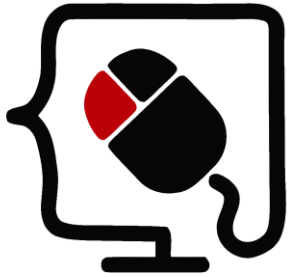


Национална програма
"Обучение за ИТ умения и кариера"
https://it-kariera.mon.bg/e-learning/

Министерството на
образованието и науката
https://www.mon.bg

Комуникация с база от данни

Entity Framework Core

Съдържание

- Преглед на Entity Framework Core
- Методът Database First
- Методът Code First
- CRUD Операции, използвайки Entity Framework Core
- Работа с LINQ

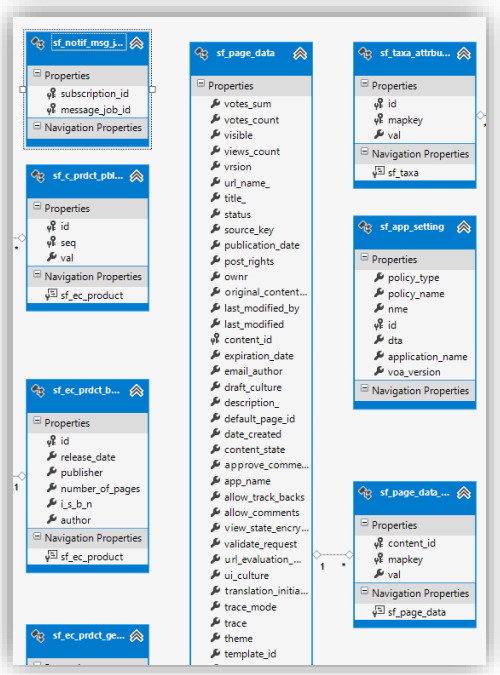
Entity Framework Core

Entity Framework Core

- Стандартната рамка за ORM за .NET и .NET Core
- Предоставя LINQ-базирани заявки за данни и CRUD операции
- Автоматично проследяване на промяната на обекти в паметта
- Работи с много релационни бази данни (с различни доставчици)
- Отворен код с независим цикъл на пускане на версии

Основен работен процес [1/2]

1. Определете модела на данни (Code First / Scaffold from DB)



2. Писане и изпълняване на заявки върху IQueryable

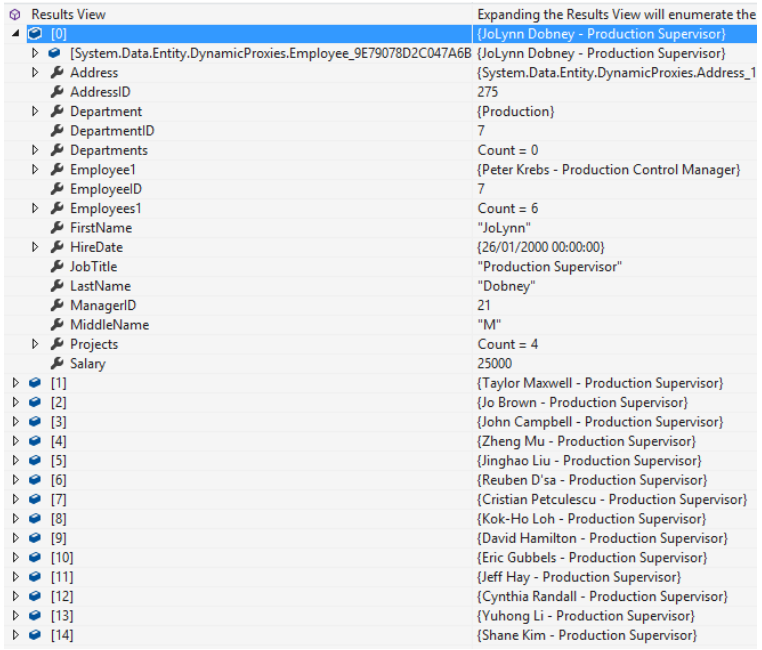
```
var toolName = "";

var snippetOptions = DefaultToolGroup
    .Tools
    .OfType<EditorListTool>()
    .Where(t =>
        t.Name == toolName &&
        t.Items != null &&
        t.Items.Any())
    .SelectMany(
        (t, index) =>
            t.Items
            .Select(item =>
                new {
                    text = item.Text,
                    value = item.Value
                }));

if (snippetOptions.Any())
{
    options[toolName] = snippetOptions;
}
```

