Zpravodaj moderníProgramování 03/2012: Tipy pro LINQ to SQL

Předpoklad ke studiu tohoto čísla: celá učebnice pro pokročilé (zelená).

LINQ to SQL je způsob práce s relační databází prostřednictvím LINQ operátorů integrovaných do programovacího jazyka, jak je vysvětlováno v poslední kapitole učebnice pro pokročilé. V tomto čísle Zpravodaje se podíváme na pár tipů k této technologii. Další, i komplikovanější otázky ohledně LINQ to SQL připravím do některého z příštích čísel Zpravodaje.

LINQ metoda Single

Na úvod si osvětlíme význam metody `Single`, která slouží k výběru jediného prvku sekvence a kde nás nečekají žádné záludnosti.

Abychom měli s čím pracovat, vytvořte si pod MS SQL Express Edition novou databázi s jedinou tabulkou `Zaměstnanci`, jejíž schéma je na obrázku:

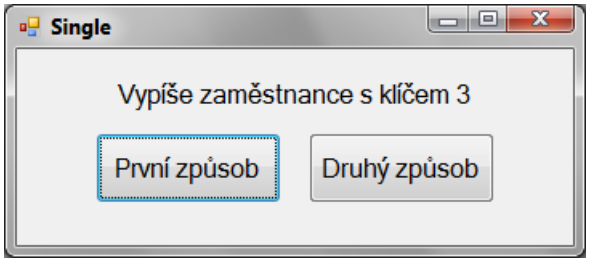
	Column Name	Data Type	Allow Nulls
🔑	Klíč	int	☐
	Příjmení	nvarchar(50)	☐
	Plat	float	☐

Tabulku naplňte nějakými daty, například:

Klíč	Příjmení	Plat
1	Petřík	12000
2	Pavlát	50000
3	Maličká	25000

Pomocí O/R návrháře dále proveďte přípravu pro LINQ to SQL (`LINQ.dbml`). **Nenechte se zlákat případnou nabídkou ke zkopírování databáze do projektu!!!**

Metoda `Single` se typicky používá, když chceme vybrat prvek podle primárního klíče. Otestujme ji v následujícím programu:



```
private void tlačítkoPrvníZpůsob_Click(object sender, EventArgs e)
{
    var linq = new LINQDataContext();
    Zaměstnanci zaměstanecČíslo3 =
        linq.Zaměstnanci.Single(zam => zam.Klíč == 3);
    MessageBox.Show(zaměstanecČíslo3.Příjmení);
}

private void tlačítkoDruhýZpůsob_Click(object sender, EventArgs e)
{
    var linq = new LINQDataContext();
    Zaměstnanci zaměstanecČíslo3 =
        linq.Zaměstnanci.Where(zam => zam.Klíč == 3).Single();
    MessageBox.Show(zaměstanecČíslo3.Příjmení);
}
```

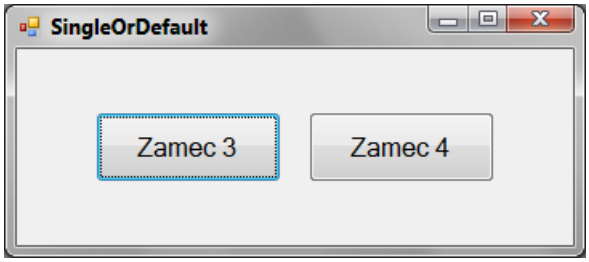
Vidíte, že buď můžeme volat `Single` s výběrovým kritériem, anebo nejdřív `Where` s kritériem a potom `Single` bez parametrů. Samotné `Where` nestačí, `Where` vrací sekvenci (byť i jednoprvkovou), my potřebujeme konkrétní prvek.

V reálné situaci uživatel vybere (např. v listboxu) prvek, který ho zajímá, my víme jeho klíč a výše uvedeným kódem jej vytáhneme z databáze.

Pokud by prvek s daným klíčem neexistoval (nebo se vyskytoval vícekrát, to by ale DB neměla dovolit), volání `Single` způsobí běhovou chybu. Může-li taková situace nastat, je třeba ji ošetřit přes `try-catch`.

SingleOrDefault

Osobně mne občas `try-catch` dost otravuje, především kvůli tomu, že všechny ty závorky a slova zaberou hromadu řádků, které se netýkají podstaty řešeného problému. Alternativou je metoda `SingleOrDefault`, která v případě, že objekt v DB není, vrátí `null` namísto vyvolání běhové chyby (při použití na sekvenci čísel by vrátila nulu, nikoli `null`, samozřejmě). Zkuste následující program:




```
private void tlačítkoZamec3_Click(object sender, EventArgs e)
{
    ZobrazZaměstnance(3);
}

private void tlačítkoZamec4_Click(object sender, EventArgs e)
{
    ZobrazZaměstnance(4);
}
```

```
private void ZobrazZaměstnance(int klíčVybraného)
{
   var linq = new LINQDataContext();
   Zaměstnanci vybranýZaměstanec =
     linq.Zaměstnanci.SingleOrDefault(zam => zam.Klíč == klíčVybraného);
   if (vybranýZaměstanec != null)
     MessageBox.Show(vybranýZaměstanec.Příjmení);
   else
     MessageBox.Show("Zaměstnanec nenalezen");
}
```

