03

3 - Linq métodos extensão

Transcrição

A proposta da presente aula é extrair o código da mediana transformando-o em uma função, servindo também para outras finalidades. Por isso, selecionaremos o seguinte trecho:

```
var contagem = query.Count();

var queryOrdenada = query.OrderBy( total => total);

var elementoCentral = queryOrdenada.Skip(contagem / 2).First();

var mediana = elementoCentral;
```

Com o trecho selecionado utilizaremos o atalho "Ctrl + . ." e assim abre uma janela com a opção "Renomear método" que passaremos a chamar de `Mediana` . Rodando o código temos o mesmo valor, ou seja, extraímos a função sem quebrar o código!

É interessante reparar que quando calculamos a **média** das "Notas Fiscais" utilizaremos uma sintaxe diferente, por exemplo, fizemos uso do `contexto.NotasFiscais.Average` . Depois, passaremos o valor `nf => nf.Total` e vamos criamos uma `query` . Isto é, a maneira como calculamos a **média** foi diferente da que usamos para encontrar a **mediana**.

Nosso próximo passo é transformar o cálculo da **mediana** seguindo o modelo da **média**. Para fazer isso vamos utilizar os métodos de extensão. O método complementará a biblioteca de código já existente.

Portanto, vamos criar um método de extensão e para isso reaproveitaremos a função `mediana` . No cálculo da `mediana` , acabamos esquecendo de um detalhe: a existência de uma quantidade de elementos pares na `query` . Assim, antes mesmo de começar a criar o método de extensão vamos acertar a função mediana. Declaramos que o `elementoCentral` já existente será o de número 1 (`elementoCentral_1`). Para o caso de um número par de itens, teremos o `elementoCentral_2` para o qual apenas copiamos o que já existe:

```
var elementoCentral_1 = queryOrdenada.Skip(contagem / 2).First();

var elementoCentral_2 = queryOrdenada.Skip(contagem / 2).First();
```

Para o caso de elementos em número ímpar a lógica é pegar o total, dividir por dois, arredondar para baixo o resultado e pular essa quantidade de casas e pegar o próximo item. Por exemplo:

```
5/2 = 2,5

O resultado arredondado: 2
O item que é a mediana: terceiro
```

No código a expressão para quantidade de objetos ímpar já foi construída:

```
var elementoCentral_1 = queryOrdenada.Skip(contagem / 2).First();
```

Para o caso de elementos de número **par** a lógica é dividir a quantidade de objetos, pegar o elemento que equivale ao resultado e também o anterior a ele. A expressão que representa essa lógica é:

```
6-1/2 = 5/2 = 2,5
```

Para calcular os objetos centrais em elementos em quantidade par faremos o seguinte cálculo:

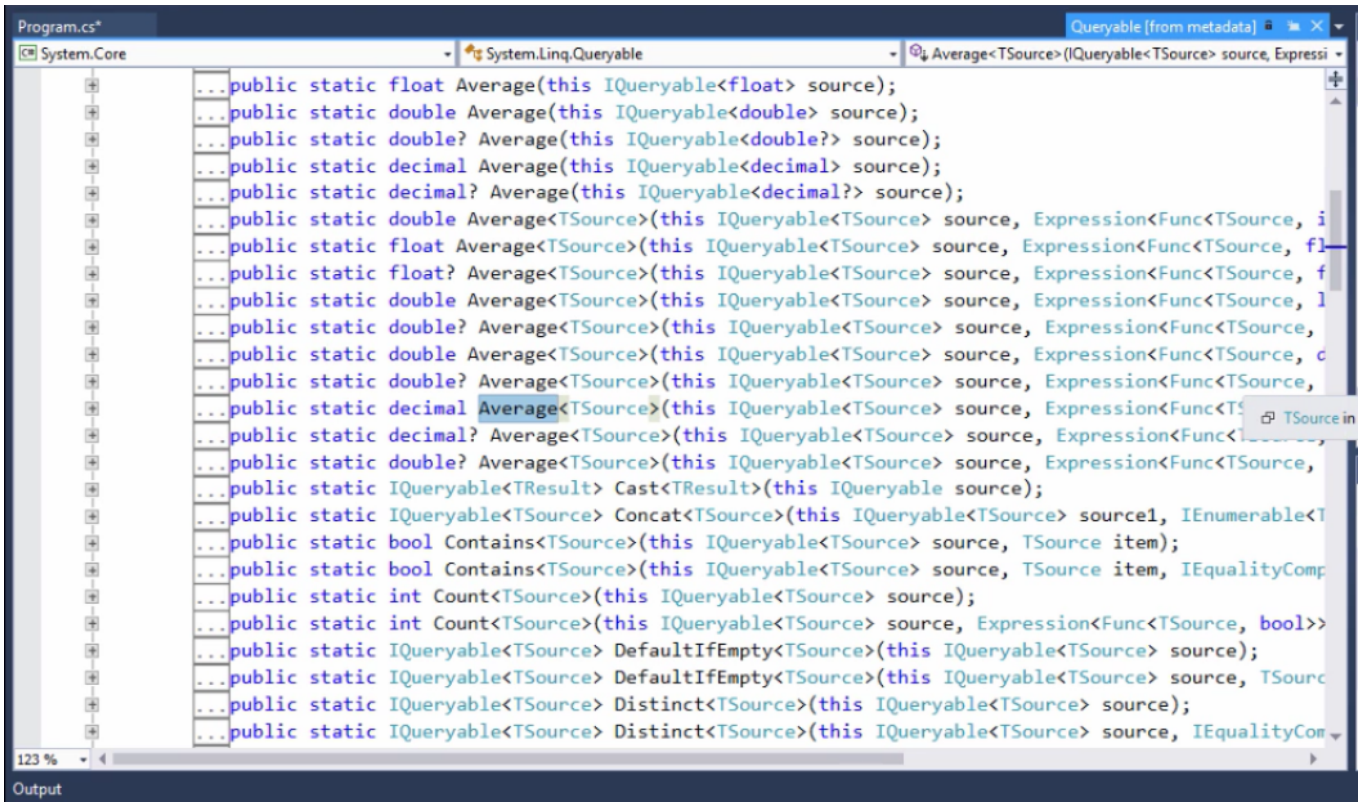
```
var elementoCentral_1 = queryOrdenada.Skip(contagem - 1) / 2).First();
```

Agora temos os dois elementos centrais!

Neste instante é preciso calcular a mediana, assim, somamos os elementos centrais e dividimos por dois:

```
var mediana = (elementoCentral_1 + elementoCentral_2) / 2
```

Feitos estes ajustes, poderemos seguir em frente e extrair o método de extensão. Para fazer isso é preciso criar uma classe estática para armazenar o método. Abaixo do `var mediana` vamos adicionar o `static class` e junto disso adicionreamos o `LinqExtension`. Também vamos incluir o método extensão. É preciso que o método contenha uma assinatura. Selecionaremos o `Average` e apertamos o "F12" e vamos para a página das definições:



Pegamos a assinatura do método `Average`, copiamos ela, retornamos ao código e colocamos isso no `static class`. Assim, ao em vez de usar o `Average` vamos chamar o método de `Mediana`. Teremos:

```
static class LinqExtension
{
    public static decimal Mediana<TSource>(this IQueryable<TSource> source, Expression<Func<TSource,
}
```

O próximo passo é pegar o corpo da função `Mediana` e colar isso dentro do método extensão. Ao fazer isso teremos:

```
static class LinqExtension
{
    public static decimal Mediana<TSource>(this IQueryable<TSource> source, Expression<Func<TSource,
    {

        var contagem = query.Count();

        var queryOrdenada = query.OrderBy( total => total);

        var elementoCentral_1 = queryOrdenada.Skip(contagem / 2).First();

        var elementoCentral_2 = queryOrdenada.Skip(contagem / 2).First();

        var mediana = (elementoCentral_1 + elementoCentral_2) / 2;
        return mediana;
}
```

Porém, se rodarmos o código ocorrerá um erro, pois estamos utilizando a variável `query` que não está sendo recebida pelo método. Assim, vamos trocar `query` por `source` que é o nome da variável que chega no método.