3. EF генерира и изпълнява SQL заявка в БД

```
exec sp_executesql N'SELECT
[Filter2].[UserInCourseId] AS [UserInCourse
[Filter2].[UserId] AS [UserId],
[Filter2].[CourseInstanceId1] AS [CourseIns
[Filter2].[FirstCourseGroupId] AS [FirstCou
[Filter2].[SecondCourseGroupId] AS [SecondC
[Filter2].[ThirdCourseGroupId] AS [ThirdCou
[Filter2].[FourthCourseGroupId] AS [FourthC
[Filter2].[FifthCourseGroupId] AS [FifthCou
[Filter2].[IsLiveParticipant] AS [IsLivePar
[Filter2].[Accommodation] AS [Accommodatior
[Filter2].[ExcellentResults] AS [ExcellentR
[Filter2].[Result] AS [Result],
[Filter2].[CanDoTestExam] AS [CanDoTestExam
[Filter2].[CourseTestExamId] AS [CourseTest
[Filter2].[TestExamPoints] AS [TestExamPoir
[Filter2].[CanDoPracticalExam] AS [CanDoPra
[Filter2].[CoursePracticalExamId1] AS [Cour
[Filter2].[PracticalExamPoints] AS [Practic
[Filter2].[AttendancesCount] AS [Attendance
[Filter2].[HomeworkEvaluationPoints] AS [Ho
FROM   (SELECT [Extent1].[UserInCourseId] A
AS [SecondCourseGroupId], [Extent1].[ThirdC
[IsLiveParticipant], [Extent1].[Accommodati
[CourseTestExamId], [Extent1].[TestExamPoir
[PracticalExamPoints], [Extent1].[Attendanc
        FROM  [courses].[UsersInCourses] AS
        INNER JOIN [courses].[CoursePractic
        WHERE ( EXISTS (SELECT
                1 AS [C1]
                FROM [courses].[CoursePract
                WHERE [Extent1].[UserInCour
        )) AND ([Extent2].[AllowExamFilesEv
INNER JOIN [courses].[CoursePracticalExams]
WHERE ([Filter2].[UserId] = @p__linq__0) AN
```

Основен работен процес [1/2]

4. EF преобразува резултатите от заявката в .NET обекти
5. Промяна на данните със C# - извиква се "**Save Changes()**"
6. EF генерира и изпълнява SQL команда за промяна на БД



Results View	Expanding the Results View will enumerate the
[0]	{JoLynn Dobney - Production Supervisor}
[System.Data.Entity.DynamicProxies.Employee_9E79078D2C047A6B	{JoLynn Dobney - Production Supervisor}
Address	{System.Data.Entity.DynamicProxies.Address_1
AddressID	275
Department	{Production}
DepartmentID	7
Departments	Count = 0
Employee1	{Peter Krebs - Production Control Manager}
EmployeeID	7
Employees1	Count = 6
FirstName	"JoLynn"
HireDate	{26/01/2000 00:00:00}
JobTitle	"Production Supervisor"
LastName	"Dobney"
ManagerID	21
MiddleName	"M"
Projects	Count = 4
Salary	25000
[1]	{Taylor Maxwell - Production Supervisor}
[2]	{Jo Brown - Production Supervisor}
[3]	{John Campbell - Production Supervisor}
[4]	{Zheng Mu - Production Supervisor}
[5]	{Jinghao Liu - Production Supervisor}
[6]	{Reuben D'sa - Production Supervisor}
[7]	{Cristian Petculescu - Production Supervisor}
[8]	{Kok-Ho Loh - Production Supervisor}
[9]	{David Hamilton - Production Supervisor}
[10]	{Eric Gubbels - Production Supervisor}
[11]	{Jeff Hay - Production Supervisor}
[12]	{Cynthia Randall - Production Supervisor}
[13]	{Yuhong Li - Production Supervisor}
[14]	{Shane Kim - Production Supervisor}

```
private void ChangeBlogPostName(int id,
    string newName)
{
    var db = new Context();

    var post = db.Posts
        .FirstOrDefault(x => x.Id == id);

    if (post == null)
    {
        throw new ArgumentException(
            "Item with that id was not fo
            "id");
    }

    post.Name = newName;

    db.SaveChanges();
}
```

```
SELECT
[Extent1].[EmployeeID] AS [EmployeeID],
[Extent1].[FirstName] AS [FirstName],
[Extent1].[LastName] AS [LastName],
[Extent1].[MiddleName] AS [MiddleName],
[Extent1].[JobTitle] AS [JobTitle],
[Extent1].[DepartmentID] AS [DepartmentID],
[Extent1].[ManagerID] AS [ManagerID],
[Extent1].[HireDate] AS [HireDate],
[Extent1].[Salary] AS [Salary],
[Extent1].[AddressID] AS [AddressID]
FROM [dbo].[Employees] AS [Extent1]
WHERE N'Production Supervisor' = [Extent1].[JobTitle]
```

