02

2 - Linq to entities paginado

Transcrição

O novo pedido do cliente é: produzir um relatório detalhado de vendas do sistema. O objetivo é que o relatório seja paginado, portanto, é preciso quebrar o relatório em páginas diferentes e cada uma deve conter um máximo de dez linhas.

Já vimos que uma venda é representada, no sistema, usando a Nota Fiscal. Desta maneira, no arquivo `Program.cs`, vamos inserir uma classe e dentro dela colocaremos o `using (var contexto = new AluraTunesEntities())` e a partir do contexto, nós vamos trazer a `NotasFiscais`. Portanto, acrescentaremos o `from nf in contexto.NotasFiscais` e abaixo o `select new`. Feito isso, é preciso adicionar a variável: `var query =`. Ao final, iremos inserir a impressão do resultado, portanto, colocamos `foreach (var nf in query)` e `Console.WriteLine(nf.Total)`. O código ficará da seguinte maneira:

```
class Program
{
    static void Main(string[] args)
    {
        using (var contexto = new AluraTunesEntities())
        {
            var query =
            from nf in contexto.NotasFiscais
            select new;

            foreach (var nf in query)
            {
                Console.WriteLine(nf.Total);
            }
        }
    }
}
```

Ao rodarmos o código, serão trazidos apenas os valores das Notas Fiscais.

Agora, vamos expandir o relatório inserindo mais colunas nele!

Para fazer isso, vamos inserir um objeto anônimo e dessa forma o relatório será exibido. Assim, adicionamos: `Numero = nf.NotaFiscalId`, `Data = nf.DataNotaFiscal`, `Cliente = nf.Cliente.PrimeiroNome + " " + nf.Cliente.Sobrenome` e, por fim, o `Total = nf.Total`. O código ficará da seguinte maneira:

```
var query =
from nf in contexto.NotasFiscais
select new;
{
Numero = nf.NotaFiscalId,
Data = nf.DataNotaFiscal,
Cliente = nf.Cliente.PrimeiroNome + " " + nf.Cliente.Sobrenome,
Total = nf.Total
}
```

Ao acrescentarmos estas informações é preciso inseri-las também no `Console.WriteLine()`. Vamos, portanto, adicionar: `nf.Numero, nf.Data, nf.Clientenf, nf.Total`. Ao fazer isso teremos um total de 4 colunas a serem impressas. Falta inserir uma string de formatação a fim de indicar a posição das colunas: `"{0}\t{1}\{2}\{3}"`:

```
ConsoleWriteLine("{0}\t{1}\{2}\{3}", nf.Numero, nf.Data, nf.Clientenf, nf.Total)
```

Ao rodarmos o código teremos as quatro colunas!

Falta paginar o relatório e para que isso aconteça é necessário, primeiro, quebrar o relatório em páginas. Portanto, vamos iniciar mostrando apenas a primeira. Para que isso ocorra é preciso utilizar um filtro que limite o número de linhas mostradas em cada página, no caso, dez linhas. Desta forma, acrescentaremos uma `query` e junto dela vamos inserir um comando, o `Take()` que possui justamente a função de pegar. Como queremos que sejam selecionadas dez linhas, passaremos o valor `10`:

```
query = query.Take(10),
```

Com isso, a aplicação mostrará a primeira página e nela aparecem apenas as dez linhas que filtramos. Lembrando que para a informação seguir aparecendo é preciso inserir o `Console.ReadKey()`. Por enquanto o código está com o seguinte aspecto:

```
class Program
{
    static void Main(string[] args)
    {
        using (var contexto = new AluraTunesEntities())
        {
            var query =
            from nf in contexto.NotasFiscais
            select new;
            {
            Numero = nf.NotaFiscalId,
            Data = nf.DataNotaFiscal,
            Cliente = nf.Cliente.PrimeiroNome + " " + nf.Cliente.Sobrenome,
            Total = nf.Total
            };

            query = query.Take(10),

            foreach (var nf in query)
            {
                ConsoleWriteLine("{0}\t{1}\{2}\{3}", nf.Numero, nf.Data, nf.Clientenf, nf.Total);
            }

            Console.ReadKey();
        }
    }
}
```