Outro problema é a impossibilidade de interpretar diferentes elementos. Isso ocorre, pois nosso desejo inicial era fazer uma operação para pegar o `selector` da `query`. O `selector` é aquilo que dirá para a consulta qual campo desejamos selecionar, assim, vamos utilizá-lo na função `mediana`. Entretanto, antes de usá-lo é preciso criar uma função que o compile. Dessa maneira, escreveremos:

```
var contagem = source.Compile()
```

O que acontece é que o `selector` ainda é apenas uma definição e não uma função, portanto quando isso for compilado será realmente transformado em um código para ser utilizado na consulta. Também vamos modificar a `queryOrdenada` para que o `select` seja feito em cima da função do `Seletor`. O código fica com o seguinte aspecto:

```
static class LinqExtension
{
    public static decimal Mediana<TSource>(this IQueryable<TSource> source, Expression<Func<TSource,
    {

    var contagem = source.Count();

    var funcSelector = selector.Compile();

    var queryOrdenada = source.Select(funcSelector).OrderBy(total => total);

    var elementoCentral_1 = queryOrdenada.Skip(contagem / 2).First();

    var elementoCentral_2 = queryOrdenada.Skip(contagem / 2).First();

    var mediana = (elementoCentral_1 + elementoCentral_2) / 2;
    return mediana;
```

Após realizar todas estas modificações, vamos utilizar o método de extensão `mediana`. Portanto, vamos acrescentar a variável `vendaMediana` que deve pegar os dados de contexto, assim, adicionamos `contexto.NotasFiscais.mediana()` que será mais um método na biblioteca. Dentro disso, passaremos também qual campo deve ser calculado para chegarmos no valor da mediana, dessa forma, introduzimos o `nf => nf.Total`. Teremos:

```
Console.WriteLine("Mediana: {0}", mediana);

var vendaMediana = contexto.NotasFiscais.Mediana(nf => nf.Total);

Console.ReadKey();
```

Agora, temos a `Mediana` no formato `contexto.NotasFiscais.Mediana(nf => nf.Total)` que equivale a chamada da média, a `contexto.NotasFiscais.Average(nf => nf.Total)`. Esta modificação serviu para trazer maior consistência e além disso é mais adequada e funcional, porém não tão elegante quanto a que acabamos de desenvolver.

Falta imprimir o resultado no `Console`, portanto escreveremos:

```
Console.WriteLine("Mediana (come método de extensão): {0}", vendaMediana)
```

O código fica da seguinte maneira:

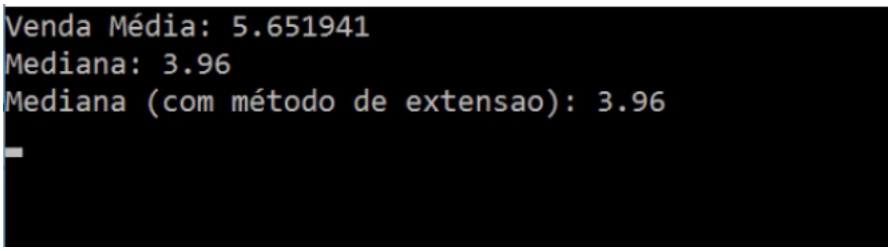
```
Console.WriteLine("Mediana: {0}", mediana);

var vendaMediana = contexto.NotasFiscais.Mediana(nf => nf.Total);

Console.WriteLine("Mediana (come método de extensão): {0}", vendaMediana);

Console.ReadKey();
```

Teremos o seguinte:



O resultado obtido é além da `Venda Média` também a `Mediana`:

```
Mediana(com método de extensao)
```

Ou seja, fomos capazes de estender e expandir a biblioteca de `LINQ` utilizando uma função própria. Também reproduzimos esse comportamento para infinitos tipos de função que desejamos agregar ao código!