

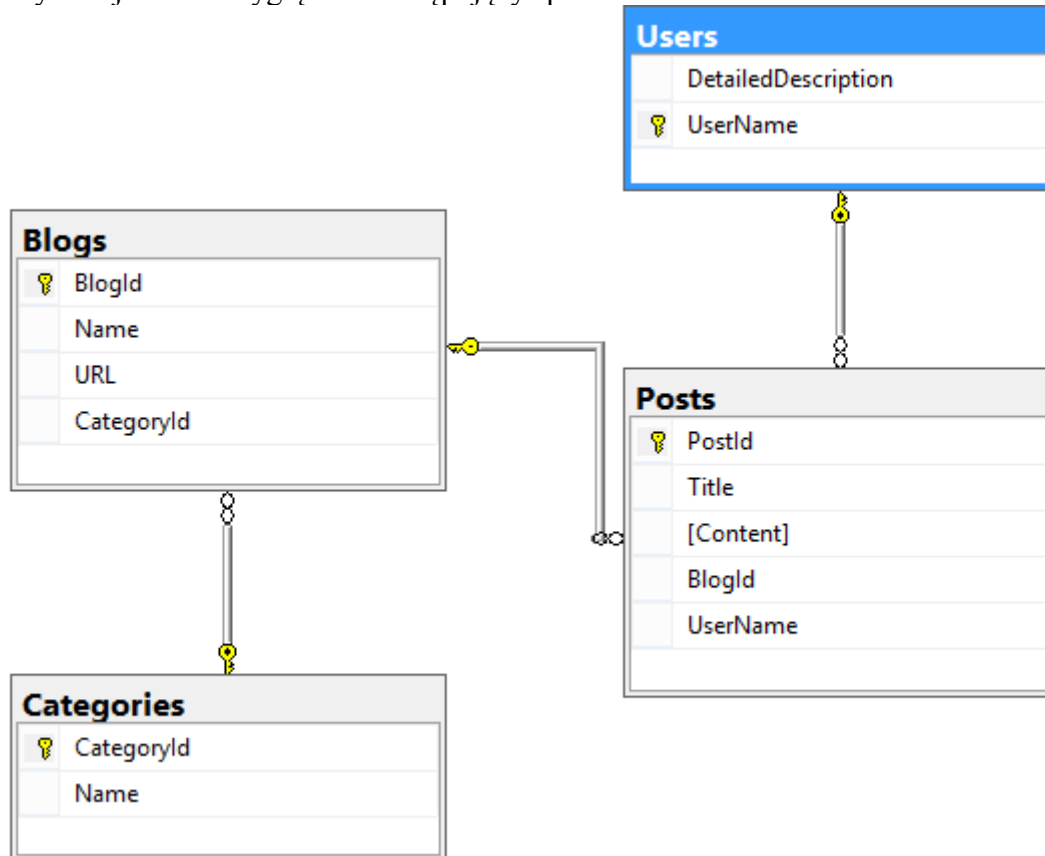
Robert Bielas

Sprawozdanie z zadania domowego (EntityFramework)

1. Postanowiłem rozszerzyć aplikację w następujący sposób:

- Każdy post związałem z jego właścicielem z tabelki User
- Dodałem nową encję <Category> określającą kategorię danego bloga.

Po modyfikacji model wygląda w następujący sposób:



Powiązanie pomiędzy Userem a jego Postami utworzyłem za pomocą Fluent Api:

```
modelBuilder.Entity<Post>()
    .HasRequired(p => p.user)
    .WithMany(u => u.Posts)
    .Map(m => m.MapKey("UserName"));
```

Podobnie postąpiłem w przypadku powiązania pomiędzy Category a jej Blogami:

```
modelBuilder.Entity<Blog>()
    .HasRequired(b => b.category)
    .WithMany(c => c.Blogs)
    .Map(m => m.MapKey("CategoryId"));
```

Stworzyłem tekstowy tryb interaktywny, który pozwala na dynamiczne dodawanie rekordów do bazy. Jest to trzon zadania domowego. To tutaj dokonuję wszystkich zapytań w obu składniach, korzystam z mechanizmu leniwego ładowania pozwalającego sprawdzić czy np. kategoria o podanej nazwie nie została już wcześniej dodana do bazy, bez ładowania encji do obiektu C#-owego, wykorzystuję deferred i imediate loading oraz używam Navigation Properties.

