Упражнения: Да направим CRUD приложение без ORM

Създаване на просто приложение

В рамките на това упражнение, ще създадем стъпка по стъпка малко CRUD конзолно приложнеие за управление на продукти в нашата база данни. Ще използваме SQL Server за БД.

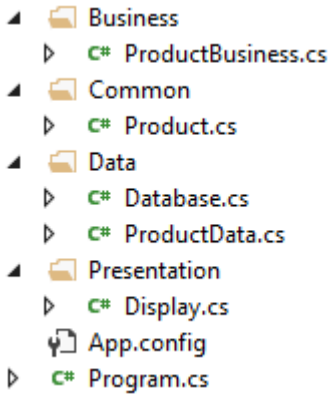
1. Създаване на БД

Отворете SQL Server Management Studio и създайте таблица, която се нарича **shop**, а в нея създайте таблица:

```
CREATE TABLE product
(
        Id INT IDENTITY PRIMARY KEY,
        Name VARCHAR(100) NOT NULL,
        Price DECIMAL(10,2),
        Stock INT
);
```

Създаване нА структура на проекта

Нашият проект ще е организиран в трислойна архитектура, за тази цел създайте следните папки в проекта ви (може да използвате и проекти в solution-а):



2. Слой за данни

В нашият стой за данни ще създадем два класа – **статичен** клас **Database.cs** и клас **ProductData.cs**

Database.cs ще поддържа връзката с БД – там ще се подава и **низа за връзка** към нея.

```
public static class Database
{
    private static string connectionString = "Server=.; Database=shop; Integrated Security=true";
    public static SqlConnection GetConnection()
    {
        return new SqlConnection(connectionString);
    }
}
```

Освен това в този слой ще добавим и **ProductData.cs** в него ще създаваме необходимите SQL команди, за да извършваме **CRUD** операции.

Ще започнем реализирането с метода **GetAll()**, който трябва да върне списък от всички продукти в таблицата ни.

За целта ще ползваме SQL заявката за извличане на всичката информация от таблицата, а след това ще обработваме нейния резултат.

Това се случва чрез **ExecuteReader()** метод на **SqlCommand** класа и създаване на обект от клас **Product** на базата на информацията от четеца. Това се случва в **while** конструкция, която работи чрез метода **Read()** на четеца, който сме получили от **ExecuteReader().** Този метод на всяка итерация връща по един ред от таблицата. Когато накрая редовете в таблицата свършат, условието, което задвижва while конструкцията се превръща във **false** и така тя спира, а ние вече разполагаме с пълния списък на базата на тази таблица. Кодът е както следва:

```
public List<Product> GetAll()
{
    var productList = new List<Product>();
    using (var connection = Database.GetConnection())
    {
        var command = new SqlCommand("SELECT * FROM product", connection);
        connection.Open();
        using (var reader = command.ExecuteReader())
        {
                while (reader.Read())
                {
                    var product = new Product(
                        reader.GetInt32(0),
                        reader.GetString(1),
                        reader.GetDecimal(2),
                        reader.GetInt32(3)
                    );

                    productList.Add(product);
                }

        }
        connection.Close();
    }

    return productList;
}
```

Следващият метод, с който ще продължим е **Get()** метода. В този метод искаме по подадено **id** на продукт да получим информацията за него. Тук ще използваме параметризирана заявка, където параметъра е нашият **id**.

След това чрез извикване отново на **ExecuteReader()** и **Read()** метода получаваме информация. Ако в нашата база данни няма продукт с желаното **id**, условието на **if-конструкцията** ще бъде **false** и по този начин **product** ще остане със стойност **null**, зададена му по начало. Кодът е както следва:

```
public Product Get(int id)
{
    Product product = null;
    using (var connection = Database.GetConnection())
    {
        var command = new SqlCommand("SELECT * FROM product WHERE Id=@id", connection);
        command.Parameters.AddWithValue("id", id);
        connection.Open();
        using (var reader = command.ExecuteReader())
        {
            if(reader.Read())
            {
                product = new Product(
                        reader.GetInt32(0),
                        reader.GetString(1),
                        reader.GetDecimal(2),
                        reader.GetInt32(3)
                );
            }
        }

        connection.Close();
    }

    return product;
}
```

Сега нека да пристъпим към реализацията на **Add** метода. В него ще използваме параметризирана INSERT заявка. Реализацията на метода е доста праволинейна:

```
public void Add(Product product)
{
    using (var connection = Database.GetConnection()){
        var command = new SqlCommand("INSERT INTO product (Name, Price, Stock) VALUES(@name, @price, @stock)", connection);
        command.Parameters.AddWithValue("name", product.Name);
        command.Parameters.AddWithValue("price", product.Price);
        command.Parameters.AddWithValue("stock", product.Stock);
        connection.Open();
        command.ExecuteNonQuery();
        connection.Close();
    }
}
```

