01

1 - Execucao tardia - Execucao imediata

Transcrição

Agora o pedido do cliente é para gerarmos um relatório mensal dos aniversariantes, separando as pessoas que fazem aniversário mensalmente. Assim, vamos preparar o contexto adicionando a seguinte linha:

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

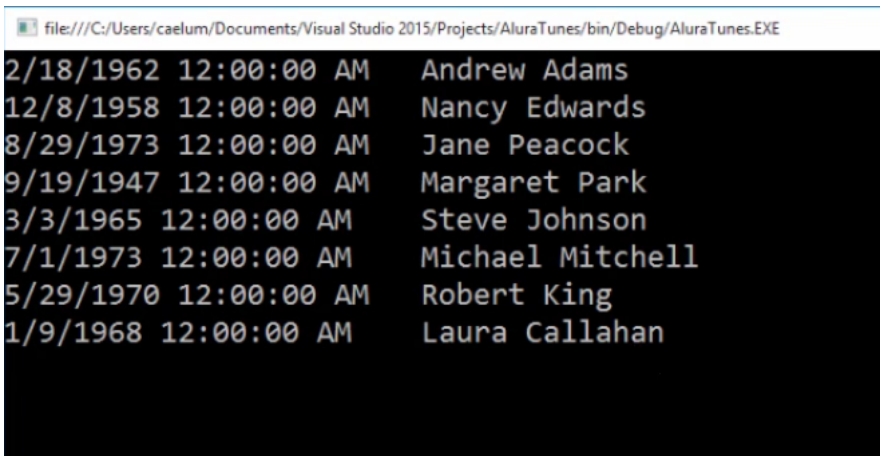
Dentro disso acrescentaremos uma variável:

```
var query = from f contexto.funcionarios
```

Ainda, vamos adicionar o `foreach(var f in query)` e também o `Console.WriteLine()` para imprimir o resultado. Para o `Console.WriteLine()` nós vamos passar as três características que buscamos: `f.DataNascimento` , `f.PrimeiroNome` e `f.Sobrenome` . Junto disso nós colocaremos a formatação que indicará a presença de três colunas, com espaço entre a segunda e a última: `"{0}\t{1}\t{2}"` . Após acrescentarmos todas estas informações, adicionaremos também o `Console.ReadKey()` . O resultado é o seguinte:

```
static void Main(string[] args)
{
    using(var contexto = new AluraTunesEntities())
    {
        var query = from f in contexto.funcionarios
                    select f;

        foreach (var f in query)
        {
            Console.WriteLine("{0}\t{1} {2}", f.DataNascimento, f.PrimeiroNome, f.Sobrenome);
        }
    }
}
```

Ao rodar o código teremos o seguinte resultado:



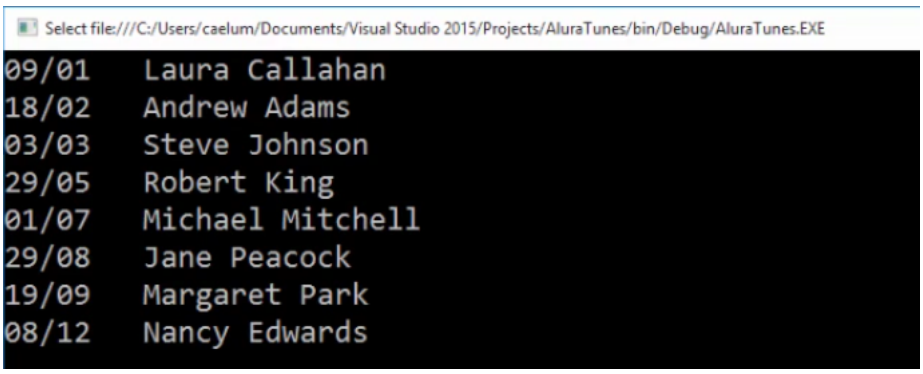
Aparecem todos os funcionários com suas respectivas datas de nascimento e nomes. O resultado já está próximo ao que desejamos, mas nosso interesse é que o ano de nascimento dos indivíduos não seja trazido junto! Portanto, é preciso modificar o código para que sejam mostrados apenas o dia e o mês. Dessa forma, vamos adicionar junto a formatação do `Console o {0:dd/MM}` .

Assim, é de se esperar que o resultado seja o nome das pessoas acompanhado de mês de nascimento e ano. Dessa maneira, o resultado é o seguinte:

Mas, a ordem que nós queremos que se estabeleça na lista é por ano de nascimentos. Assim, adicionamos abaixo do `from f in contexto.Funcionarios` uma cláusula de tipo `orderby` seguido de `f.DataNascimento` . Vamos acessar a data através da propriedade `value` e só depois acessamos o `Month` - propriedade que contêm o mês de nascimento (`f.DataNascimento.Value.Month`). Como também queremos que a ordem se estabeleça por meio dos dias de nascimento, vamos introduzir o `f.DataNascimento.Value.Day` . Teremos o seguinte código:

```
static void Main(string[] args)
{
    using (var contexto = new AluraTunesEntities())
    {
        var query = from f in contexto.Funcionarios
        orderby f.DataNascimento.Value.Month, f.DataNascimento.Value.Day
        select f;

    }
}
```

O resultado será o seguinte:



Agora, vamos separar os aniversariantes por mês.

