* **Създаване и отваряне на връзка към SQL Server-a:**

SqlConnection dbCon = new SqlConnection(

“Server=.\\SQLEXPRESS; ” + “Database=Softuni; ” + “Integrated Security=true”);

dbCon.Open();

using (dbCon)

{

SqlCommand command = new SqlCommand(“SELECT \* FROM Employees”, dbCon);

SqlDataReader reader = command.ExecuteReader();

using (reader)

{

reader.Read();

Console.WriteLine(reader[index]);

reader.GetName(index); - връща името на колоната

reader.FieldCount(); - връща броя на колоните

}

}

- или само . на Server; това в скобите се нарича Connection string

* **Други методи:**

ExecuteScalar() - връща една стойност

ExecuteReader() - връща цяла таблица

ExecuteNonQuery() - при Insert и Delete

* **Executing LINQ-to-Entities:**

using (var context = new SoftUniEntities())

{

var employees =

from e in context.Employees

where e.JobTitle == “Design Engineer”

select e;

}

или

public partial class SoftUniEntities : DbContext

{

public IDbSet<Employee> Employees { get; set; }

public IDbSet<Project> Projects { get; set; }

public IDbSet<Department> Departments{ get; set; }

}

или

using (var context = new SoftUniEntities())

{

var employees = context.Employees

.Where(c => c.JobTitle == “Design Engineering”)

.Select(c => c.FirstName)

.ToList();

}

* **Създаване на нов ред в базата данни:**

var project = new Project()

{

Name = “Judge System”,

StartDate = new DateTime(2015, 4, 15),

};

context.Projects.Add(project);

context.SaveChanges();

* **Каскадно добавяне на нов ред:**

Employee employee = new Employee();

employee.FirstName = “Petya”;

employee.LastName = “Grozdarska”;

employee.Projects.Add(new Project { Name = “SoftUni Conf” });

softUniEntities.Employees.Add(employee);

softUniEntitites.SaveChanges();

* **Изтриване на ред от базата:**

Employees employee = softUniEntities.Employees.First();

softUniEntities.Employees.Remove(employee);

softUniEntities.SaveChanges();

* **Overload Constructor:**

public class IceCream

{

public IceCream(string flavour, int scoops) - Constructor

{

this.Flavour = flavour;

this.Scoops = scoops;

}

public IceCream() - Overloaded Constructor/ chain-ване на

: this(“Vanilla”, 3) {} конструктори

public string Flavour { get; set; } - Parameters

public int Scoops { get; set; }

public override string ToString() - като кажем ToString(), ще ни

{ връща something

return “something”;

}

}

* **Клас със полета:**

public class Person

{

private int age; - поле

public string Name { get; set; }

public int Age {

get { return this.age; }

set {

if (value < 0) { throw new InvalidArgumentException(); }

this.age = value;

}

}

}

* **Създаване на клас DbContext:**

public class ForumContext : DbContext

{

public ForumContext() : base(“ForumDb”) {} - ForumDb е името на connection string-a, има го и в app.config

- за всяка таблица в базата по едно:

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

public virtual DbSet<Post> Posts { get; set; }

public virtual DbSet<PostAnswer> PostAnswers { get; set; }

public virtual DbSet<Tag> Tags { get; set; }

}

* **Създаване на базата данни:**

В Main-a:

SoftUniContext context = new SofrUniContext();

Context.Database.Initialize(true); - този ред после го трием

* **Други видове методи:**

static void PrintName(string firstName, string lastName = “Dimitrova”)

{

Console.WriteLine(firstName + “ ” + lastName);

}

static void PrintName(params string[] names)

{} - може да приема множество

Параметри

* **Задаване базата да се дропне, ако по нея има промени:**

public MyDBContext() : base(“MyConnectionString”)

{

Database.SetInitializer(

new DropCreateDatabaseIfModelChanges<MyDBContext>()

);

}

// new DropCreateDatabaseAlways

// new CreateDatabaseIfNotExists

// new MigrateDatabaseToLatestVersion

* **Database Seeding:**

public class MyDBInitializer : DropCreateDatabaseAlways<MyDBContext>

{

protected override void Seed(MyDBContext context)

{

// TODO create, insert, delete, update or something else

base.Seed(context);