É preciso refatorar o `10`, pois, a boa prática indica que é melhor extrair uma variável ou constante que possibilite fazer a modificação uma única vez e em um mesmo local. Portanto, basta selecionar o `10` e usar o comando "Ctrl + .". Aparecerão algumas possibilidades de refatoração, das quais selecionaremos o `Introduce local constant for '10'`. Depois,

renomearemos a constante para `TAMANHO_PAGINA`. O maiúsculo será utilizado, pois este é um padrão de constante. Teremos o seguinte:

```
            select new
            {
            Numero = nf.NotaFiscalId,
            Data = nf.DataNotaFiscal,
            Cliente = nf.Cliente.PrimeiroNome + " " + nf.Cliente.Sobrenome,
            Total = nf.Total
            };

            query = query.Take(TAMANHO_PAGINA);

            foreach (var nf in query)
            {
                ConsoleWriteLine("{0}\t{1}\{2}\{3}", nf.Numero, nf.Data, nf.Clientenf, nf.Total);
            }
```

Vamos reposicionar o `const int TAMANHO_PAGINA = 10;` para baixo do `static void Main(string[] args)`.

Ao rodarmos a aplicação, nada estará quebrado! Agora, para trazer a segunda página é preciso pular a quantidade de linhas equivalente a primeira página. Para fazer isso é preciso modificar a query. Portanto, utilizamos o comando equivalente ao ato de saltar: o `skip()`. Para este comando, nós podemos passar a constante que determinamos anteriormente, a `TAMANHO_PAGINA`.

```
query = query.Skip(TAMANHO_PAGINA);

query = query.Take(TAMANHO_PAGINA);
```

Porém, um erro ocorre! O `skip` só pode ser utilizado em consultas ordenadas, o que não é o caso.

Para fazer a consulta funcionar é preciso ordená-la. Dessa maneira, adicionamos o `orderby` e junto disso inserimos também o `NotaFiscalId` que indica por onde a ordenação deve se guiar:

```
var query
from nf in contexto.NotasFiscais
orderby nf.NotaFiscalId
select new
    {
        Numero = nf.NotaFiscalId,
        Data = nf.DataNotaFiscal,
        Cliente = nf.Cliente.PrimeiroNome + " " + nf.Cliente.Sobrenome,
        Total = nf.Total
    };
```

Ao rodar o código o resultado é uma ordenação conforme o número da nota fiscal. Inclusive, os dez primeiros resultados não serão mostrados, pois correspondem a primeira página.

Por fim, antes que todas as páginas do relatório sejam impressas é preciso realizar algumas modificações! Por exemplo, para chegar a terceira página, quantas linhas devem ser puladas? Neste caso, é preciso utilizar uma conta que nos auxilie nesta tarefa! Vamos pensar um pouco: o número que saltamos de páginas é o número da página multiplicado pelo número de

linhas de cada página. Mas, as páginas não começam a contar a partir de `1` e, sim, do `0`, uma vez que a página `1` não precisa que sejam puladas linhas.

A lógica é a seguinte:

```
PÁGINA      SALTAR
1            0 = 0X10 = 0X
2            10 = 1X10 = 1X
3            20 = 2X10 = 2X
4             30 = 3X30 = 3X
```

A fórmula que encontrada será a seguinte:

```
(página - 1) x Tamanho Página
```

No código nós vamos adicionar o seguinte:

```
numeroDePulos = (numeroPagina - 1) * TAMANHO_PAGINA;
```

Teremos:

```
numeroDePulos = (numeroPagina - 1) * TAMANHO_PAGINA

query = query.Skip(TAMANHO_PAGINA);

query = query.Take(TAMANHO_PAGINA);
```

A variável `numeroPagina`, entretanto, ainda não existe! Por isso, é preciso criá-la:

```
int numeroPagina = 1;
int numeroDepulos = (numeroPagina - 1) * TAMANHO_PAGINA;

query = query.Skip(TAMANHO_PAGINA);

query = query.Take(TAMANHO_PAGINA);
```

Vamos deslocar o `int numeroPagina = 1;` para antes da `var query`. Ao fazer isso, modificaremos o `TAMANHO_PAGINA` por `numeroDePulos`:

```
int numeroPagina = 1;

var query
from nf in contexto.NotasFiscais
orderby nf.NotaFiscalId
select new
    {
        Numero = nf.NotaFiscalId,
        Data = nf.DataNotaFiscal,
        Cliente = nf.Cliente.PrimeiroNome + " " + nf.Cliente.Sobrenome,
```

```
            Total = nf.Total
        };


    int numeroDepulos = (numeroPagina - 1) * TAMANHO_PAGINA;

    query = query.Skip(numeroDePulos);

    query = query.Take(TAMANHO_PAGINA);
```

Ao inserirmos as modificações podemos verificar que a primeira página estará funcionando conforme esperado. Ao modificar o `int numeroPagina = 1;` para `int numeroPagina = 2;` observamos as linhas de 11 a 20 e assim por diante!

