02

2 - Linq métodos extensão

Transcrição

Para esta aula o cliente requisitou o cálculo da mediana das vendas.

A mediana consiste em pegar uma lista, ordenar linearmente seus valores e selecionar os elementos centrais. No caso de objetos em quantidade par, os dois itens que ocupam o centro são somados e divididos para resultar no valor da mediana. No caso de um valor ímpar, basta selecionar o item central.

Agora que sabemos o que é uma mediana, vamos aplicar isso ao código! Nosso objetivo é encontrar a mediana das vendas usando as Notas Fiscais, portanto vamos criar uma nova consulta. Primeiro, inserimos um comentário apenas para lembrar qual o valor da média, `//Venda Média: 5,651941` .

Pegamos a nota fiscal e inserimos uma `query` . Escrevemos `contexto.NotasFiscais` e junto disso, normalmente iríamos inserir uma função com o nome `Mediana` , mas como ela não existe será preciso criar uma nova função para calcular a mediana. Apesar de um pouco trabalhoso, customizar uma função facilitará nossa vida!

Assim, vamos adicionar a variável `query` :

```
var query =
from ng In contexto.NotasFiscais
select nf;
```

Falta pegar o objeto central. Para isto, é preciso pular uma certa quantidade de elementos, portanto utilizaremos o método `Skip()` . O problema é que não temos certeza sobre a quantidade usada, assim, vamos inserir o `query.Skip(contagem)` . Antes do `Skip()` , declararemos uma variável e armazenaremos uma quantidade de elementos para a consulta. Teremos a variável `contagem` que será igual a `query.Count()` . No caso de objetos pares, a contagem será feita pegando o número total e dividindo por dois para chegar na metade, portanto adicionaremos `query.Skip(contagem / 2)` . Além de fazer esse cálculo, é preciso também pegar um elemento, assim, vamos acrescentar o `First` .

Teremos:

```
//Venda Média: 5.651941

var query =
from ng In contexto.NotasFiscais
select nf;

var contagem = query.Count();

query.Skip(contagem / 2).First();

Console.ReadKey();
```

O resultado da consulta são todos os elementos ordenados! O que fizemos na consulta foi pedir para pular os elementos - Skip - e pegar o primeiro elemento - First.

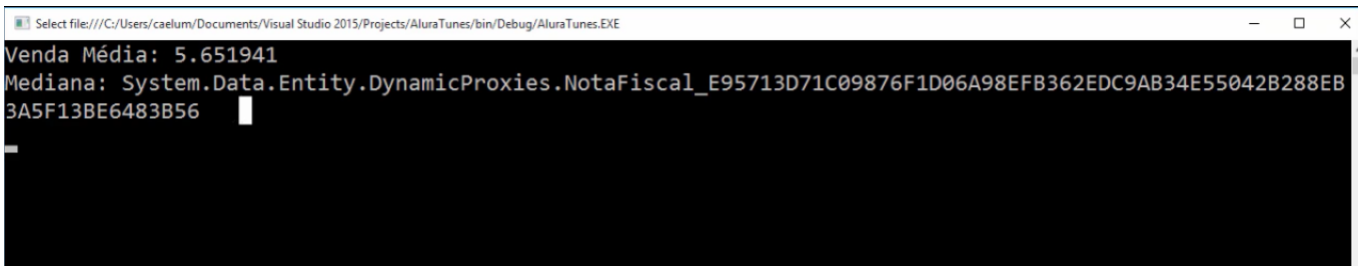
Mas, ainda falta pegar o elemento central. Assim, vamos utilizar a variável `elementoCentral`. Ainda, vamos nomear a `var mediana` que será igual a `elementoCentral`. Também passaremos para o `Console.WriteLine()` o `"Mediana: {0}", mediana`:

```
//Venda Média: 5.651941

var query =
from ng In contexto.NotasFiscais
select nf;

var contagem = query.Count();

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

var mediana = elementoCentral;

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

Console.ReadKey();
```

Ao rodar a aplicação, um erro é indicado. Ele ocorreu, pois é preciso ordenar a consulta antes de utilizar o `Skip()`. Para solucionar a situação adicionaremos uma variável que armazena a nova consulta, a `queryOrdenada`, e equivalemos isso a `query.OrderBy()`. Ainda, para o `query.OrderBy()` introduzimos a expressão `lambda`, a `nf => nf.Total`. Por fim, junto do `query.Skip` colocamos `Ordenada`. Teremos:

```
//Venda Média: 5.651941

var query =
from ng In contexto.NotasFiscais
select nf;

var contagem = query.Count();

var queryOrdenada = query.OrderBy(inf => nf.Total);

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


var mediana = elementoCentral;
```

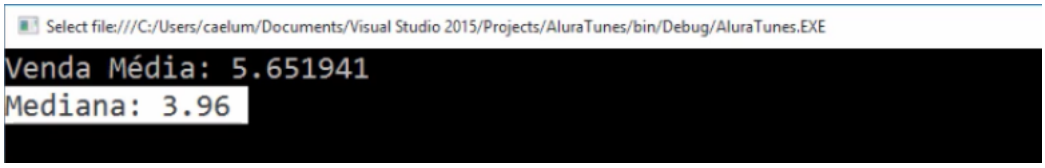
Rodando o código é mostrado o seguinte resultado:



Um objeto é trazido, mas nosso desejo era que o resultado fosse um valor! Como queremos o valor total é preciso corrigir o código e escrever `select nf.Total`. Feito isso é necessário também substituir o `nf.Total` do `OrderBy` para `total => total`. O código fica da seguinte maneira:

```
var query =
from nf in contexto.NotasFiscais
select nf.Total;

var contagem = query.Count();

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

var elementoCentral =
    query Ordenada.Skip(contagem/ 2).First();
```

Agora, ao rodar a consulta temos o seguinte resultado:



A mediana possui o valor de exato de `3.96` !