Entity Framework Core: Конфигурация

- За да добавите поддръжка на EFC към проект във Visual Studio:
  - Инсталирайте го от

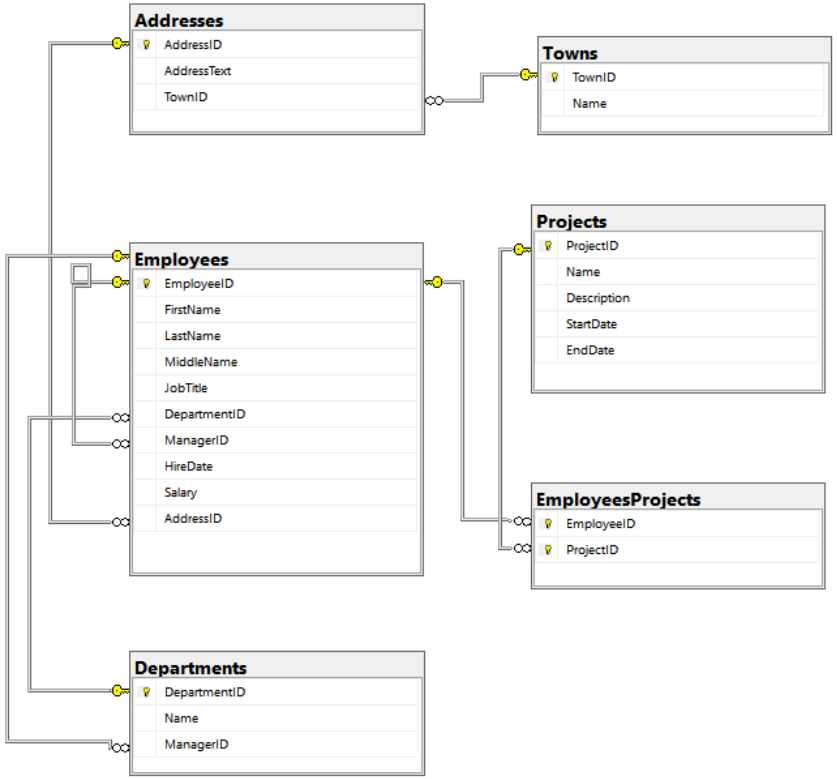
```
Install-Package Microsoft.EntityFrameworkCore
```

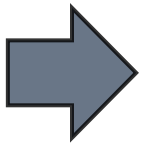
  - EFC е модулен - различни допълнителни пакети могат да бъдат инсталирани:

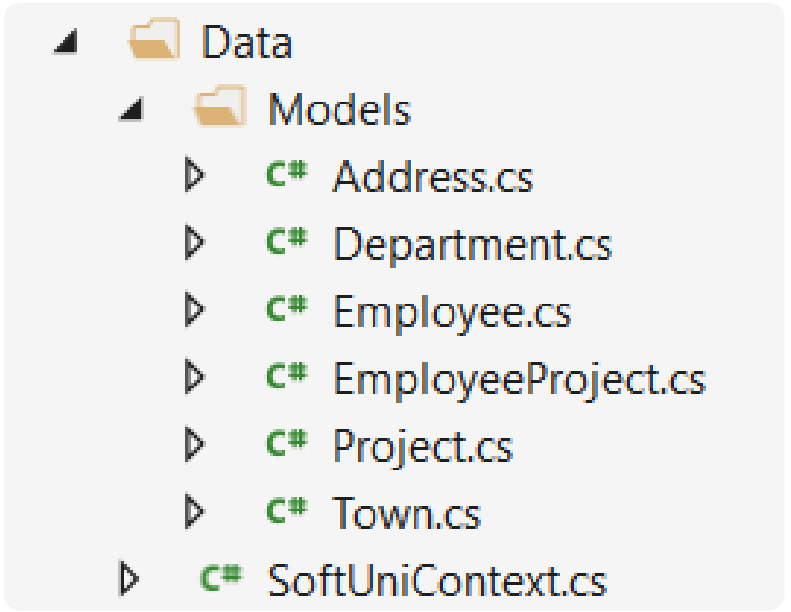
```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

Използване на метода Database First [1/2]

- Методът **Database First** моделира класовете с обекта след като базата данни е създадена:







Използване на метода Database First [2/2]

- Scaffolding DbContext от DB с командата Scaffold-DbContext в конзолата за управление на пакети:

```
Scaffold-DbContext
  -Connection "Server=.;Database=…;Integrated Security=True"
  -Provider Microsoft.EntityFrameworkCore.SqlServer
  -OutputDir Data
```

- Scaffolding предварително изисква следните пакети:

```
Install-Package Microsoft.EntityFrameworkCore.Tools
Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design
```

Използване на метода Code First [1/9]

- Създава съответните таблици в БД следвайки тяхно описание чрез класове
- Как да го използваме?
  - Създаваме класове описващи таблиците в нашата база, като чрез специален синтаксис оказваме връзките между тях
  - Създаваме клас, който задължително наследява DbContext – той ще е на шият БД контекст
    - В него добавяме като DbSet<T> класовете, описващи таблиците ни
    - Пренаписваме метода OnConfiguring(), за да окажем по какъв начин да се свържем с нашата БД

Използване на метода Code First [2/9]

- Накрая използвайки Package Manager конзолата добавяме поддръжка на миграции и с команда създаваме нашата база данни
- Нужни пакети:

```
Microsoft.EntityFrameworkCore.Tools
Microsoft.EntityFrameworkCore.Design
```

Използване на метода Code First [3/9]

- Да започнем със създаването на нашите ентитита
- EF е достатъчно умен, за да създаде първичен или вторичен ключ, забелязвайки имената на нашите свойства, завършващи на ID, Id

```
public class OrderDetail
{
    public int OrderDetailID { get; set; }
    public int OrderID { get; set; }
    public int ProductID { get; set; }
    public int Quantity { get; set; }
    public Order Order { get; set; }
}
```

OrderID, ще е референция към първичния ключ на поръчката, за която сме добавили връзка

EF разбира този синтаксис и ще създаде връзка между двете ни таблици

Използване на метода Code First [4/9]

- Съответно една поръчка ще има много на брой детайли за себе си
  - Типа на връзката е един-към-много

```
public class Order
{
    public int OrderID { get; set; }
    public int CustomerID { get; set; }
    public int EmployeeID { get; set; }
    public DateTime OrderDate { get; set; }
    public List<OrderDetail> OrderDetails { get; set; }
}
```

EF разбира, че една поръчка има много детайли и ще създаде връзка един-към-много

Използване на метода Code First [5/9]

- Накрая добавяме класовете като DbSet<T> в нашият контекст
- Описваме по какъв начин да стане връзката с БД