}

}

public MyDBContext() : base(“MyConnectionString”)

{

Database.SetInitializer(new MyDBInitializer());

}

* **Създаване на първоначална миграция:**

Tools => NuGet Package Manager => Package Manager Console

Enable-Migrations

* или за повече параметри:

get-help Enable-Migrations -detailed

* автоматични миграции:

Enable-Migrations -EnableAutomaticMigrations

* за да приложим промените върху базата:

Update-Database -Verbose (Verbose ще ни даде и SQL кода)

* **Конфигуриране на миграции:**

Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyDBContext, Configuration>());

- позволява промени, които премахват съществуващи колони:

public Configuration()

{

this.AutomaticMigrationsEnabled = true;

this.AutomaticMigrationDataLossAllowed = true;

}

* **Ръчни миграции:**

Add-Migration [Name]

Update-Database или

Update-Database -TargetMigration: <Name>

В създадената миграци има два метода:

* Up() - обновява базата към настоящия модел
* Down() - връща базата към предишната версия

В Up() може:

* Да се добави нов индекс:

CreateIndex(

TableName(“Car”),

ColumnName(“RegistrationNumber”),

Unique?(true),

IndexName(“IX\_Car\_BrandId”)

);

* Да се създаде нова колона с избрана от нас дифоутна стойност:

AddColumn(

“dbo.Blogs”,

“Rating”,

c => c.Int(

nullable: false,

defaultValue: 3)

);

* **Изпълняване на SQL заявка в миграция:**

Sql(@”UPDATE dbo.Posts

SET Abstract = LEFT(Content, 100)

WHERE Abstract IS NULL”);

* **Seed-ване по време на миграция:**

protected override void Seed(ForumContext context)

{

context.Tags.AddOrUpdate(t => t.Text, new Tag { Text = “C#” });

context.SaveChanges();

}

* **Създаване на SQL Script със миграцията в него:**

Update-Database -script - трябва да се напише преди да бъдат

изпълнени миграциите!!!!!!!

Update-Database -sourcemigration <Name> -script - ако искам да си

изберем конкретна миграция

* **One-to-Zero-Or-One връзка:**

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

public virtual Address Address { get; set; }

}

public class Address

{

[Key]

[ForeignKey(“Student”)]

public int Id { get; set; }

public string Text { get; set; }

public virtual Student Student { get; set; }

}

* **One-to-Many връзка:**

public class Department

{

public Department()

{

this.Employees = new HashSet<Employee>();

}

public int Id { get; set; }

…..

public virtual ICollection<Employee> Employees { get; set; }

}

public class Employee

{

public int Id { get; set; }

…..

public virtual Department Department { get; set; }

}

* **Many-to-Many връзка:**

public class Course

{

public Course()

{

this.Students = new HashSet<Student>();

}

…..

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

}

public class Student

{

public Student()

{

this.Courses = new HashSet<Course>();

}

……

public virtual ICollection<Course> Courses { get; set; }

}

* **Една таблица е свързана с друга, чрез два ключа:**

public class Person

{

public int Id { get; set; }

public string Name { get; set; }

public virtual Town PlaceOfBirth { get; set; }

public virtual Town CurrentResidence { get; set; }

}

public class Town

{

public Town()

{

this.Natives = new HashSet<Person>();

this.Residents = new HashSet<Person>();

}

public int Id { get; set; }

public string Name { get; set; }

[InverseProperty(“PlaceOfBirth”)]

public ICollection<Person> Natives { get; set; }

[InverseProperty(“CurrentResidence”)]

public ICollection<Person> Residents { get; set; }

}

* **Mapping Table с допълнителни полета:**

public class Bet

{

public Bet()

{

this.BetGames = new HashSet<BetGame>();

}

[Key]

public int Id { get; set; }

public decimal BetMoney { get; set; }

public virtual ICollection<BetGame> BetGames { get; set; }

}

public class Game

{

public Game()

{

this.BetGames = new HashSet<BetGame>();

}

[Key]

public int Id { get; set; }

public int Goals { get; set; }

public virtual ICollection<BetGame> BetGames { get; set; }

}

public class BetGame

{

[Key]

[Column(Order = 1)]