Após ordenar a consulta, é preciso, finalmente, imprimir o relatório inteiro. Primeiro vamos refatorar o código, pois ele já está muito grande. Isto é, vamos extrair uma função que deve imprimir uma página. Assim, selecionamos o código e damos um "Ctrl + ." e aparecerá a opção "Extract Method". O novo método receberá o nome de `ImprimirPagina` . Nosso código ficará da seguinte maneira:

```
const int TAMANHO_PAGINA = 10;

using (var contexto = new AluraTunesEntities())
{
    int numeroPagina = 3;

    ImprimirPagina(TAMANHO_PAGINA, contexto, numeroPagina);

    Console.ReadKey();
}
```

O problema é que, da maneira como está escrito, parece que a impressão deve ser feita apenas para uma página. Nosso objetivo é que as páginas sejam impressas uma por uma até o final. Portanto, primeiro, vamos mensurar a quantidade de linhas totais. Para isso, vamos introduzir `contexto.NotasFiscais.Count()` , no qual o método `Count()` é responsável pela contagem. O resultado disso nós armazenaremos na variável `numeroNotasFiscais` . Falta, ainda, calcular a quantidade total de páginas do relatório e para chegar a este número vamos pegar o número de linhas do relatório e dividir pelo tamanho da página: `numeroNotasFiscais /TAMANHO_PAGINA` . Isso nós armazenaremos na variável `numeroPaginas` . Nosso código ficará da seguinte maneira:

```
const int TAMANHO_PAGINA = 10;

using (var contexto = new AluraTunesEntities())
{
    int numeroPagina = 3;

    var numeroNotasFiscais= contexto.NotasFiscais.Count();
    var numeroPaginas = numeroNotasFiscais / TAMANHO_PAGINA;

    ImprimirPagina(TAMANHO_PAGINA, contexto, numeroPagina);

    Console.ReadKey();
}
```

O problema é que a conta não resulta em um número correto de páginas. É preciso arredondar os números e a fim de solucionar o impasse vamos utilizar um método presente na biblioteca, o `Math.Ceiling()`. A palavra *ceiling*, em inglês, significa teto, isto é, o arredondamento será para cima. Porém, ao utilizar a função, `Math.Ceiling()` também é preciso converter a fórmula para decimal para que ela de fato funcione. Teremos: `Math.Ceiling((decimal)NumeroNotasFiscais / TAMANHO_PAGINA)`.

Ainda falta imprimir as páginas do relatório e para fazer isso, nós vamos criar um laço `for` e junto dele nós inserimos a variável `p` que equivale a páginas. Assim, `p = 1; p < numeroPaginas; p++`. Dentro do `for` nós colocamos o `ImprimirPagina(TAMANHO_PAGINA, contexto, numeroPagina);`. Teremos o seguinte:

```
using (var contexto = new AluraTunesEntities())
{
    int numeroPagina = 3;

    var numeroNotasFiscais= contexto;NotasFiscais.Count();
    var numeroPaginas = numeroNotasFiscais / TAMANHO_PAGINA;

    for (var p = 1; p < numeroPaginas; p++)
    {

        ImprimirPagina(TAMANHO_PAGINA, contexto, numeroPagina);
    }

    Console.ReadKey();
}
```

O problema disso é que não especificamos que `p` deve pegar o número de páginas final, portanto, é preciso dizer que `p` é menor ou igual a `numeroPaginas`:

```
for (var p = 1; p <= numeroPaginas; p++)
```

Falta, substituir o `numeroPagina` que passamos para o `ImprimirPagina` por `p`. Teremos:

```
ImprimirPagina(TAMANHO_PAGINA, contexto, p);
```

Por fim, removemos o `int numeroPagina = 3` que estará sobrando!

Nosso código ficará da seguinte maneira:

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

    var numeroNotasFiscais= contexto.NotasFiscais.Count();
    var numeroPaginas = numeroNotasFiscais / TAMANHO_PAGINA;

    for (var p = 1; p < numeroPaginas; p++)
    {

        ImprimirPagina(TAMANHO_PAGINA, contexto, p);
    }
```

```
        Console.ReadKey();
    }
```

Ao rodar a aplicação teremos um relatório que mostra da linha `1 a 412`, que é a última que aparece no relatório. O problema é que a impressão aparece em sequência, então, não saberemos onde cada linha está situada. Para resolver a questão, vamos inserir uma última instrução, antes da seguinte linha:

```
foreach (var nf in query)
```

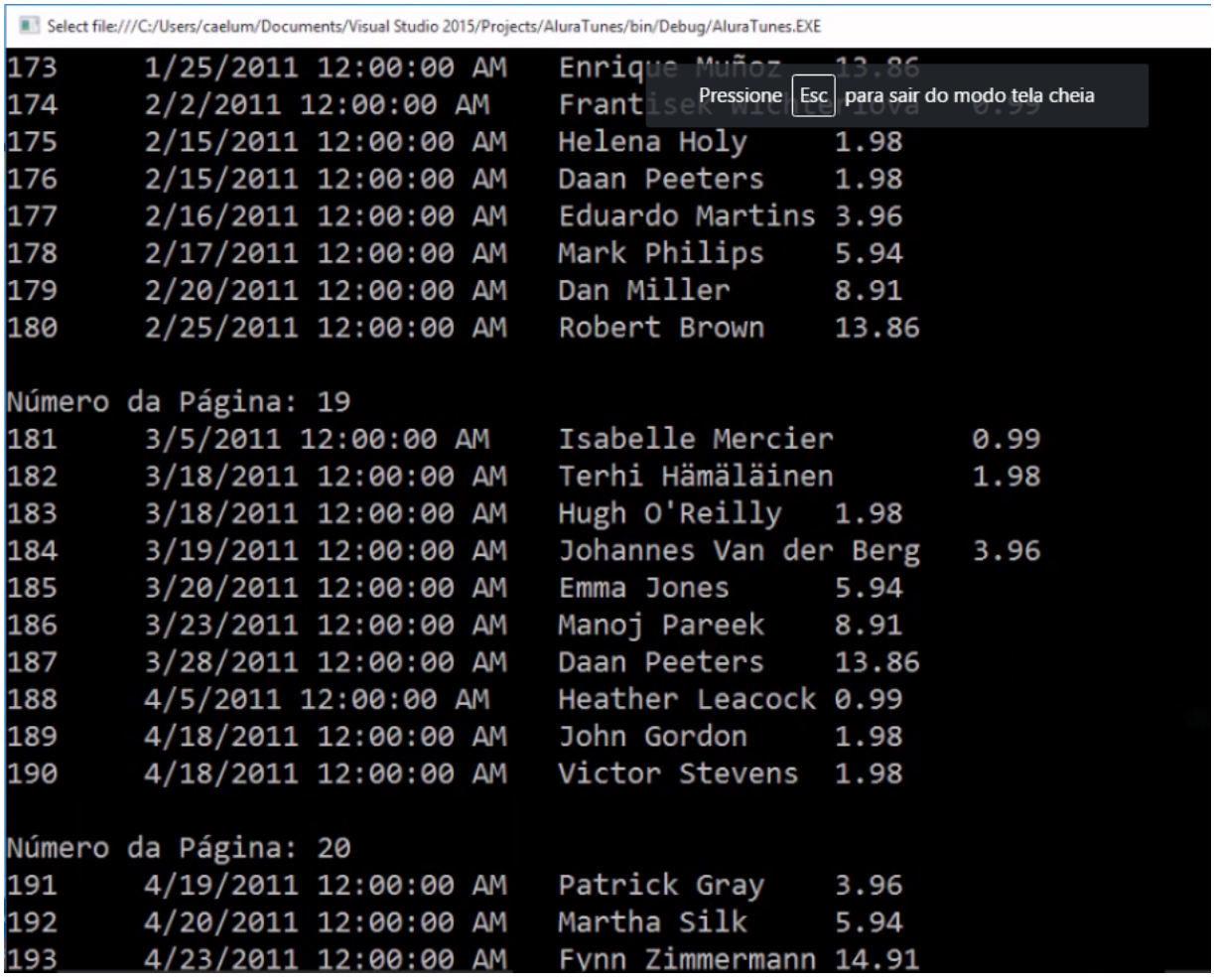
Nós colocamos o `ConsoleWriteLine()` e passamos para isso o `"Numero da Pagina, {0}", numeroPagina`. Teremos:

```
query = query.Skip(NumeroDePulos);

query = query.Take(TAMANHO_PAGINA);

Console.WriteLine();
Console.WriteLine("Numero da Pagina, {0}", numeroPagina);
```

Ao rodar a aplicação o resultado é, finalmente, uma divisão por páginas:



Acabamos de aprender a fazer a impressão de um relatório paginado. Este é um sistema extremamente utilizado! Por exemplo, na caixa de e-mails do **Gmail**, pois podemos visualizar diferentes páginas, inclusive pulando-as sem precisar passar por todas as páginas anteriores!