Program wygląda następująco: po uruchomieniu execa pojawia się konsolka listująca zbiór opcji dostępnych dla użytkownika.

```
What do you want to do?
1 - add a Category
2 - add a new User (deffered loading of result)
3 - Add new Post
4 - add new blog
b - select list of blogs
p - select list of posts
c - select list of categories
u - select list of users
<UPArrow> - print info about posts of the particular titles (where contains method syntax)
<DownArrow> - print info about users (query syntax)
i - for each post in db print info about the user who added it and the blog it belongs to (query syntax + Nav Properties)
s - print statistics: for each user print total amount of his posts, for each category print total amount of its blogs, foreach
blog print total amount of its posts (lazy loading)
g - for each blog in db print info about its category and list info about each of its posts(eager loading + navigation
props + method syntax group join)
x - exit:
```

Wprowadzając z klawiatury odpowiedni klawisz przechodzimy do funkcji, która dotyczy odpowiedniego aspektu bazy danych.

W opisach opcji pojawiają się fragmenty w okrągłych nawiasach - dają one pewną intuicję jakie mechanizmy zastosowano w danej funkcji. Jak widać, niektóre opcje są bardziej złożone, ponieważ oprócz zwykłych sql-owych zapytań zawierają inne aspekty oferowane przez framework (jak np. Navigation Properties przy funkcji "i").

Przed wszystkim pojawiają się funkcje dodające różne elementy do bazy. Moja baza po częściowym ręcznym wypełnieniu wygląda następująco:

1	use [codefirst.Program+blogcontext]
2	
3	select * from blogs
4	select * from posts
5	select * from users
6	select * from categories

100 %

ResultsMessages

	BlogId	Name	URL	CategoryId
1	50	Wiki	NULL	1
2	51	Wookie	NULL	2

	PostId	Title	Content	BlogId	UserName
1	43	JanMiodek	NULL	50	Rob
2	44	Sparta	NULL	50	Basia
3	45	BennyHill	NULL	51	Rob
4	46	MiodoweLata	NULL	51	Basia

	DetailedDescription	UserName
1	NULL	Bart
2	NULL	Basia
3	NULL	Rob
4	NULL	User

	CategoryId	Name
1	1	General
2	2	Humor
3	3	Beer

Wstawianie zapobiega powstawaniu redundancji. Za każdym razem, kiedy element ma być wstawiony do bazy, dokonuję zapytania sql sprawdzający czy blog/ kategoria/użytkownik o podanej nazwie lub post o danym tytule nie jest już wstawiony do bazy. Sprawdzania dokonuję w obu syntaxach, przykładowo:

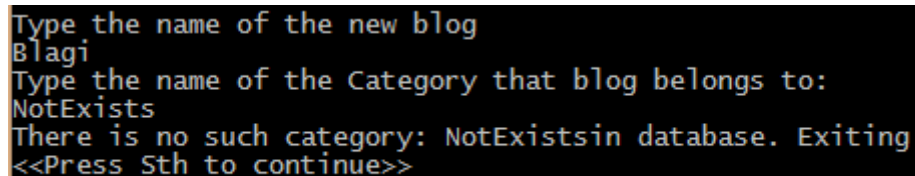
```
Console.WriteLine("Type the title of the Post:");
var title = Console.ReadLine();
var check = from p in db.posts
            where title == p.Title
            select p;
if (check.Count() != 0)
{
    Console.WriteLine("There already is the post of title " + title);
    return;
}
```

oraz:

```
Console.WriteLine("Type the name of the User:");
var name = Console.ReadLine();

var check = db.users.Select(u => u).Where(u => u.UserName == name);
if (check.Count() != 0)
{
    Console.WriteLine("There already is the user of name " + name);
}
```

W przypadku wstawiania elementu powiązanego kluczem obcym z innym elementem w bazie także dokonuję testu czy obiekt istnieje - jeśli tak to dodaję odpowiednie elementy do właściwych kolekcji, w przeciwnym wypadku, kiedy na jakimś etapie nie można utworzyć połączenia (np przyporządkować posta użytkownikowi, gdy ten nie istnieje) to robimy "rollbacka" i wracamy do konsoli startowej widocznej powyżej.