```
public class MyContext : DbContext
{
    public DbSet<OrderDetail> OrderDetails { get; set; }
    public DbSet<Order> Orders { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Data Source=(localdb)\ProjectsV13;Initial Catalog=StoreDB;");
    }
}
```

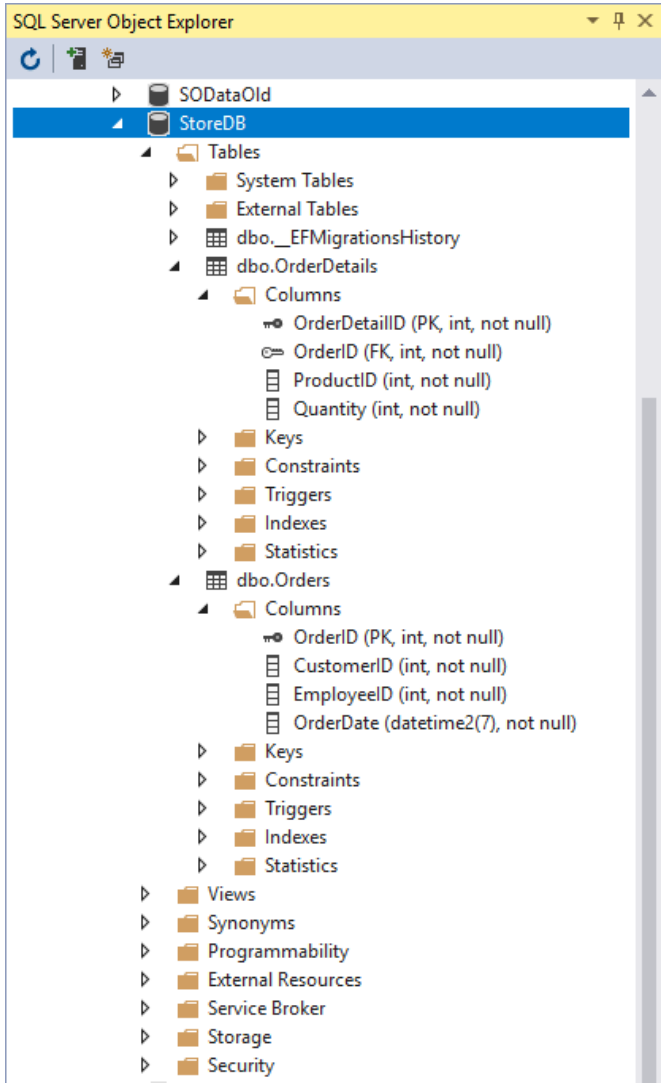
Използване на метода Code First [6/9]

- Добавяне на първата миграция
- Създаване на БД по описаните таблици
- Използвайки Package Manager Console:

```
$ Add-Migration Initial
$ Update-Database
```

- Вече съществува съответната БД, описана чрез нашите класове

Използване на метода Code First [7/9]



Използване на метода Code First [8/9]

- Fluent API – начин да се конфигурират по-сложни връзки в нашата БД, както и да се наложат ограничения за колоните
- Пренаписва се методът OnModelCreating(ModelBuilder modelBuilder) в нашият клас, служещ ни за контекст+
  - Чрез него може да се конфигурират връзки като 0-към-много, много-към-много и тн.

Използване на метода Code First [9/9]

Операциите за добавяне, изтриване, промяна и извличане се извършват по същия начин, както при Database First метода

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<BookCategory>()
        .HasKey(bc => new { bc.BookId, bc.CategoryId });
    modelBuilder.Entity<BookCategory>()
        .HasOne(bc => bc.Book)
        .WithMany(b => b.BookCategories)
        .HasForeignKey(bc => bc.BookId);
    modelBuilder.Entity<BookCategory>()
        .HasOne(bc => bc.Category)
        .WithMany(c => c.BookCategories)
        .HasForeignKey(bc => bc.CategoryId);
}
```

Описването на връзките става по лесен начин чрез методи със себеописващи се наименования на английски

EF Компоненти [1/2]