[ForeignKey(“Game”)]

public int GameId { get; set; }

public virtual Game Game { get; set; }

[Key]

[Column(Order = 2)]

[ForeignKey(“Bet”)]

public int BetId { get; set; }

public virtual Bet Bet { get; set; }

public string ResultPrediction { get; set; }

}

И в Context:

public virtual DbSet<BetGame> BetGames { get; set; }

* **Атрибути:**
* [Key] - set primary key

- за композител ключ:

[Key]

[Column(Order=1)]

public int StudentId { get; set; }

[Key]

[Column(Order=2)]

public int CourseId { get; set; }

* [ForeignKey] - свързва навигационно пропърти и foreign key пропърти в рамките на един клас (променя името на колона)
* [Table(“NewName”)] - за промяна името на таблица
* [Table(“NewName”, Schema=”Admin”)]
* [Column(“StudentName”, Order=2, TypeName=”varchar(50)”)] - за промяна на името на колона, реда й и типа данни
* [Required] - set to NOT NULL
* [MinLength] - това е само клиентска валидация
* [MaxLength]/[StringLength] - и клиентска валидация, и валидация в базата данни
* [Range] - за цифрови ст-ти, клиентска валидация
* [Index(“INDEX\_Name”, IsUnique=true)] -създава индекс за колона
* [NotMapped] - няма да е част от модела, само за наши цели
* DDL - Data Definition Language
* DML - Data Manipulation Language
* **Инициализиране на Fluent API (Model Builder):**

In Context:

protected override void

OnModelCreating(DbModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

}

* **Set primary key:**

modelBuilder.Entity<Student>().HasKey(s => s.StudentKey);

* **Преименуване на таблица (Admin е схемата, ако искаме да я сменим го слагаме):**

modelBuilder.Entity<Order>().ToTable(“OrderRef”, “Admin”);

* **Преименуване на колона:**

modelBuilder.Entity<Student>()

.Property(s => s.Name)

.HasColumnName(“StudentName”)

.HasColumnType(“varchar”);

* **Добавяне на атрибути към колона:**

modelBuilder.Entity<Person>().Property(p => p.FirstName)

.IsOptional()

.IsRequired()

.IsFixedLength()

.HasMaxLength()

and others…

* **Връзка One-to-Zero-or-One:**

modelBuilder.Entity<Address>()

.HasRequired(a => a.Student)

.WithOptional(s => s.Address);

* **Връзка One-to-One:**

modelBuilder.Entity<Address>()

.HasRequired(a => a.Student)

.WithRequiredDependent(s => s.Address);

* **Връзка One-to-Many:**

modelBuilder.Entity<Comment>()

.HasRequired(c => c.Post)

.WithMany(p => p.Comments)

Ако искаме да сменим=> .HasForeignKey(c => c.PostKey);

името на foreign key-a

* **Връзка Many-to-Many:**

modelBuilder.Entity<Student>()

.HasMany(s => s.Courses)

.WithMany(c => c.Students);

* **Връзка Many-to-Many Self Reference:**

modelBuilder.Entity<Book>()

.HasMany(b => b.RelatedBooks)

.WithMany()

.Map(m =>

{

m.MapLeftKey("BookId");

m.MapRightKey("RelatedId");

m.ToTable("RelatedBooks");

});

public class Book

{

public Book()

{

this.RelatedBooks = new HashSet<Book>();

}

[Key]

public int Id { get; set; }

[Required]

[StringLength(50, MinimumLength = 1)]

public string Title { get; set; }

public virtual ICollection<Book> RelatedBooks { get; set; }

}

* **Връзка Many-to-Many с променен Foreign Key:**

предното +

.Map(cs =>

{

cs.ToTable(“StudentCourses”);

cs.MapLeftKey(“StudentKey”);

cs.MapRightKey(“CourseRefId”);

});

* **Ако искаме да не включваме някоя колона:**

modelBuilder.Entity<Department>().Ignore(d => d.Budget);

* **Изключване на каскадното изтриване:**

modelBuilder.Entity<Course>()

.HasRequired(t => t.Department)

.WithMany(t => t.Courses)

.HasForeignKey(d => d.DepartmentID)

.WillCascadeOnDelete(false);

