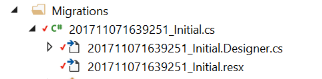
db.Categories.Add(category);

db.SaveChanges();

1. Następnie połączyłam się do mojej lokalnej bazy



1. Następnie włączyłam możliwość migrowania oraz dodałam migracje.



1. Formularz CategoryForm:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data.Entity;

namespace ConsoleApplication1

{

public partial class CategoryForm : Form

{

private ProdContext db = new ProdContext();

public CategoryForm()

{

InitializeComponent();

}

//Load to domyślny Event całego formularza

private void CategoryForm\_Load(object sender, EventArgs e)

{

//ładuje do bufora danych na kliencie z serwera bazy danych

db.Categories.Load();

//połaczenie kontrolki z danymi z Entity Framework

categoryBindingSource.DataSource = db.Categories.Local.ToBindingList();

db.Products.Load();

productBindingSource.DataSource = db.Products.Local.ToBindingList();

}

private void categoryDataGridView\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

//po kliknięciu na komórkę z Categories

string a;

if (e.ColumnIndex == 1)

{

var categoryId = Convert.ToInt32(categoryDataGridView.Rows[e.RowIndex].Cells[0].Value);

filterProducts(categoryId);

}

}

private void filterProducts(int categoryId)

{

//method syntax

var products = db.Products

.Where(product => product.CategoryId == categoryId)

.ToList();

productDataGridView.DataSource = products;

}

private void filterProductsQuery(int categoryId)

{

var query = from p in db.Products

join c in db.Categories on p.CategoryId equals c.CategoryId

where c.CategoryId == categoryId

select p;

List<Product> products = query.ToList<Product>();

}

private void Order\_Add(object sender, EventArgs e)

{

frm\_Order f = new frm\_Order();

DialogResult res=f.ShowDialog(this);

if (res==DialogResult.OK)

{

Order o = f.order;

}

}

private void Order\_Save(object sender, EventArgs e)

{

db.SaveChanges();

this.orderDataGridView.Refresh();

}

private void Category\_Save(object sender, EventArgs e)

{

db.SaveChanges();

this.categoryDataGridView.Refresh();

}

private void productDataGridView\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void categoryDataGridView\_CellContentClick\_1(object sender, DataGridViewCellEventArgs e)

{

}

private void Delete(object sender, EventArgs e)

{

db.SaveChanges();

this.orderDataGridView.Refresh();

}

private void bindingNavigatorDeleteItem\_ForeColorChanged(object sender, EventArgs e)

{

}

private void button3\_Click(object sender, EventArgs e)

{

frm\_Customer f = new frm\_Customer();

DialogResult res = f.ShowDialog(this);

if (res == DialogResult.OK)

{

Customer c = f.customer;

db.Customers.Add(c);

db.SaveChanges();

}

}

}

}

1. Następnie dodałam metody dostępowe.

method based syntax

var categories = db.Categories

.Select(c => c.Name).ToList();

Console.WriteLine("Categories Names:");

foreach (String c in categories)

{

Console.WriteLine(c);

}

//Display all Categories from the database

var query = from b in db.Categories

orderby b.Name descending

select b;

Console.WriteLine("All categories in the database (method syntax):");

foreach (var item in query)

{

Console.WriteLine(item.Name);

}

Console.WriteLine("Products quantity for each category:");

Methods.CountProductsForCategoryQ(db);

CategoryForm f = new CategoryForm();

f.ShowDialog();

Console.WriteLine("Press any key to exit...");

Console.ReadKey();

Utwworzyłam też nową klasę Methods, w której stworzyłam metody wyświetlające wszystkie kategorie i produkty w różnych wariantach (Joiny, Navigation Property, Eager Loading).

Oznaczenie: M w końcówce oznacza, że jest to Method Syntax, zaś Q - Query Syntax

namespace ConsoleApplication1

{

class Methods

{

//METODY DOSTĘPOWE

//Navigation Property - z tabeli Category, po zależnościach dochodzimy do Produktów

public static void PrintCategoriesAndProductsQ(ProdContext db)

{

var query = from b in db.Categories

orderby b.Name descending

select b;

foreach (var categoryName in query)

{

Console.WriteLine("Category name: {0}", categoryName.Name);

foreach (Product product in categoryName.Products)

Console.WriteLine("description: {0}", product.Name);

}

}

//Navigation Properties, wraz z kategorią nazwy produktów z danej kategorii

public static void PrintCategoriesAndProductsM(ProdContext db)

{

IQueryable<Category> query = db.Categories;

foreach (var categoryName in query)

{

Console.WriteLine(categoryName.Name);

foreach (Product product in categoryName.Products)

{

Console.WriteLine("Product: {0}", product.Name);

}

}

}

//Eager loading + Navigation Property

public static void PrintCategoriesAndProductsEagerLoadingQ(ProdContext db)