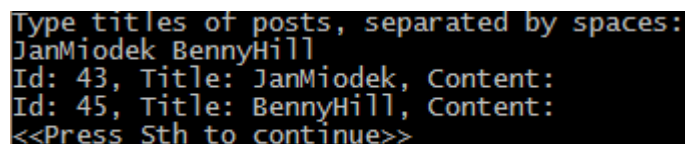
```
Type the name of the new blog
Błagi
Type the name of the Category that blog belongs to:
NotExists
There is no such category: NotExists in database. Exiting
<<Press Sth to continue>>
```

Za wstawianie służą funkcje add\*(db);

2. Wykorzystanie obu składni L2E.

- Postanowiłem do listy opcji dodać możliwość szczegółowego opisanie danej encji. Użytkownik pisze np. do których postów pragnie otrzymać opis: "Post1 post2 post3" po wybraniu z głównego menu strzałki w dół. Implementacja wygląda następująco (przykład dla query syntax):

```
Console.WriteLine("Type titles of posts, separated by spaces: ");
var titles = Console.ReadLine().Split();
var posts = db.posts.Where(b => titles.Contains(b.Title));
foreach (var post in posts)
{
    post.printSelf();
}
```



```
Type titles of posts, separated by spaces:
JanMiodek BennyHill
Id: 43, Title: JanMiodek, Content:
Id: 45, Title: BennyHill, Content:
<<Press Sth to continue>>
```

lub (przykład dla method syntax):

```
Console.WriteLine("Type user names, separated by spaces: ");
var userNames = Console.ReadLine().Split();
var users = from u in db.users
             where userNames.Contains(u.UserName)
             select u;

foreach (var user in users)
{
    user.printSelf();
}
```

```
Type user names, separated by spaces:
Rob Basia
UserName: Basia, Description:
UserName: Rob, Description:
<<Press Sth to continue>>
```

- Użycie Join w query syntax (opcja "i", wynik częściowy bo się nie zmieściło):

```
=====
PostInfo:
Id: 44, Title: Sparta, Content:
UserInfo:
UserName: Basia, Description:
BlogInfo:
Id: 50, Name: Wiki, URL:
=====

=====

PostInfo:
Id: 45, Title: BennyHill, Content:
UserInfo:
UserName: Rob, Description:
BlogInfo:
Id: 51, Name: Wookie, URL:
=====

=====

PostInfo:
Id: 46, Title: MiodoweLata, Content:
UserInfo:
UserName: Basia, Description:
BlogInfo:
Id: 51, Name: Wookie, URL:
=====
<<Press Sth to continue>>
```

```
var posts = from u in db.users
             join p in db.posts
             on u.UserName equals p.user.UserName
             join b in db.blogs
             on p.BlogId equals b.BlogId
             select p;
foreach(var post in posts)
{
    Console.WriteLine("\n=====\n");
    Console.WriteLine("PostInfo:");
    post.printSelf();
}
```

```

Console.WriteLine("\nUserInfo:");
post.user.printSelf();
Console.WriteLine("\nBlogInfo:");
post.blog.printSelf();
Console.WriteLine("\n=====\n");

```

```

}

```

- Użycie GroupJoin w method syntax:

```

=====

Blog:
Id: 50, Name: Wiki, URL:

Category of the blog:
Id: 1, Name: General

Blog's posts' list:
Id: 43, Title: JanMiodek, Content:
Id: 44, Title: Sparta, Content:

Total Amount of posts: 2

=====

=====

Blog:
Id: 51, Name: Wookie, URL:

Category of the blog:
Id: 2, Name: Humor

Blog's posts' list:
Id: 45, Title: BennyHill, Content:
Id: 46, Title: MiodoweLata, Content:

Total Amount of posts: 2

=====

<<Press Sth to continue>>

```

```

var blog_group = db.blogs.(...)
    .GroupJoin(db.posts,
        b => b.BlogId,
        p => p.BlogId,
        (b,g) => new {blog = b, posts = g}
    );

```

3. Wykorzystanie mechanizmów lazy i eager loading.

- Wykorzystanie lazy loading:

Przykładowo w funkcji drukującej statystyki bazy danych:

```

db.Configuration.LazyLoadingEnabled = true;
//Only users
IEnumerable<User> users = db.users.ToList<User>();
foreach (User user in users)
{

```

```

    var postsAmount = db.Entry(user).Collection(u =>
u.Posts).Query().Count<Post>();
    Console.WriteLine("User: {0} has written: {1} posts", user.UserName,

```

```

postsAmount);
    }
    Console.WriteLine("\n");
    //Only categories
    IList<Category> categories = db.categories.ToList<Category>();
    foreach (Category cat in categories)
    {
        var blogsAmount = db.Entry(cat).Collection(c =>
c.Blogs).Query().Count<Blog>();
        Console.WriteLine("Category: {0} has: {1} posts",cat.Name, blogsAmount);

    }
    Console.WriteLine("\n");
    //Only blogs for now
    IList<Blog> blogs = db.blogs.ToList<Blog>();
    foreach (Blog blog in blogs)
    {
        var postsAmount = db.Entry(blog).Collection(b =>
b.Posts).Query().Count<Post>();
        Console.WriteLine("Blog: {0} has: {1} posts",blog.Name, postsAmount);
    }

