01

1- Linq to entities - Linq paralelo

Transcrição

A empresa virtual, **Alura Tunes**, tem interesse em realizar uma ação de marketing: colocar em diversas cidades do país folders que contenham um QR Code que represente músicas da loja virtual, assim, pessoas cadastradas na **Alura Tunes** e que coletarem um total de 20 códigos ganharão uma assinatura anual da loja.

Nossa tarefa, como desenvolvedores, é criar ferramentas para gerar um QR Code - uma tarefa complicada. Mas existem bibliotecas que podem nos auxiliar nesta tarefa... Vamos utilizar o `NuGet` - ferramenta pertencente ao **Visual Studio** - para instalar pacotes de bibliotecas. Nosso objetivo é obter uma biblioteca chamada `ZXing`, para instalá-la usamos o atalho "Ctrl + Q" e na caixa de texto que aparece digitamos `NuGet`, ao finalizar o carregamento encontramos as seguintes informações:

Escolhemos a opção "NuGet Packet Manager" e aguardaremos até que o gerenciador de pacotes seja aberto. Após o carregamento podemos digitar:

```
Install-Package ZXing.Net
```

Ao dar um "Enter" a biblioteca começa a ser instalada:

Uma vez concluída a instalação, podemos fazer um teste rápido para verificar se as imagens são geradas com sucesso. Vamos declarar uma variável que desenhe os `QR Codes` para a simulação: `BarcodeWriter()` e igualaremos isso a `new BarcodeWriter()`. Como queremos que o resultado tenha um padrão visual `QR Code`, vamos juntar o `barcodeWriter` com `Format` e igualaremos a `BarcodeFormat.QR_CODE`. O código ficará com da seguinte forma:

```
class Program
{
    static void Main(string[] args)
    {
        var barcodeWriter = new BarcodeWriter();
        barcodeWriter.Format = BarcodeFormat.QR_CODE;
    }
}
```

O **Visual Studio** indica um erro no `barcodeWriter`:

Isso ocorre pois o formato `Bitmap` não está definido para a aplicação. O `Bitmap` é o que permite que a imagem `QR Code` seja gerada, portanto, para solucionar a falha é preciso adicionar nas referências o `System.Drawing`:

Ainda, é preciso definir o tamanho da imagem que desejamos obter - a medida deverá ser estabelecida em pixels. Vamos incluir dentro a largura e altura, `Width = 100` e `Weight = 100`, respectivamente:

```
class Program
{
    static void Main(string[] args)
    {
        var barcodeWriter = new BarcodeWriter();
        barcodeWriter.Format = BarcodeFormat.QR_CODE;
```

```
            barcodeWriter.Options = new ZXing.Common.EncodingOptions
            {
                Width = 100,
                Height = 100
            };
        }
    }
```

Um código de barra armazena apenas números, o `QR Code` em comparação, é mais sofisticado a ponto de armazenar outras informações. Assim, vamos adicionar o `barcodeWriter.Write()` e passar para ele uma `string` que contenha as informações. Elas deverão ser inseridas no `QR Code` . Portanto, vamos escrever `barcodeWriter.Write("Meu teste")` . Após criar parte do conteúdo do `QR Code` é necessário também salvar o que fizemos dentro de um arquivo para acessá-lo, dessa forma, adicionaremos o `Save()` e depois, passaremos o nome do arquivo e o formato da imagem: `QRCode.jpg` e `ImageFormat.Jpeg` . O código ficará da seguinte maneira:

```
class Program
{
    static void Main(string[] args)
    {
        var barcodeWriter = new BarcodeWriter();
        barcodeWriter.Format = BarcodeFormat.QR_CODE;
        barcodeWriter.Options = new ZXing.Common.EncodingOptions
        {
            Width = 100,
            Height = 100
        };

        barcodeWriter.Write("Meu teste").Save("QRCode.jpg", ImageFormat.Jpeg);

    }
}
```

Agora que já temos isso salvo em um arquivo, é preciso confirmar se ele realmente foi guardado! Para conferir basta acessar o caminho da pasta onde nosso projeto está salvo, "AluraTunes > bin > Debug". O resultado será o seguinte:

Esse código contem uma mensagem. Para verificar qual é ela, vamos utilizar um site Webqr (webqr.com) que faz a leitura do `QR Code` . Basta inserir dentro do campo que aparece o arquivo que acabamos de construir. Fazendo isso teremos como resultado a mensagem que aparece gravada no código:

A mensagem que aparece é "Meu código", exatamente o mesmo conteúdo que nós gravamos no código!