{

var query = from b in db.Categories.Include("Products")

orderby b.Name descending

select b;

foreach (var categoryName in query)

{

Console.WriteLine("Category name: {0}", categoryName.Name);

foreach (Product product in categoryName.Products)

Console.WriteLine("description: {0}", product.Name);

}

}

//+Eager loading - zapytanie o jeden typ tabeli ładuje od razu także powiązaną tabelę jako część zapytania

//+Navigation Property

public static void PrintCategoriesAndProductsEagerLoadingM(ProdContext db)

{

var categories = db.Categories

.Include(c => c.Products)

//opcjonalnie po nazwie encji Include("Products")

.ToList();

foreach (var record in categories)

{

Console.WriteLine("Category Name: {0}", record.Name);

foreach (var p in record.Products)

{

Console.WriteLine("Product: {0}", p.Name);

}

}

}

//Join

public static void PrintCategoriesAndProductsJoinM(ProdContext db)

{

var query = db.Categories

.Join(db.Products,

product => product.CategoryId,

category => category.CategoryId,

(category, product) =>

new {

c = category,

p = product

});

foreach (var record in query)

{

Console.WriteLine("Category Name " + record.c, "Product Name " + record.p);

}

}

public static void PrintCategoriesAndProductsJoinQ(ProdContext db)

{

var query = from ca in db.Categories

join pr in db.Products

on ca.CategoryId equals pr.CategoryId

//orderby c.Name

select new

{

c = ca,

p = pr

};

foreach (var record in query)

{

Console.WriteLine("Category Name: " + record.c.Name + " Product Name: " + record.p.Name);

}

}

//dodatkowe:

public static void PrintOnlyCategoriesNamesM(ProdContext db)

{

List<String> categoryNames = db.Categories

.Select(c => c.Name)

.ToList();

foreach (var categoryName in categoryNames)

{

Console.WriteLine(categoryName);

}

}

public static void PrintOnlyCategoriesNamesQ(ProdContext db)

{

var query = from c in db.Categories

orderby c.Name descending

select c.Name;

foreach (var categoryName in query)

{

Console.WriteLine(categoryName);

}

}

Metoda zliczająca ilość produktów dla każdej kategorii:

//Agregacja – Count

public static void CountProductsForCategoryQ(ProdContext db)

{

var query = from c in db.Categories

orderby c.Name descending

select new

{

CategoryID = c.CategoryId,

CategoryName = c.Name,

ProductsQuantity = c.Products.Count()

};

foreach (var c in query)

{

Console.WriteLine("Category Name: {0} \t ProductsQuantity: {1}",

c.CategoryName,

c.ProductsQuantity);

}

}

}

}

Część II

Rozszerzenie aplikacji o możliwość składania zamówieć na produkty. Zrealizowano jako rozszerzenie rozpoczętej aplikacji WindowsFormowej.

Dodałam tabelę Orders:

Klasa Order

public class Order

{

[Key]

public int OrderId {get; set; }

public Customer Customer { get; set; }

public Product Product { get; set; }

public int Quantity { get; set; }

public char Status { get; set; }

}

Dodanie zbioru obiektów typu Order

public class ProdContext : DbContext

{

public DbSet<Category> Categories { get; set; }

public DbSet<Product> Products { get; set; }

public DbSet<Customer> Customers { get; set; }

public DbSet<Order> Orders { get; set; }

}

Dodałam nowe formularze Order, dzięki któremu możliwe jest dodanie zamówienia

Na głównym formularzu udostępniłam możliwość dodawania nowego klienta. Po kliknięciu pojawia się nowy formularz do wypełnienia, w którym wpisuje się nazwę firmy oraz opis. Po zatwierdzeniu OK zostaje dodany do bazy nowy rekord Customer, w przypadku kliknięcia Cancel, dodawanie jest anulowane, baza danych się nie zmienia. Formularz ma pole customer, jeśli użytkownik zatwierdzi zmiany, po wróceniu do głównego formularza, czyli po zamknięciu się okienka, można odwołać się do wartości wpisanych w tym polu.

Oto kod wykonujący opisane powyżej działania.

namespace ConsoleApplication1

{

public partial class frm\_Customer : Form

{

public Customer customer;

private Boolean created = false;

public frm\_Customer()

{

InitializeComponent();

}

private void frm\_Customer\_Load(object sender, EventArgs e)

{

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

string q = this.textBox1.Text;

if (!created)

{

this.customer = new Customer { CompanyName = q };

created = true;

}

else

{

this.customer.CompanyName = q;

}

}

private void textBox2\_TextChanged(object sender, EventArgs e)

{

string d = this.textBox2.Text;

if (!created)

{

this.customer = new Customer { Description = d };

created = true;

}

else

{

this.customer.Description = d;

}

}

}

}