```

- wykorzystanie eager loading:

Przykładowo w już widzianej powyżej funkcji blogInfoGroupJoin:

```

var blog_group = db.blogs.Include("Posts").Include("category")
//here we have eager loading of all blog's posts - since we need to load them anyway - and
its category

```

```

        .GroupJoin(db.posts,
            b => b.BlogId,
            p=> p.BlogId,
            (b,g) => new {blog = b, posts = g}
        );
    foreach(var blog_struct in blog_group)
    {
        Console.WriteLine("\n=====\n");
        Console.WriteLine("\nBlog: ");
        blog_struct.blog.printSelf();
        Console.WriteLine("\nCategory of the blog: ");
        blog_struct.blog.category.printSelf();
        Console.WriteLine("\nBlog's posts' list: ");
        foreach(var post in blog_struct.posts)
        {
            post.printSelf();
        }
        Console.WriteLine("\nTotal Amount of posts: {0}",
blog_struct.posts.Count());
        Console.WriteLine("\n=====\n");
    }

```

Najważniejsze z kodu powyżej jest wykorzystanie metody Include - skoro i tak musimy wypisać wszystkie elementy listy Posts dla konkretnego bloga, to od razu dołączamy je do aplikacji, a nie czekamy na moment drukowania(widoczny po groupjoinie).

4.Wykorzystanie mechanizmów deferred i immediate execution.

- Wykorzystanie deffered execution:

```

var users = from u in db.users
            select u;
// deffered lading - query is here, before adding new user to db...
db.users.Add(new User { UserName = name });
db.SaveChanges();
Console.WriteLine("User " + name + " added successfully, Situation at the
moment:");
        foreach (var user in users) user.printSelf();
// but he will be printed automagically here anyway

```

Komentarze powyżej dokładnie opisują co się dzieje: w funkcji addUser, jeśli nie nastąpiła redundancja, to tworzymy zapytanie w zmiennej users. Przed iterowaniem po zapytaniu dodajemy nowo utworzonego użytkownika do bazy. Pomimo tego że użytkownik został dodany po utworzeniu zapytania, pojawia się on w kolekcji users:

```
type the name of the User:
New
User New added successfully, Situation at the moment:
UserName: Bart, Description:
UserName: Basia, Description:
UserName: New, Description:
UserName: Rob, Description:
UserName: User, Description:
<<Press Sth to continue>>
```

- wykorzystanie immediate execution:

```
var amountOfCategories = (from c in db.categories
                          select c).Count();
// ^ everything is immediately loaded, so our category won't be included here...
Console.WriteLine("There are {0} categories in the db", amountOfCategories);
//...because we add it here and not before Count()
db.categories.Add(new Category { Name = name });
db.SaveChanges();
Console.WriteLine("Category " + name + " added successfully");
```

```
Type the name of the Category:
BrandNew
There are 3 categories in the db
Category BrandNew added successfully
<<Press Sth to continue>>
```

W momencie kiedy wykonałem powyższą komendę, w bazie były tylko kategorie: General, Humor, Beer.

Widać, że funkcje agregujące dokonują operacji na kolekcjach bazodanowych nie czekając na nic. Podobnie byłoby przy skorzystaniu z metody toList() na zapytaniu.

## 5. Wykorzystanie Navigation Properties.

Przykładowo:

```
var blog_group = db.blogs.Include("Posts").Include("category")
                    .GroupJoin(db.posts,
                                b => b.BlogId,
                                p => p.BlogId,
                                (b,g) => new {blog = b, posts = g}
                    );
foreach(var blog_struct in blog_group)
{
    Console.WriteLine("\n=====\n");
    Console.WriteLine("\nBlog: ");
    blog_struct.blog.printSelf();
    Console.WriteLine("\nCategory of the blog: ");
    blog_struct.blog.category.printSelf();
    Console.WriteLine("\nBlog's posts' list: ");
    foreach(var post in blog_struct.posts)
    {
        post.printSelf();
    }
    Console.WriteLine("\nTotal Amount of posts: {0}",
blog_struct.posts.Count());
    Console.WriteLine("\n=====\n");
}
```

Uwaga na NavigationProperty blog\_struct.blog i jej metodę printSelf().