По сходен начин се реализира и Update метода. Разликата е, че в него ще използваме... UPDATE по Id ☺

```
public void Update(Product product)
{
    using (var connection = Database.GetConnection())
    {
        var command = new SqlCommand("UPDATE product SET Name=@name, Price=@price, Stock=@stock WHERE Id=@id", connection);
        command.Parameters.AddWithValue("id", product.Id);
        command.Parameters.AddWithValue("name", product.Name);
        command.Parameters.AddWithValue("price", product.Price);
        command.Parameters.AddWithValue("stock", product.Stock);
        connection.Open();
        command.ExecuteNonQuery();
        connection.Close();
    }

}
```

Остана да реализираме метода за изтриване по id. Там ще използваме отново параметризирана заявка:

```
public void Delete(int id)
{
    using (var connection = Database.GetConnection())
    {
        var command = new SqlCommand("DELETE product WHERE Id=@id", connection);
        command.Parameters.AddWithValue("id", id);
        connection.Open();
        command.ExecuteNonQuery();
        connection.Close();

    }
}
```

Навярно сте забелязали, че боравим с обекти от клас **Product**. Сега е време да го създадем. Тъй като този клас ще се явява общото между отделните елементи в нашата архитектура, той ще бъде в папката Common. В този клас няма нищо особено – просто свойства за 4-те колони от нашата таблица и конструктор. Вие вече сте реализирали такива класове, така че в този няма нищо ново за вас.

```
class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public int Stock { get; set; }

    public Product()
    {

    }

    public Product(int id, string name, decimal price, int stock)
    {
        this.Id = id;
        this.Name = name;
        this.Price = price;
        this.Stock = stock;
    }

}
```

3. Бизнес слой

Сега е време да реализираме класът с бизнес логиката на нашето приложение. Този клас сам по себе си е доста лесен за реализиране. Всичко, от което имаме нужда е поле от клас **ProductData** и методи, които викат тези на **ProductData** – **GetAll()**, **Get()**, **Add()**, **Update()**, **Delete()**:

```
class ProductBusiness
{
    private ProductData manager = new ProductData();

    public List<Product> GetAll()
    {
        return manager.GetAll();
    }

    public Product Get(int id)
    {
        return manager.Get(id);
    }

    public void Add(Product product)
    {
        manager.Add(product);
    }

    public void Update(Product product)
    {
        manager.Update(product);
    }

    public void Delete(int id)
    {
        manager.Delete(id);
    }
}
```

С това направихме проста реализация на нашата бизнес-логика, която предстои да използваме в Display.cs класа.

4. Презентационен слой

Тук ще имаме Display.cs клас. Той ще реализира конзолно меню, от което ще въвеждаме желана опция и съответно по този начин ще бъде контролиран вход/изхода на програмата.

Ще започнем със създаването на частен ShowMenu() метод, който ще бъде викан, за да показва какви са възможностите:

```
private void ShowMenu()
{
    Console.WriteLine(new string('-', 40));
    Console.WriteLine(new string(' ', 18)+"MENU"+new string(' ', 18));
    Console.WriteLine(new string('-', 40));
    Console.WriteLine("1. List all entries");
    Console.WriteLine("2. Add new entry");
    Console.WriteLine("3. Update entry");
    Console.WriteLine("4. Fetch entry by ID");
    Console.WriteLine("5. Delete entry by ID");
    Console.WriteLine("6. Exit");
}
```

Този метод ще се вика от друг частен метод – **Input().** Целта на този метод е да получим вход от потребителя – номер на желаната операция и според това да извикаме някой от другите методи. Методът може да бъде реализиран, чрез позната за вас **do/while** конструкция и **switch/case** конструкция. Една примерна реализация на метода би изглеждала ето така:

```
private void Input()
{
    var operation = -1;
    do
    {
        ShowMenu();
        operation = int.Parse(Console.ReadLine());
        switch (operation)
        {
            case 1:
                ListAll();
                break;
            case 2:
                Add();
                break;
            case 3:
                Update();
                break;
            case 4:
                Fetch();
                break;
            case 5:
                Delete();
                break;
            default:
                break;
        }
    } while (operation != closeOperationId);
}
```

В кода по-горе **closeOperationId** е поле на класа Display, в което задаваме номерът на операцията за затваряне на приложението, при въвеждането на който трябва да спрем да приемаме вход от потребителя:

```
private int closeOperationId = 6;
```

Самото извикване на **Input()** метода ще се случва в конструктора на класа Display:

```
public Display()
{
    Input();
}
```

Сега трябва да реализираме останалите методи. Ще започнем с **Add().** Водещата задача през този метод е той да въвежда информация за нов продукт. Съответно в него ние в отделни променливи ще въвеждаме името, цената и наличността на продукта, след което ще създаваме обект от клас **Product** и чрез обектът **productBusiness** ще извикваме **Add()** метода на бизнес логиката.

