

Jazyk C# 2

7. seminář

Večeřa J., Janoščík R.

Univerzita Palackého v Olomouci

4.4.2023

Reakce na úkoly (1/2)

- 2x stejný string s dotazem, kdy tedy použít parametry a kdy je nemusíme použít?

```
1 SqlCommand add = new SqlCommand("INSERT INTO students (OsCislo, Jmeno, Prijmeni, UserName, Rocnik, OborKomb)
    VALUES (@OsCislo, @Jmeno, @Prijmeni, @UserName, @Rocnik, @OborKomb);", conn);
2 add.Parameters.Add(new SqlParameter("OsCislo", "R007"));
3 add.Parameters.Add(new SqlParameter("Jmeno", "James"));
4 add.Parameters.Add(new SqlParameter("Prijmeni", "Bond"));
5 add.Parameters.Add(new SqlParameter("UserName", "JB7"));
6 add.Parameters.Add(new SqlParameter("Rocnik", 12));
7 add.Parameters.Add(new SqlParameter("OborKomb", "APLINF"));
8 int aff = add.ExecuteNonQuery();
9
10 Console.WriteLine($"Affected after adding: {aff}");
11 SqlCommand add2 = new SqlCommand("INSERT INTO students (OsCislo, Jmeno, Prijmeni, UserName, Rocnik,
    OborKomb) VALUES (@OsCislo, @Jmeno, @Prijmeni, @UserName, @Rocnik, @OborKomb);", conn);
12 add2.Parameters.Add(new SqlParameter("OsCislo", "R17002"));
13 add2.Parameters.Add(new SqlParameter("Jmeno", "Karel"));
14 add2.Parameters.Add(new SqlParameter("Prijmeni", "Rizek"));
15 add2.Parameters.Add(new SqlParameter("UserName", "rizkka02"));
16 add2.Parameters.Add(new SqlParameter("Rocnik", 3));
17 add2.Parameters.Add(new SqlParameter("OborKomb", "APLINF"));
```

Reakce na úkoly (2/2)

- Nepoužití metod
- Limitace výstupu přímo v kódu, nikoliv v SQL

```
1 var i = 0;
2 var command = new SqlCommand("SELECT DISTINCT Jmeno, Prijmeni FROM
    students ORDER BY Prijmeni ASC;", conn);
3 using (var dr = command.ExecuteReader()) {
4     while (dr.Read()) {
5         if (i >= 2 && i < 8)
6             Console.WriteLine($"{dr[0]}, {dr[1]}");
7         i++;
8     }
9 }
```

● vs.

```
1 SqlCommand ukoll = new SqlCommand("SELECT DISTINCT Jmeno, Prijmeni FROM
    students ORDER BY Prijmeni OFFSET 2 ROWS FETCH NEXT 6 ROWS ONLY;",
    conn);
```

Object-relational mapping (ORM)

- Automatické mapování objektů do relační databáze
- Náš pohled – transparentní práce s databází pomocí objektů
- Odstínění od konkrétní implementace databáze (MSSQL, MySQL, PostgreSQL, ...)
- Dva přístupy
 - ▶ `CodeFirst` – Databáze se generuje z programu
 - ▶ `DatabaseFirst` – Programový kód se generuje dle struktury DB
- Zde ukážeme pouze `CodeFirst`

Entity Framework

- První verze (2008) součástí .NET Frameworku, nad LINQ to SQL
- Od verze 6 (2013) Opensource, mimo .NET Framework
- <https://docs.microsoft.com/en-us/ef/>
- Dvě verze:
 - ▶ EF6 - Frameworková <https://github.com/dotnet/ef6>
 - ▶ EF Core – pro .NET Core <https://github.com/dotnet/efcore>
- CodeFirst i DatabaseFirst
- Líné vyhodnocování
- Podpora LINQ
- Podpora MSSQL, Oracle, MySQL, ... (<https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli>)

EF – instalace – NuGet Package Manager

- Ruční linkování knihoven pracné (verze, závislosti, hledání knihoven v různých zdrojích)
- NuGet Package Manager – správce balíčků s repozitáři (podobně jako v Linuxu)
- Pravý klik na projekt → Manage NuGet Packages
- Správa verzí, licencí, závislostí
- Možno více zdrojů
- Možno vytvářet vlastní balíčky
- Balíčky „per projekt“ nebo „per solution“
 - ▶ Microsoft.EntityFrameworkCore
 - ▶ Microsoft.EntityFrameworkCore.Design
 - ▶ Microsoft.EntityFrameworkCore.Proxies
 - ▶ Microsoft.EntityFrameworkCore.Sqlite