Desta maneira, vamos introduzir a variável `mesAniversario` com valor equivalente a `1`. Assim, vamos introduzir nessa variável um `Console.WriteLine()` e passamos para ele `Mês: {0}, mesAniversario`:

```
var mesAniversario = 1;

Console.WriteLine("Mês: {0}", mesAniversario);

var query = from f in contexto.funcionarios
            orderby f.DataNascimento.Value.Month, f.DataNascimento.Value.Day
            select f;
```

Depois, vamos adicionar também um filtro para pegar apenas os funcionários que fazem aniversário em um determinado mês. Desta forma, adicionaremos `where` acompanhado da seguinte condição:

```
f.DataNascimento.Value.Month == mesAniversario
```

O trecho do código ficará assim:

```
var mesAniversario = 1;

Console.WriteLine("Mês: {0}", mesAniversario);

var query = from f in contexto.Funcionarios
            where f.DataNascimento.Value.Month == mesAniversario
            orderby f.DataNascimento.Value.Month, f.DataNascimento.Value.Day
            select f;
```

O resultado ao rodar a aplicação é o seguinte:

A `Laura Callahan` faz aniversário no `mês 1` (janeiro).

A ideia é imprimir mês a mês os demais aniversariantes, até dezembro.

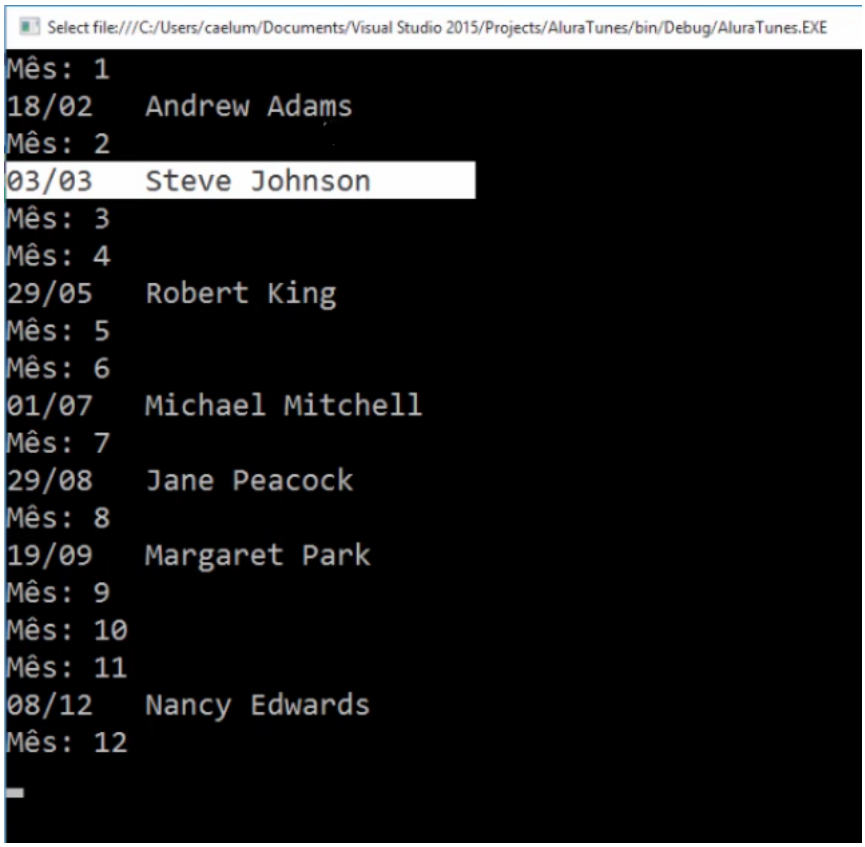
Para fazer isso vamos inserir dentro de `mesAniversario` um laço `while()` e passaremos a condição de que enquanto o mês for menor que dezembro, todos os aniversariantes dos meses anteriores deverão ser listados:

```
var mesAniversario = 1;

while (mesAniversario <= 12)
```

Feito isso, é preciso também incrementar o mês do aniversário. Dessa forma, dentro da variável `query` nós colocaremos o `mesAniversario++`:

```
var mesAniversario = 1;

while (mesAniversario <= 12)
{
    Console.WriteLine("Mês: {0}", mesAniversario);

    var query = from f in contexto.Funcionarios

            where f.DataNascimento.Value.Month == mesAniversario
            orderby f.DataNascimento.Value.Month, f.DataNascimento.Value.Day
            select f;

mesAniversario ++;
}
```

Rodando a aplicação o resultado é o seguinte relatório:



O problema é que o mês descrito não corresponde aos meses dos aniversários que o acompanham.