* **Специален клас, в който може да си отделим конфигурацията с Model Builder-a:**

public class StudentConfiguration : EntityTypeConfiguation<Student>

{

public StudentConfiguration()

{

This.HasKey(s => s.StudentKey);

}

}

modelBuilder.Configurations.Add(new StudentConfiguration());

* **Една таблица има две връзки към друга таблица:**

modelBuilder.Entity<Person>()

.HasOptional(p => p.CurrentResidence)

.WithMany(t => t.Residents)

.HasForeignKey(p => p.CurrentResidenceId); int?

modelBuilder.Entity<Person>()

.HasOptional(p => p.PlaceOfBirth)

.WithMany(t => t.Natives)

.HasForeignKey(p => p.PlaceOfBirthId); int?

* **SELECT с анонимен обект с Entity Framework:**

var employeesWithTown = context

.Employees

.Select(employee => new

{

EmployeeName = employee.Name,

TownName = employee.Town.Name

});

* **Join-ване на таблици с Entity Framework:**

var employees = softUniEntities.Employees.Join(

softUniEntities.Departments,

(e => e.DepartmentID),

(d => d.DepartmentID),

(e, d) => new

{

Employee = e.FirstName,

JobTitle = e.JobTitle,

Department = d.Name

});

* **Групиране на таблици с Entity Framework:**

var groupedEmployees =

from employee in softUniEntities.Emloyees

group employee by employee.JobTitle;

ИЛИ

var groupedCustomers = softUniEntities.Employees.

GroupBy(employee => employee.JobTitle);

//.Select(e => new

{

DepartmentID = e.Key, // .Key е важноооо!!!!

AverageSalary = e.Average(d => d.Salary)

}).ToList();

var data = context.Employees.Include(e => e.Department);

* **Пример за ViewModel:**

public class UserInfoView

{

public string Alias { get; set; }

public byte[] Avatar { get; set; }

}

//Example

var currentUser = context.Users

.Find(8)

.Select(u => new UserInfoView

{

Alias = u.FirstName + “ ” + u.LastName,

Avatar = u.Avatar

})

.SingleOrDefault();

//Example

public static UserInfoView GetUserInfo(int Id) { … }

* **Table per Hierarchy:**

modelBuilder.Entity<Vehicle>()

.Map<Car>(m => m

.Requires(“Discriminator”)

.HasValue(“Car”)

)

.Map<Truck>(m => m

.Requires(“Discriminator”)

.HasValue(“Truck”)

);

* **Table per Type:**

modelBuilder.Entity<ChildClassOne>()

.ToTable(“TableNameOfChildClassOne”);

modelBuilder.Entity<ChildClassTwo>()

.ToTable(“TableNameOfChildClassTwo”);

* **Table per Concrete Type:**

modelBuilder.Entity<Car>().Map(m =>

{

m.MapInheritedProperties();

m.ToTable(“Car”);

});

modelBuilder.Entity<ParentClass>()

.Property(p => p.”Id”)

.HasDatabaseGenerationOption(DatabaseGenerationOption.None);

* **Изпълняване на Native Sql Query в Entity Framework:**

In Context:

ctx.Database.SqlQuery<return-type>(native-SQL-query);

string query = “SELECT COUNT(\*) FROM dbo.Employees”;

var queryResult = ctx.Database.SqlQuery<int>(query);

int customersCount

* **Изпълняване на Native Sql Query с параметри:**

var context = new SoftUniContext();

string nativeSQLQuery =

“SELECT FirstName + ‘ ’ + LastName ” +

“FROM dbo.Employees WHERE JobTitle = {0}”;

var employees = context.Database.SqlQuery<string>(nativeSQLQuery, “Marketing Specialist”);

ИЛИ

string nativeSQLQuery =

“SELECT FirstName + ‘ ’ + LastName ” +

“FROM dbo.Employees WHERE JobTitle = @nameParam”;

var nameParam = new SqlParameter(“@nameParam”, “Peter”);

var employees = context.Database.SqlQuery<string>(nativeSQLQuery, nameParam);

* **Attached and Detached Objects:**

Когато са Detached можем да ги променяме при нас, но не се променят в базата.