O intuito é gravar em cada código uma música da loja virtual! Uma vez alcançado o resultado, vamos inserir uma pasta para guardar os demais códigos, assim, caso ela não exista, vamos colocar a condição para que ela seja criada. Portanto, adicionaremos o `if (!Directory.Exists("Imagens")` :

```
if (!Directory.Exists("Imagens"))
   Directory.CreateDirectory("Imagens");
```

Observe que substituímos o "Imagens" por uma constante usada ao longo de todo código.

Ainda, vamos varrer as faixas da base de dados e para cada uma, será criado um código `QR Code` !

Para começar a fazer isso é preciso preparar um contexto. Faremos issocom a seguinte linha:

```
using (var contexto = new AluraTunesEntities())
```

Dentro disso, vamos declarar uma consulta que pega e traz todas as faixas. Faremos isso dentro da variável `queryFaixas` :

```
using (var contexto = new AluraTunesEntities())
{
var queryFaixas =
from f in contexto.Faixas
select f;

}
```

Na sequência, é preciso materializar a `query` em uma lista em memória. Assim, utilizaremos a variável `listaFaixas` . Feito isso podemos criar outra `query` , que irá efetivamente gerar códigos do tipo QR Code. Inicialmente, construiremos uma consulta simples, portanto, acrescentaremos a `queryCodigos` e, dentro dela, iremos inserir `listaFaixas.Select(f => f)` . Com isso, adicionaremos também o `foreach` e o `Console.WriteLine` que deve imprimir o resultado:

```
using (var contexto = new AluraTunesEntities())
{
var queryFaixas =
from f in contexto.Faixas
select f;

var listaFaixas = queryFaixas.ToListt();

var queryCodigos = listaFaixas.Select(f => f);

foreach (var item in queryCodigos)
{
    Console.WriteLine(item.FaixaId);
}
```

O próximo passo é modificar a `query` acrescentando as informações necessárias para que o código seja gerado. Para fazer isso, é preciso modificar o `select` para que ele traga um objeto anônimo `listaFaixas.Select(f => new)` . Dentro disso, introduziremos o `Arquivo` que será igual a `string.Format()` . Assim, passaremos o caminho em que o documento deverá ser salvo e o nome da pasta onde deverão ser guardados os arquivo:

```
Arquivo = string.Format(“{0}\\{1}”. jpg, Imagens, f.FaixaId)
```

O código ficará da seguinte maneira:

```
var queryCodigos = listaFaixas.Select(f => new
{
    Arquivo = string.Format(“{0}\\{1}. jpg”, Imagens, f.FaixaId)

});

foreach (var item in queryCodigos)
```

```
{
    Console.WriteLine(item.FaixaId);
}
```

Falta trazer a representação visual do `QR Code`, dessa maneira, adicionaremos `barcodeWriter.Write()`. O `Write` serve para "encodar", transformar a mensagem que vamos passar em código! Adicionaremos uma mensagem que contenha a URL da loja virtual e, junto disso, é preciso introduzir o `Id` de cada faixa, usando uma `string` de formatação:

```
barcode.Writer.Write(string.Format( "aluratunes.com/Faixa/{0}", f .faixaId))
```

Essa propriedade necessita de um nome, portanto, iremos chamá-la de `Imagens`:

```
var queryCodigos = listaFaixas.Select(f => new
{
    Arquivo = string.Format("{0}\\{1}.jpg", Imagens, f.FaixaId),
    Imagem = bancodeWriter.Write(string.Format("aluratunes.com/faixa/{0}", f.faixaId))

});
```

Agora, vamos substituir a linha do `foreach` por um comando que salve as imagens em arquivos. Ao usarmos o `item.Imagem.Save()`, cada imagem deverá ser salva em um local específico. Portanto, falta especificar onde o arquivo deverá ser gravado e também qual o formato da imagem:

```
item.Imagem.Save(item.Arquivo, ImageFormat.jpeg)
```

O código ficará da seguinte maneira:

```
var queryCodigos = listaFaixas.Select(f => new
{
    Arquivo = string.Format("{0}\\{1}.jpg", Imagens, f.FaixaId),
    Imagem = bancodeWriter.Write(string.Format("aluratunes.com/faixa/{0}", f.faixaId))

});

foreach (var item in queryCodigos)
{
    item.Imagem.Save(item.Arquivo, ImageFormat.Jpeg);
}
```

A próxima tarefa é quantificar o tempo de espera para que as imagens sejam geradas. Para isso, utilizaremos o `StopWatch` que é um componente que funciona como um cronômetro. Vamos declará-lo acima da variável `queryCodigos`, assim, adicionaremos `Stopwatch` e junto o `StartNew()` para criar novas informações que devem ser armazenadas no `Stopwatch stopwatch`. Fazendo isso, a instância é criada e o cronômetro começa a contar:

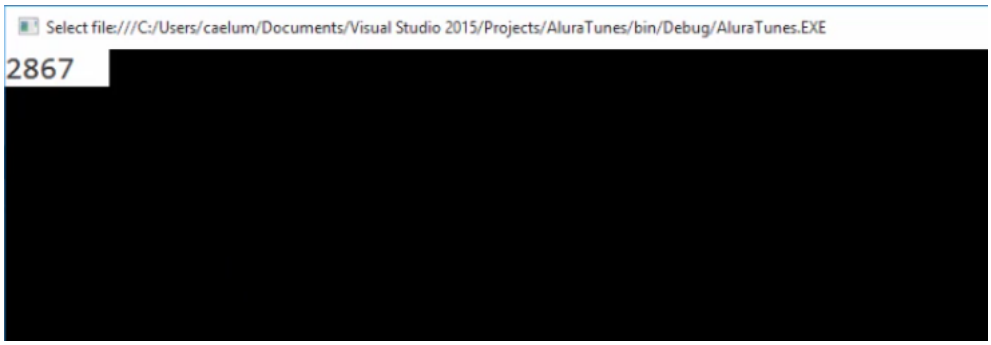
```
var listaFaixas = queryFaixas.ToList();

Stopwatch stopwatch = Stopwatch.StartNew();
```

O próximo passo é fazer uma contagem para verificar quantas linhas de elementos são retornadas!

Assim, vamos adicionar acima do `foreach()` o `queryCodigos.Count()` e, nele, armazenaremos a contagem na variável `int contagem` para verificar quanto tempo demorou a execução da consulta e o carregamento das imagens. Após medir o tempo é preciso pedir ao cronômetro para que ele pare de contar, assim, vamos introduzir o `stopwatch.Stop()`, justo abaixo do que acabamos de inserir. Como queremos que o tempo de duração da execução seja impresso, vamos adicionar o `Console.WriteLine()`. Em seguida, passaremos para ele o `stopwatch` e também a quantidade de tempo em milissegundo: `ElapsedMilliseconds`:

```
int contagem = queryCodigos.Count();

stopwatch.Stop();

Console.WriteLine(stopwatch.ElapsedMilliseconds);

foreach (var item in queryCodigos)
```

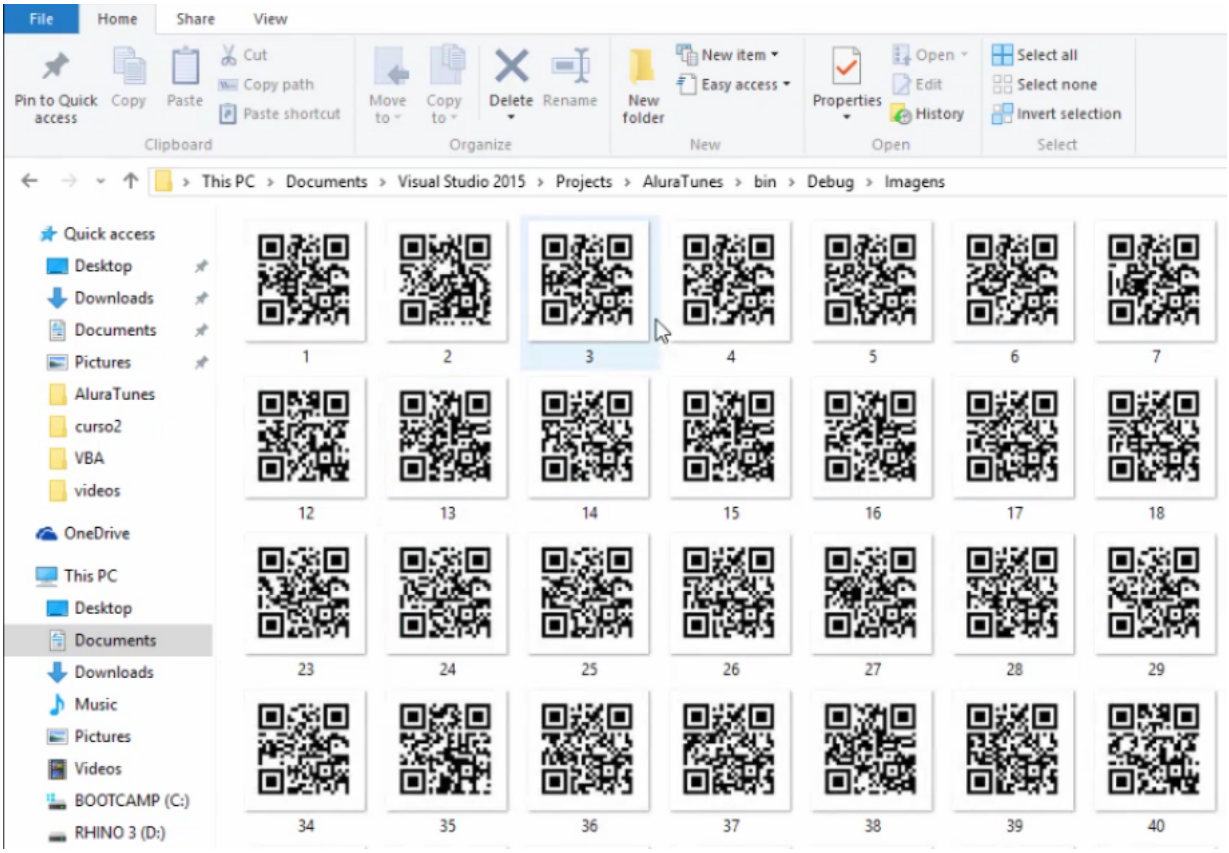
Ao rodarmos a aplicação, o resultado será o seguinte:



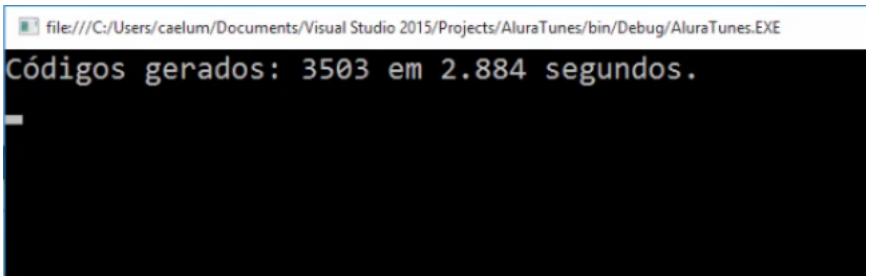
O resultado retornará um valor em milissegundo, ao dividirmos isso por `1000.0`, obteremos uma medida em segundos. Assim, vamos modificar o `Console.WriteLine()` para: `"Códigos gerados: {0} em {1} segundos."`. Adicionaremos também a variável `contagem` e o código fica da seguinte maneira:

```
int contagem = queryCodigos.Count();

stopwatch.Stop();

Console.WriteLine("Códigos gerados: {0} em {1} segundos.", contagem, stopwatch.ElapsedMilliseconds /

foreach (var item in queryCodigos)
{
    item.Imagem.Save(item.Arquivo, ImageFormat.Jpeg);
}
```

Vamos verificar também como os arquivos estão salvos em nossa pasta. Abrindo a pasta "Imagens", encontraremos diversas fotos referentes ao `QR Code`:



Ao rodarmos a aplicação, teremos o seguinte resultado:

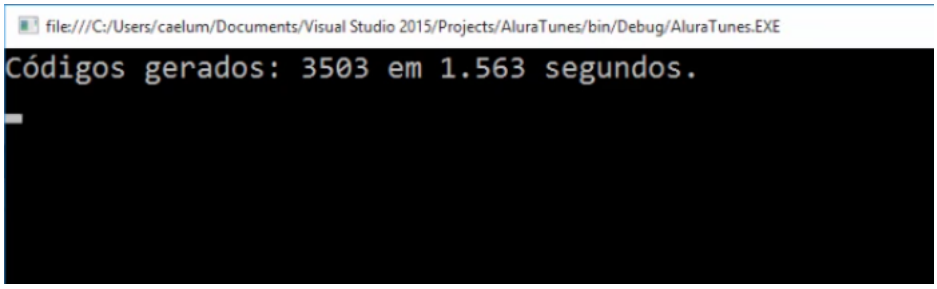


Foram gerados `3503` códigos em `2,8` segundos! A pergunta é: Podemos melhorar a performance da consulta?