Кодът изглежда както следва:

```
private void Add()
{
    Product product = new Product();
    Console.WriteLine("Enter name: ");
    product.Name = Console.ReadLine();
    Console.WriteLine("Enter price: ");
    product.Price = decimal.Parse(Console.ReadLine());
    Console.WriteLine("Enter stock: ");
    product.Stock = int.Parse(Console.ReadLine());
    productBusiness.Add(product);
}
```

Следващият метод, който ще бъде реализиран е **ListAll().** В него трябва да си създадем променлива **products,** която ще получи своята стойност благодарение на метода **GetAll()** от бизнес логиката. След това трябва просто да обходим всички получени елементи и да ги изведем. Тук може да проявите въображение по начина на извеждане на елементите и да изведете информацията и по-красиво, тъй като примерната реализация не е направена особено красиво ☺

```
private void ListAll()
{
    Console.WriteLine(new string('-', 40));
    Console.WriteLine(new string(' ', 16)+"PRODUCTS"+new string(' ', 16));
    Console.WriteLine(new string('-', 40));
    var products = productBusiness.GetAll();
    foreach (var item in products)
    {
        Console.WriteLine("{0} {1} {2} {3}", item.Id, item.Name, item.Price, item.Stock);
    }
}
```

Сега, след като можем да виждаме вече всички елементи в таблицата, нека да преминем към възможността да ги редактираме. Това ще се случва в методът **Update(),** той ще иска от потребителя **id** на продукта за редактиране. Ако такъв продукт действително съществува, потребителят ще въвежда новите данни за него. Забележете, че този метод може да се направи значително по-удобен, отколкото в примера – например да показва досегашната стойност на свойствата в обекта... и не само ☺

```
private void Update()
{
    Console.WriteLine("Enter ID to update: ");
    int id = int.Parse(Console.ReadLine());
    Product product = productBusiness.Get(id);
    if (product != null)
    {
        Console.WriteLine("Enter name: ");
        product.Name = Console.ReadLine();
        Console.WriteLine("Enter price: ");
        product.Price = decimal.Parse(Console.ReadLine());
        Console.WriteLine("Enter stock: ");
        product.Stock = int.Parse(Console.ReadLine());
        productBusiness.Update(product);
    }
    else
    {
        Console.WriteLine("Product not found!");
    }
}
```

След като направихме методът за редактиране, е време да направим и метод за визуализиране на информацията по **id** на даден продукт. Тук ще използваме **Get()** метода от бизнес логиката.

```
private void Fetch()
{
    Console.WriteLine("Enter ID to fetch: ");
    int id = int.Parse(Console.ReadLine());
    Product product = productBusiness.Get(id);
    if (product != null)
    {
        Console.WriteLine(new string('-', 40));
        Console.WriteLine("ID: " + product.Id);
        Console.WriteLine("Name: " + product.Name);
        Console.WriteLine("Price: " + product.Price);
        Console.WriteLine("Stock: " + product.Stock);
        Console.WriteLine(new string('-', 40));
    }
}
```

Накрая ще реализираме и метод за изтриване на продукт по неговото **id.** Тук отново ще използваме съответния метод от бизнес логиката:

```
private void Delete()
{
    Console.WriteLine("Enter ID to delete: ");
    int id = int.Parse(Console.ReadLine());
    productBusiness.Delete(id);
    Console.WriteLine("Done.");
}
```

За да завършим нашето приложение имаме нужда единствено от създаването на обект от клас **Display** в рамките на **Main** метода в **Program.cs.**

Така приключва нашата реализация на приложение без ORM. Плюсът тук е, че имаме по-голям контрол върху заявките и кодът прави **точно това, което сме му казали**. За сметка на това, ако желаем да добавим втора таблица **clients,** които да купуват продуктите, а след това и **orders,** в която да се записва поръчката на всеки клеинт, ще трябва да повторим част от тривиалните заявки, които написахме тук, а освен това ще трябва да пишем и доста допълнително код.

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист".

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).