var client = context.Clients.FirstOrDefault();

context.Entry(client).State; // changed or unchanged

context.Entry(client).State = System.Data.Entity.EntityState.Deleted;

context.SaveChanges();

* **Bulk DELETE: Install Entity Framework Extended**

context.Users.Delete(u => u.FirstName == “Pesho”);

* **Bulk UPDATE:**

context.Employees.Update(

t => t.Name == “Nasko”,

t => new Employee() { Name = “Plamen” });

* **Изпълняване на процедури от базата данни (stored procedures):**

CREATE PROCEDURE UpdateAge @age INT

AS

BEGIN

UPDATE Employees SET Age = Age + @age;

END

var nameParam = new SqlParameter(“@nameParam”, “Peter”);

var employees = context.Database.SqlQuery<string>(nativeSQLQuery, nameParam);

SqlParameter ageParameter = new SqlParameter(“@age”, SqlDbType.Int);

ageParameter.Value = 2;

context.Database.ExecuteSqlCommand(“UpdateAge @age”, ageParameter);

* **Eager loading:**

context.Towns.Include(town => town.Employees);

* **Изключване на Lazy Loading:**

In Context:

public CompanyContext() : base(“name=”CompanyContext””)

{

Configuration.LazyLoadingEnabled = false;

}

* **Explicit Loading:**

context.**Entry**(blog)

.Collection(b => b.Posts)

.Query()

.Where(p => p.Tags.Contains(“entity-framework”))

.**Load**();

var localPosts = context.Posts.Local;

localPosts.Add(new Post { Name = “What’s New in EF” });

localPosts.Remove(context.Posts.Find(1));

* **Concurrency (едновременни промени в/у един контекст от различни хора):**

ConcurrencyMode=Fixed - in DB First project

[ConcurrencyCheck] - in Code First project

* **Изключване на каскадното изтриване глобално и ръчно:**

modelBuilder.Conventions

.Remove<OneToManyCascadeDeleteConvention>();

modelBuilder.Entity<User>()

.HasMany(u => u.Answers)

.WithRequired(a => a.User)

.WillCascadeOnDelete(false);

* **Подобряване на производителността:**

try

{

context.Configuration.AutoDetectChangesEnabled = false;

var product = context.Products.Find(productId);

….

}

finally

{

context.Configuration.AutoDetectChangesEnabled = true;

}

* **Ако да видим информация, без да я променяме, може да изключим tracking-a:**

context.Products

.AsNoTracking()

.Where(p => p.Price < 150)

.ToList();

* **Singleton Pattern:**

public class Authenticator

{

private static Authenticator instance;

private Authenticator() { … }

public static Authenticator Instance

{

get

{

if (instance == null) instance = new Authenticator();

return instance;

}

}

}

* **Data Transfer Object - DTO:**

public class ProductDTO

{

public string Name { get; set; }

public int StockQty { get; set; }

}

* **Manual Mapping:**

var product = context.Products.FirstOrDefault();

var productDto = new ProductDTO

{

Name = product.Name,

StockQty = product.ProductStocks.Sum(ps => ps.Quantity)

};

* **Auto Mapping:**

Install-Package AutoMapper

* **Auto Mapping Initialization:**

Mapper.Initialize(cfg => cfg.CreateMap<Product, ProductDTO>());

var product = context.Products.FirstOrDefault();

ProductDTO dto = Mapper.Map<ProductDTO>(product);

* **Multiple Mapping:**

Mapper.Initialize(cfg =>

{

cfg.CreateMap<Product, ProductDTO>();

cfg.CreateMap<Order, OrderDTO>();

cfg.CreateMap<Client, ClientDTO>();

cfg.CreateMap<SupportTicket, TicketDTO>();

});

* **Customized Mapping:**

Mapper.Initialize(cfg =>

cfg.CreateMap<Product, ProductDTO>()

.ForMember(dto => dto.StockQty,

opt => opt.MapFrom(src =>

src.ProductStocks.Sum(p => p.Quantity))));

* **Flatten Complex Properties:**

Mapper.Initialize(cfg =>

cfg.CreateMap<Event, CalendarEventFrom>()

.ForMember(

dto => dest.Date, opt => opt.MapFrom(src => src.Date.Date))

.ForMember(

dto => dest.Hour, opt => opt.MapFrom(src => src.Date.Hour))

.ForMember(

dto => dest.Minute, opt => opt.MapFrom(src => src.Date.Minute)));

* **Collection Mapping:**

List<ProductDTO> productDTOs =

Mapper.Map<Product[], List<ProductDTO>>(products); ИЛИ

List<ProjectDTO> projectsDtos = context.Products.