- Класът DbContext:
  - Съдържа връзката към базата данни и преобразуваните класове
  - Осигурява достъп до данни, базиран на LINQ
  - Осигурява проследяване на идентичността, проследяване на промените и API за CRUD операции
- Entity classes
  - Всяка таблица от база данни се свежда до C# клас

EF Компоненти [2/2]

- Асоциации (връзки между таблиците)
  - Асоциацията е базирана на първичен ключ / чужд ключ между два класа
  - Позволява навигация от един обект към друг

```
var courses = student.Courses.Where(…);
```

- Concurrency контрол
  - Entity Framework използва optimistic concurrency контрол
    - Няма заключване по подразбиране
    - Автоматично открива concurrency конфликти

Четене на Данни

Заявки към БД с помощта на EFC

Класът DbContext

- DbContext предоставя:
  - CRUD Операции
    - Начин за достъпване на записите
    - Метод за добавяне на нови записи (методът Add())
    - Възможност за манипулиране на данни от база данни чрез промяна на обекти
- Изпълнение на LINQ заявки като SQL заявки
- Управление на база данни създаване/изтриване/миграция

Използване на Класът DbContext

- Първо създайте инстанция на класа DbContext:

```
var context = new SoftUniDbContext();
```

- В конструктора може да бъде подаден низ за свързване към БД
- Свойствата на класа DbContext:
  - Database – EnsureCreated/Deleted методи, DB връзка
  - ChangeTracker – Съдържа информация за вградения тракера за промени
  - Всички таблици са изредени като свойства в следния формат :
    - DbSet<Employee> Employees { get; set; }

Четене на Данни с LINQ Заявки [1/2]

- Изпълненние на LINQ заявка:

```
using (var context = new SoftUniEntities())
{
  var employees = context.Employees
    .Where(e => e.JobTitle == "Design Engineer")
    .ToArray();
}
```

EF превежда това до SQL заявка

- Employees е свойство към класа DbContext:

```
public partial class SoftUniEntities : DbContext
{
  public DbSet<Employee> Employees { get; set; }
  public DbSet<Project> Projects { get; set; }
  public DbSet<Department> Departments { get; set; }
}
```

Четене на Данни с LINQ Заявки [2/2]

- Може да се използват и extension методи в заявката

```
using (var context = new SoftUniEntities())
  var employees = context.Employees
      .Where(c => c.JobTitle == "Design Engineering")
      .Select(c => c.FirstName)
      .ToList();
```

- Намиране на запис по ID

```
using (var context = new SoftUniEntities())
{
  var project = context.Projects.Find(2);
  Console.WriteLine(project.Name);
}
```

Прости операции с LINQ [1/2]

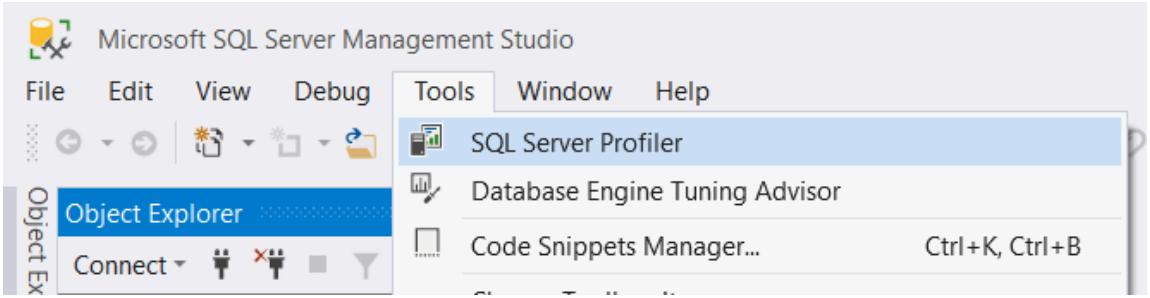
- Where()
  - Търси по дадено условие
- First/Last() / FirstOrDefault/LastOrDefault()
  - Получава първия / последния елемент, който съответства на условието
  - Хвърля InvalidOperationException грешка без OrDefault
- Select()
  - Преобразува колекция до друг тип
- OrderBy() / ThenBy() / OrderByDescending()
  - Сортира колекция по дадено условие