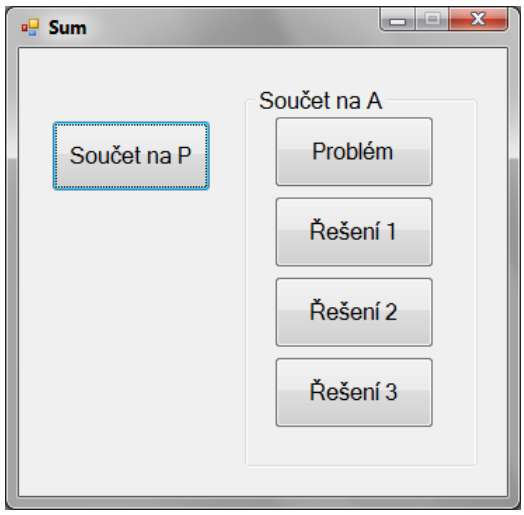
Upozorním na jednu možná trochu zradu se `SingleOrDefault`. V případě, že v sekvenci je více než jeden prvek zadaného kritéria, nevrátí metoda `null`, nýbrž vyvolá běhovou chybu. Pokud vytahujete prvek dle primárního klíče, tak se vám to nestane, DB to nedovolí, je však třeba na to myslet při použití `SingleOrDefault` v jiných situacích (kdy aplikace očekává jednu hodnotu, kvůli nějaké nekonzistenci v datech jsou tam ale hodnoty dvě).

Married

Když je v LINQ metoda `Single`, měla by tam být i metoda `Married`, ne? Z nějakého důvodu tam ale není :)

Sum

Pojďme se podívat na vysčítání sekvence. Vypočteme součet platů všech zaměstnanců, jejichž příjmení začíná na P:




```
private void tlačítkoSoučetP_Click(object sender, EventArgs e)
{
   var linq = new LINQDataContext();
   double součet = linq.Zaměstnanci.
                   Where(zam => zam.Příjmení.StartsWith("P")).
                   Sum(zam => zam.Plat);
   MessageBox.Show(součet.ToString());
}
```

Petřík s Pavlátem dají dohromady 62 000, vše OK.

Pokud ale změníme počáteční písmeno z P na A, dojde k běhové chybě (s velmi podivnou chybovou hláškou):

```
private void tlačítkoProblém_Click(object sender, EventArgs e)
{
   // Jediná změna oproti tlačítkuSoučetP je "A" namísto "P"
   var linq = new LINQDataContext();
   double součet = linq.Zaměstnanci.
                   Where(zam => zam.Příjmení.StartsWith("A")).
                   Sum(zam => zam.Plat);
   MessageBox.Show(součet.ToString());
}
```

Jaká je příčina? Metoda `Sum` si v databázi neumí poradit s prázdnou sekvencí. Nemáme žádného zaměstnance na A. Je to nepříjemné – proč mám řešit, jestli vysčítávaná sekvence je prázdná či nikoli? Nicméně je tomu tak, podívejme se, jak si s tím poradit.

Řešení 1: Výslovně testujeme, jestli je sekvence prázdná.

```
private void tlačítkoŘešení1_Click(object sender, EventArgs e)
{
   var linq = new LINQDataContext();
   var výběr =
       linq.Zaměstnanci.Where(zam => zam.Příjmení.StartsWith("A"));
   double součet = 0;
   if (výběr.Count() > 0)
       součet = výběr.Sum(zam => zam.Plat);
   MessageBox.Show(součet.ToString());
}
```

Teoreticky by ve víceuživatelské aplikaci mohl nastat problém, pokud by mezi voláními `Count` a `Sum` jiný uživatel vymazal všechny existující zaměstnance na A.

Řešení 2: Využijeme toho, že metoda `Sum` havaruje na prázdné sekvenci pouze v databázi, nikoli při provádění v paměti aplikace:

```
private void tlačítkoŘešení2_Click(object sender, EventArgs e)
{
   var linq = new LINQDataContext();
   double součet = linq.Zaměstnanci.
                   Where(zam => zam.Příjmení.StartsWith("A")).
                   ToArray().
                   Sum(zam => zam.Plat);
   MessageBox.Show(součet.ToString());
}
```

Samozřejmě se ale nyní bude z DB serveru přenášet mnohem více dat (celá sčítaná sekvence).

Řešení 3: Téhož efektu jako v řešení 2 se docílí uložením výběru do `IEnumerable`, nikoli do `var` (= v tomto případě `IQueryable`):

```
private void tlačítkoŘešení3_Click(object sender, EventArgs e)
{
   var linq = new LINQDataContext();
   IEnumerable<Zaměstnanci> výběr =
       linq.Zaměstnanci.Where(zam => zam.Příjmení.StartsWith("A"));
   double součet = výběr.Sum(zam => zam.Plat);
   MessageBox.Show(součet.ToString());
}
```

Překlad na SQL příkazy

V učebnici je ukázáno, jakým způsobem lze zobrazit SQL příkazy, na něž jsou překládány volání LINQ metod. Poté, co jsme se ve Zpravodaji 01/2011 seznámili s konzolovými aplikacemi, pojďme se podívat ještě na alternativní způsob zobrazení generovaných SQL příkazů.

V projektu zkoumajícím metodu `Sum`:

1. Za každé instancování řídicí třídy LINQ vložte řádek:

   ```
   linq.Log = Console.Out;
   ```

2. Ve vývojovém prostředí vyberte nabídku *Project* > *(jméno_proj) Properties*. V kartě Application nastavte „Output type“ na „Console Application“.

Prozkoumejte, co DB provádí po stisku každého tlačítka!

Radek Vystavěl, 1. března 2012

Pokud Vám Zpravodaje moderníProgramování připadají užitečné, doporučte jejich odběr svým známým. Mohou se přihlásit na webu www.moderniProgramovani.cz.