.Where(p => p.Id == 18)

.ProjectTo<ProductDTO>() !!!!!!!!!

.ToList();

* **Flattening Complex Objects:**

public class OrderDTO

{

public string ClientName { get; set; }

public decimal Total { get; set; }

}

Mapper.Initialize(cfg => cfg.CreateMap<Order, OrderDTO>());

OrderDTO dto = Mapper.Map<Order, OderDTO>(order);

* **Inheritance Mapping:**

Mapper.Initialize(cfg => {

cfg.CreateMap<Order, OrderDTO>()

.Include<OnlineOrder, OnlineOrderDTO>()

.Include<MailOrder, MailOrderDTO>();

cfg.CreateMap<OnlineOrder, OnlineOrderDTO>();

cfg.CreateMap<MailOrder, MailOrderDTO>();

});

* **JSON Array:**

[5, “text”, true] - can hold different data types

* **JSON Object:**

String jsonObject =

“{

‘firstName’: ‘Vladimir’, ‘lastName’: ‘Georgiev’,

‘jobTitle’: ‘Technical Trainer’, ‘age’: 25

}”

* **Превръщане от JSON обекти в C# обекти и обратно, чрез JavaScript Serializer:**

Add Reference to System.Web.Extentions assembly

var product = new Product()

{

Cost = 25,

Description = null,

Name = “ =Oil pump”

};

var serializer = new JavaScriptSerializer();

var jsonProduct = serializer.Serialize(product);

var objProduct = serializer.Deserialize<Product>(jsonProduct);

* **Parse Dictionary to JSON Object:**

var products = new Dictionary<string, Product>

{

{ “pump”, firstProduct },

{ “filter”, secondProduct }

};

var jsonProducts = serializer.Serialize(products);

// json object will look like that:

{

‘pump’: {‘Id’: 0, ‘Name’: ‘Oil Pump’, ‘Description’: null, ‘Cost’: 25},

‘filter’: {‘Id’:0, ‘Name’: ‘Oil Filter’, ‘Description’: null, ‘Cost’: 15}

}

* **Installing JSON.NET:**

From NuGet Package Manager or

in Package Manager Console: Install-Package Newtonsoft.Json

* **Parsing to JSON objects and visa versa:**

var jsonProduct = JsonConvert.SerializeObject(product);

var objProduct = JsonConvert.DeserializeObject<Product>(jsonProduct);

* **Saving Json to file:**

File.WriteAllText(“team.json”, teamToExportInJson);

* **Indent/ Formatting the JSON output:**

JsonConvert.SerializeObject(product, Formatting.Indented);

* **Deserialize to anonymous types:**

var json = @”{ ‘firstName’: ‘Vladimir’,

‘lastName’: ‘Georgiev’,

‘jobTitle’: ‘Technical Trainer’ }”;

var template = new

{

FirstName = string.Empty,

LastName = string.Empty,

JobTitle = string.Empty

};

var person = JsonConvert.DeserializeAnonymousType(json, template);

* **Parsing an object to JSON with configuring:**

public class User

{

[JsonProperty(“user”)]

public string Username { get; set; }

[JsonIgnore]

public string Password { get; set; }

}

* **JObject:**

JObject obj = JObject.Parse(jsonProduct);

Може да се вземе и от файл:

using (StreamReader reader = [File.OpenText(@”c:\categories.json”))](mailto:File.OpenText(@\”c:/categories.json\”)))

{

JObject obj = (JObject)JToken

.ReadFrom(new JsonTextReader(reader));

// ….