Прости операции с LINQ [2/2]

- Any()
  - Проверява дали някой елемент съответства на условие
- All()
  - Проверява дали всички елементи съответстват на условие
- Distinct()
  - Връща само уникалните елементи от колекция
- Skip() / Take()
  - Пропуска / взима X на брой елементи

Проследяване на SQL Заявките

- Заявки, изпратени до SQL Server, могат да бъдат наблюдавани с SQL Server Profiler
  - Включено е в SQL Server Management Studio:



- Заявките също могат да бъдат наблюдавани с Express Profiler

https://expressprofiler.codeplex.com/

CRUD операции с EFC

Създаване на Нови Данни

- За да създадете нов ред в БД, използвайте метода Add(...) на съответния DbSet:

```
var project = new Project()
{
  Name = "Judge System",
  StartDate = new DateTime(2015, 4, 15),
};

context.Projects.Add(project);
context.SaveChanges();
```

Създаване на нво Project обект

Добавяне на обекта към DbSet-а

Изпълнява SQL заявка

Каскадни Добавяния

- Можем да добавяме и каскадно:

```
Employee employee = new Employee();
employee.FirstName = "Petya";
employee.LastName = "Grozdarska";
employee.Projects.Add(new Project { Name = "SoftUni Conf"});
softUniEntities.Employees.Add(employee);
softUniEntities.SaveChanges();
```

- Проектът ще бъде добавен, когато служителя бъде добавен в базата данни

Промяна на съществуващи данни

- DbContext позволява промяна на свойствата на обект и запазване на промяната в базата данни
  - Просто заредете записа, променете го и извикайте SaveChanges()
- DbContext автоматично проследява всички промени

```
Employees employee = softUniEntities.Employees.First();
employees.FirstName = "Alex";
context.SaveChanges();
```

Изтриване на съществуващи данни

- Изтриването се извършва чрез Remove() върху зададена колекция

```
Employees employee =
  softUniEntities.Employees.First();
softUniEntities.Employees.Remove(employee);
softUniEntities.SaveChanges();
```

Маркира обекта за изтриване при следващото записване

Изпълнете командата за изтриване в SQL

Следене на промените

Следене на промените [1/5]

- Методът SaveChanges()
  - Минава през вътрешна колекция на DbContext класа
  - В зависимост от състоянието (Unchanged, Modified, Added, Deleted) генерира чрез type reflection заявка към БД
  - Как се пази състоянието?
    - В списък от сложни обекти, които по същество пазят две важни свойства – референция към следените обекти, в която се пази даден обект и състоянието му (обектите са от тип DbEntityEntry)
    - Запазената референция е към следения обект

Следене на промените [2/5]

- При инициализация на обект (нека за пример вземем User) в Stack паметта се запазва референция сочеща към паметта (Heap), в която този обект се съхранява
- При направени промени в тази памет съответния на инстанцията DbEntityEntry обект променя съхраняваното състояние

Следене на промените [3/5]

```
User u = context.Users.Find(1);
```

- Към момента нашият потребител u се пази във вътрешния списък на контекста със състояние Unchanged

```
u.Username = "New_Username"
```

- След като сме променили свойства на обекта u в паметта (Heap-a) се отразят нашите промени, те ще се отразят и в списъка на контекста
- Нашият потребител вече се пази със състояние Modified

Следене на промените [4/5]

```
context.Users.Remove(u);
```

- Ако премахнем потребителя от контекста, той не се изтрива от базата – състоянието му се променя на Deleted

```
User newUser = new User();
context.Users.Add(newUser);
```

- Подобно, ако добавим нов потребител в контекста, той не се добавя в базата от данни, а само започва да се следи (състоянието му е Added)

Следене на промените [5/5]

```
context.SaveChanges();
```

- Методът SaveChanges() обхожда вътрешния списък с DbEntityEntry елементи и за всеки следен обект изпълнява съответната заявка в зависимост от състоянието на записа

Национална програма
"Обучение за ИТ умения и кариера"
https://it-kariera.mon.bg

Министерството на
образованието и науката
https://www.mon.bg