EF – vytvoření modelu

- = definice tříd, které chceme uchovávat v databázi
- **Pozor na CodeFirst konvence** [https://msdn.microsoft.com/en-us/library/jj679962\(v=vs.113\).aspx](https://msdn.microsoft.com/en-us/library/jj679962(v=vs.113).aspx) např.
 - ▶ Property obsahující „id“ bude primární klíč
 - ▶ Odkaz na jinou entitu pomocí `virtual + typ + název`
 - ▶ (Předchozí zdroj pro EF6, pro EFCore <https://docs.microsoft.com/en-us/ef/core/modeling/relationships>)
- Např.:

```
1 public class Customer {
2     public int Id {get; set;}
3     public string Name {get; set;}
4     public int AddressId {get; set;}
5     public virtual Address Address {get; set;}
6 }
```

EF – vytvoření kontextu

- Objekt reprezentující připojení k databázi

```
1 using Microsoft.EntityFrameworkCore
```

- Předek: `DbContext`
- Pro každou entitu `DbSet`
- Např.:

```
1 public class EshopContext : DbContext
2 {
3     public EshopContext () : base ("EshopConnection")
4     { }
5     public DbSet<Address> Addresses { get; set; }
6     public DbSet<Customer> Customers { get; set; }
7 }
```

- Lze dodefinovat názvy sloupců pomocí atributů

```
1 [Column ("jmeno_zakaznika")]
2 public string Name { get; set; }
```

- ...

Vytvoření databáze

- **Můžeme potřebovat EF nástroje pro dotnet**

```
dotnet tool install --global dotnet-ef
```

- **Případně dodat do PATH**

```
export PATH="$PATH:/home/radek/.dotnet/tools"
```

- **Přidání úvodní migrace (viz dále)**

```
dotnet ef migrations add InitialCreate
```

- **Vytvoření databáze**

```
dotnet ef database update
```

Přístup k datům – vložení záznamu

- Vytvoření záznamu

```
1 try {
2     using (EshopContext ctx = new EshopContext()) {
3         Address a = new Address() { City = "Olomouc", Street = "17.
           Listopadu", Number = 14 };
4         Customer c = new Customer() { Address = a, Name = "Karel
           Vomacka" };
5         ctx.Custommers.Add(c);
6         ctx.SaveChanges();
7     }
8 } catch (Exception e) {
9     Console.WriteLine(e);
10 }
```

- Vždy potřeba uložit změny `ctx.SaveChanges()`
- Automatické vložení/uložení všech dotyčných entit

Přístup k datům – čtení dat

- LINQ dotaz na DbSet

```
1 try {  
2     using (EshopContext ctx = new EshopContext()) {  
3         foreach (Customer c in ctx.Customers.Where(p  
4             =>p.Address.City == "Olomouc" )) {  
5             Console.WriteLine($"{c.Name}, {c.Id}");  
6         }  
7 } catch (Exception e) {  
8     Console.WriteLine(e);  
9 }
```

- Zobrazení vygenerovaných dotazů:

```
1 options.UseLoggerFactory(LoggerFactory.Create(b => {  
2     b.AddConsole(); }));
```

Změna modelu – migrace (1/2)

- Při změně modelu (definice tříd) musíme „protlačit“ změnu i do databáze
- Systém verzování databáze a *migrací*
- Při automatickém založení databáze nám vznikla i tabulka `_EFMigrationHistory`, která zaznamenává změny v DB.
- Zjistili jsme, že jsme zapomněli na město

```
1 public class Address {
2     public int Id { get; set; }
3     public string Street { get; set; }
4     public int Number { get; set; }
5     public string City { get; set; }
6 }
```

Změna modelu – migrace (2/2)

- Změnu musíme dostat do databáze
- Zadáme příkaz: `dotnet ef migrations add City` (City je název dané migrace)
- a aktualizujeme databázi: `dotnet ef database update`
- Databáze se aktualizovala, dodal se sloupec pro město

Vazba 1:N

- K adrese pouze doplníme:

```
public virtual ICollection<Student> Students { get; set; }
```

- Pokud jsou vazby zřejmé, máme automaticky k dispozici seznam studentů na dané adrese
- Jinak potřeba dodat atribut [`InverseProperty("Address")`] se specifikací inverzní vlastnosti
- Případně pomocí FluentAPI