}

Console.WriteLine(obj[“Name”]); // Oil Pump

* **JObjects, queried with LINQ:**

var json = @”{’products’: [

{‘name’: ‘Fruits’, ‘products’: [‘apple’, ‘banana’, ‘orange’]},

{‘name’: ‘Vegetables’, ‘products’: [‘cucumber’, ‘patato’, ‘eggplant’]}]}”;

JObject productsAsJObj = JObject.Parse(json);

int index = 1;

List<string> products = productsAsJObj[“products”].Select(c =>

string.Format(“{0}. {1} ({2})”,

index++,

c[“name”],

string.Join(“, ”, c[“products”])

));

// 1. Fruits (apple, banana, orange)

// 2. Vegetables (cucumber, patato, eggplant)

* **Важно!!!!!!**

var products = context.Products

.Where(p => p.Price >= 500 && p.Price <= 1000)

.OrderBy(p => p.Price)

.ToList()

.Select(p => new

{

Name = p.Name,

Price = p.Price,

Seller = p.Seller.FirstName + " " + p.Seller.LastName

})

.ToList();

За да може да свържем два стринга, трябва първо да ги ToList()-нем!!!!!!!!!!!!

* **XML Example:**

<?xml version=”1.0” encoding=”UTF-8”?>

<library name=”Developer’s Library”>

<book>

<title>Professional C# 4.0 and .NET 4</title>

<author>Christian Nagel</author>

<isbn>978-0-470-50225-9</isbn>

</book>

<book>

<title>Teach Yourself XML in 10 Minutes</title>

<author>Andrew H. Watt</author>

<isbn>978-0-672-32471-0</isbn>

</book>

</library>

* **Reading XML:**

string str =

@”<?xml version=””1.0””?>

<!-- comment at the root level -->

<Root>

<Child>Content</Child>

</Root>”;

XDocument doc = XDocument.Parse(str);

* **Loading XML from file:**

XDocument xmlDoc = XDocument.Load(“../../books.xml”);

* **Working with XDocument:**

var cars = xmlDoc.Root.Elements();

foreach (var car in cars)

{

string make = car.Element(“make”).Value;

string model = car.Element(“model”).Value;

Console.WriteLine($”{make} {model}”);

}

* **Променяне или добавяне на стойност на елемент в XML:**

customer.SetElementValue(“birth-date”, “1990-10-04T00:00:00”);

customer.Emelent(“birth-date”).Value = “1990-10-04T00:00:00”;

* **Премахване на елемент:**

var youngDriver = customer.Element(“is-young-driver”);

youngDriver.Remove();

* **Взимане на атрибут на елемент:**

customer.Attribute(“name”).Value;

* **Взимане на списък с всички атрибути на елемент:**

var attrs = customer.Attributes();

* **Променяне стойността на атрибут:**

customer.SetAttributeValue(“age”, “21”);

* **Търсене в XML с LINQ:**

XDocument xmlDoc = XDocument.Load(“cars.xml”);

var cars = xmlDoc.Root.Elements()

.Where(e => e.Element(“make”).Value == “Opel” &&

(long)e.Element(“travelled-distance”).Value >= 300000)

.Select(c => new

{

Model = c.Element(“model”).Value,

Travelled = c.Element(“travelled-distance”).Value

})

.ToList();

foreach (var car in cars)

{

Console.WriteLine(car.Model + “ ” + car.Travelled);

}

* **Създаване на XML:**

<books>

<book>

<author>Don Box</author>

<title lang=”en”>Essential .NET</title>

</book>

</books>

XDocument xmlDoc = new XDocument();

xmlDoc.Add(

new XElement(“books”,

new XElement(“book”,

new XElement(“author”, “Don Box”),

new XElement(“title”, “ASP.NET”, new XAttribute(“lang”, “en”))

)));

* **Запазване на XML във файл:**

xmlDoc.Save(“myBooks.xml”);

* **Запазване на XML във файл, като изключваме автоматичното форматиране:**

xmlDoc.Save(“myBooks.xml”, SaveOptions.DisableFormatting);

* **Serialize object to xml file:**

// може и product.GetType()

var serializer = new XmlSerializer(typeof(ProductDTO));

var writer = new StreamWriter(“myProduct.xml”);

using (writer)

{

serializer.Serialize(writer, product);

}

* **Превръщане на xml файл в някакъв обект:**

var serializer = new XmlSerializer(typeof(ProductDTO));

var reader = new StreamReader(“../../product.xml”);

var product = (ProductDTO)serializer.Deserialize(reader);