Vamos distribuir a tarefa de gerar imagens entre diversos núcleos do processador para alcançar um resultado mais rápido! Assim, vamos modificar a `query` usando o comando `.AsParallel()` antes do `select`, como isto iremos paralelizar a tarefa de gerar código em diversos núcleos:

```
var queryCodigos =
listaFaixas
.AsParallel()
.Select(f => new
{
    Arquivo string.Format("{0}\\{1}.jpg", imagens, f.FaixaId),
    Imagem = barcodeWriter.Write(string.Format("aluratunes.com/faixa{0}", f.FaixaId))
});
```

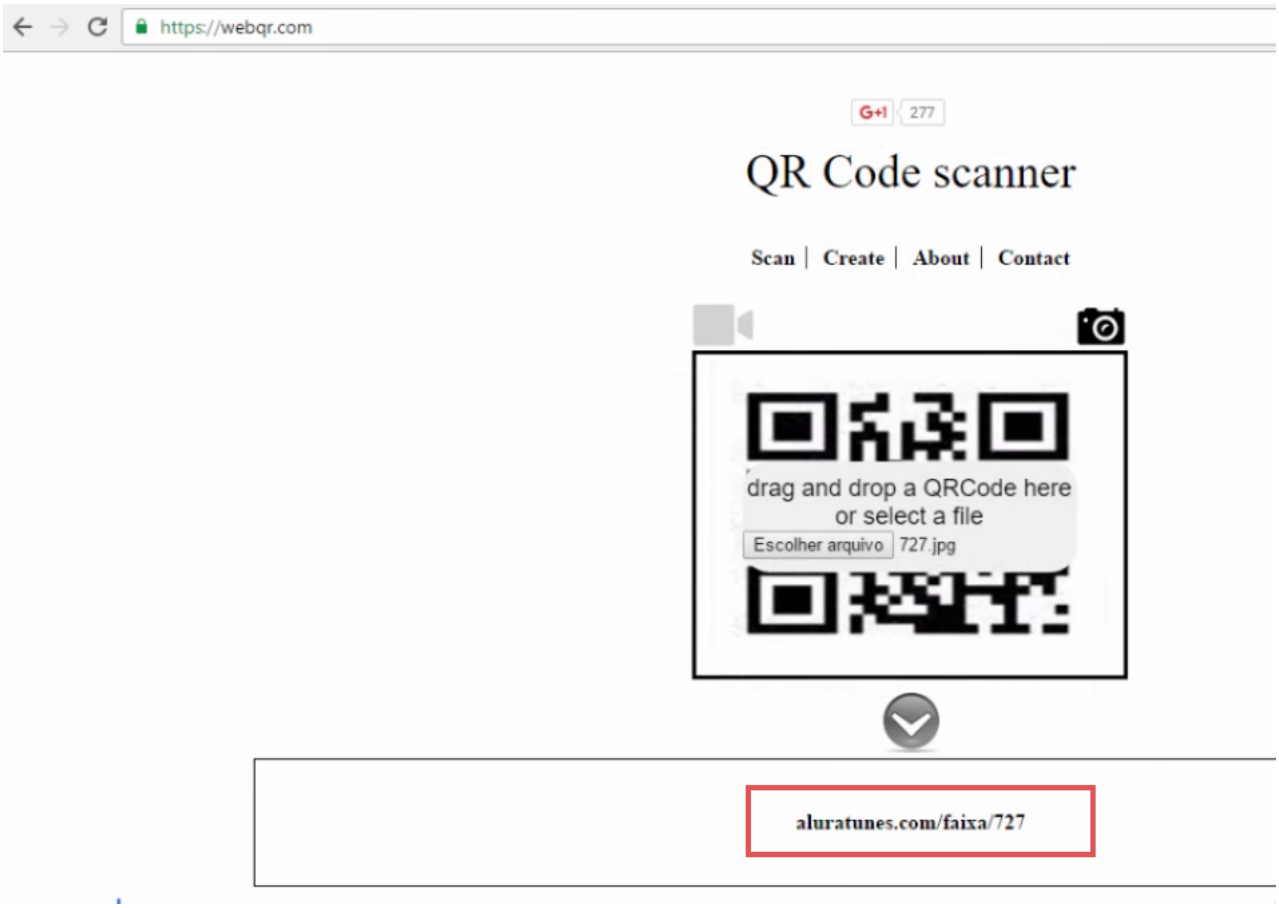
Ao rodar a aplicação temos o seguinte resultado:



Ou seja, conseguimos diminuir o tempo de `2,8` segundos para `1,563` !

O resultado do procedimento foi a diminuição sensível do tempo de execução e carregamento de imagens graças ao comando `AsParallel()` .

Para verificar a mensagem agregada ao `QR CODE` , vamos jogar o arquivo de nome `727` no site [Webqr (webqr.com)]:



Recapitulando:

Nesta aula, mostramos como construir um `QR Code` para utilizar em uma ação de Marketing da loja **AluraTunes**. Fizemos uso de uma biblioteca específica a `XZing` e observamos que gerar códigos pode demorar um pouco, portanto, utilizamos a estratégia de paralelizar o código para economizar tempo!