<https://docs.microsoft.com/en-us/ef/core/modeling/> `DbContext`

```
1 protected override void OnModelCreating(ModelBuilder modelBuilder) {  
2     modelBuilder.Entity<Student> ()  
3         .HasOne(s => s.Address)  
4         .WithMany(a => a.Students);  
5 }
```

- Priority: Konvence, atributy, FluentAPI

Vazba M:N

- V předchozích verzích stačilo vytvořit navigační seznamy, případně dodefinovat ve FluentAPI
- V EFCore musíme ručně vytvořit mezilehlou entitu v modelu

```
1 public class StudentSubject {  
2     public int StudentId { get; set; }  
3     public virtual Student Student { get; set; }  
4     public int SubjectId { get; set; }  
5     public virtual Subject Subject { get; set; } nny  
6 }
```

- A dodefinovat navigační property:

```
1 protected override void OnModelCreating(ModelBuilder modelBuilder) {  
2     modelBuilder.Entity<StudentSubject>().HasKey(s => new {s.StudentId, s.SubjectId});  
3     modelBuilder.Entity<StudentSubject>().HasOne(ss => ss.Student).WithMany(s =>  
4         s.Subjects).HasForeignKey(ss => ss.StudentId);  
5     modelBuilder.Entity<StudentSubject>().HasOne(ss => ss.Subject).WithMany(s =>  
6         s.Students).HasForeignKey(ss => ss.SubjectId);  
7 }
```

Vazby

- Problematika vazeb mezi objekty je složitější
- Někdy nám to podchytí dobré pojmenování
- Jindy je potřeba ručně dodefinovat (atributy, FluentAPI)
- Více k vazbám: `https://docs.microsoft.com/en-us/ef/core/modeling/relationships`

Editace dat

- Stačí změnit property u entity a uložit
- Pozor abychom pracovali s DbEntitou a ne „obyčejným objektem“

```
1 try {
2     using (UniversityContext ctx = new UniversityContext()) {
3         // Vytvoríme nový predmet (automaticky se vloží)
4         Subject sub = new Subject() {
5             Credits = 6,
6             Name = "Platforma .NET"
7         };
8         // Najdeme DbEntitu studenta
9         Student s = ctx.Students.FirstOrDefault(p => p.Id == 1);
10        if (s != null) { // Tu upravíme
11            s.Name = "Dave Lister";
12            s.Subjects.Add(new StudentSubject() {Student = s, Subject =
13                sub});
14            ctx.SaveChanges();
15        }
16    }
17 }
```

Mazání dat

- Nejdříve najdeme entitu

```
1 Student s = ctx.Students.FirstOrDefault(p => p.Id == 1);
```

- Poté ji odstraníme

```
1 ctx.Students.Remove(s);  
2 ctx.SaveChanges();
```

- Můžeme také jen změnit její stav:

```
1 Student s = ctx.Students.FirstOrDefault(p => p.Id == 1);  
2 ctx.Entry(s).State = EntityState.Deleted;  
3 ctx.SaveChanges();
```

- Smaže jen tuto entitu, ty které obsahuje nemusí
- Jde nastavit smazání všech souvisejících, mohou být ale navázány jinam – více podrobností:

<https://docs.microsoft.com/en-us/ef/core/saving/cascade-delete>

EF – závěrem

- Entity Framework Core je velmi rozsáhlé téma
- Ukázány pouze základy, důležité je samostudium
- Dát pozor, zda je dokumentace pro EFCore nebo pro EF6
 - ▶ Neustále se vyvíjí (odlišnosti ve verzích)
 - ▶ Modulárnost (ne všechny návody uvádí `using`)
- Některé databázové connectory nemusí podporovat vše
- Dobrý pomocník pro programátora, ale má i svá úskalí

Úkol

- Navrhnout vhodnou strukturu databázových objektů pro uložení studentů (v rozsahu studentiPredmetu.xml z minulých hodin) a jejich známek
- Vytvořit DB strukturu pomocí CodeFirst
- Vložit do databáze data studentů(z XML) a náhodně jim (rozumně) vygenerovat známky
- Pomocí LINQ dotazů a EntityFrameworku provést všechny úkoly z minulého cvičení:
- Výběr unikátních jmen a příjmení (3. - 8.), vymazání, editace, výpis studentů se známkami.