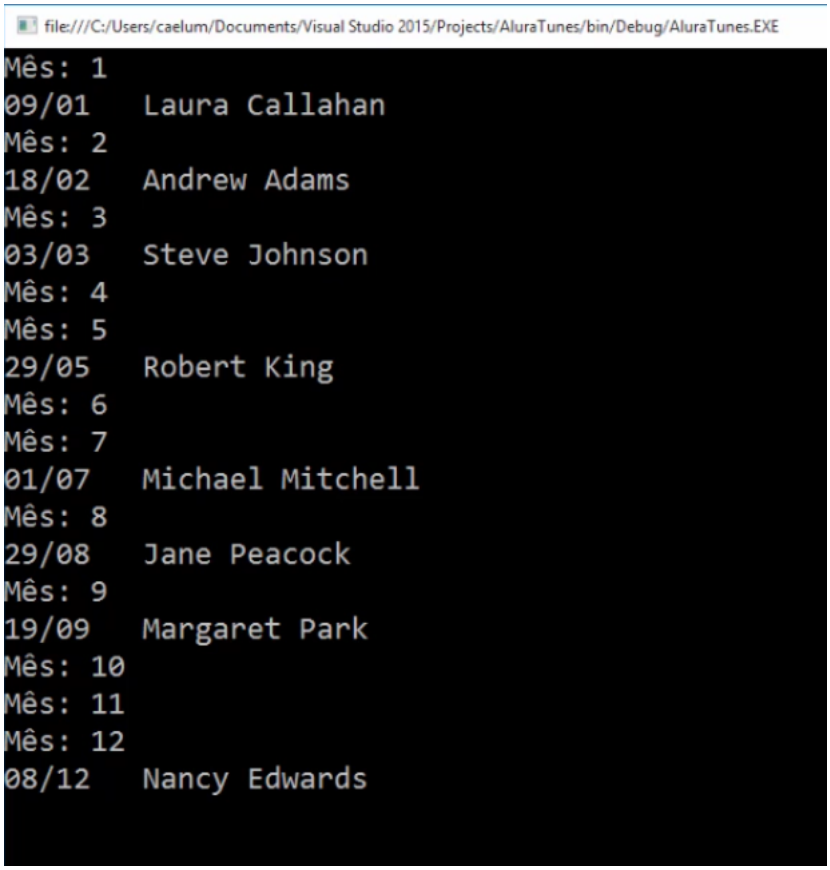
O que acontece é que definimos, primeiro, a consulta para ser parte do mês `1` e ela foi incrementada para os meses subsequentes. Por baixo dos panos, uma consulta `LINQ` é armazenada para compor uma árvores de expressões, um conjunto de comandos que busca na origem dos dados e quem faz isso é a entidade `Funcionarios`.

Temos que tomar cuidado para saber se uma consulta está sendo efetuada no servidor ou localmente, aonde quer que esteja a origem de dados!

Vamos modificar a variável `query` por uma lista que deve ser instanciada em memória, assim, chamaremos ela de `lista` e o conteúdo nós deixaremos entre parênteses. Ademais, vamos adicionar um comando que deve converter a `query` em uma lista que deve ser guardada em memória, o `To.List()`. Temos quer modificar também o `foreach()` que deve passar a receber o `f in lista`:

```
var lista = (from f in contexto.Funcionarios
             where f.DataNascimento.Value.Month == mesAniversario
             orderby f.DataNascimento.Value.Month, f.DataNascimento.Value.Day
             select f).To.List() ;

mesAniversario++;

foreach (var f in lista)
{
     Console.WriteLine("{0:dd/MM}\t{1} {2}", f.DataNascimento, f.PrimeiroNome, f.Sobrenome);
        }
}
```

O resultado rodando a aplicação é o seguinte:



Por fim, temos os aniversariantes relacionados aos meses certos!

O método `ToList()` serve para converter uma coleção de dados em uma lista, dicionário, que vão ficar em memória!

Quando o `mesAniversario++` é modificado, a definição de consulta já foi compilada e avaliada pelo método `ToList()`. Dessa maneira, o resultado da lista já estava em memória, ou seja, ao incrementarmos o `mesAniversario++`, a lista já não vai será alterada pelo mês de aniversário. Isto ocorre pois a lista passou a ser um objeto em memória e não a definição de consulta que foi usada primeiramente. Agora, mesmo se alterarmos o mês de aniversário, o resultado da lista não sofrerá mais mudanças.

Então, a consulta está sendo executada imediatamente! Isto significa que ela possui uma execução imediata pelo método `ToList()`!

A execução imediata é útil quando não é interessante consultar o servidor de banco de dados, frequentemente, para trazer uma informação que já foi exibida uma primeira vez.

Uma execução imediata como a que acabamos de fazer para trazer uma lista em memória é útil para guardar a informação em um `cache`. Pois, dessa maneira não é preciso buscar a informação do servidor, pode-se simplesmente consultar o `cache` no banco de dados. Por outro lado, uma consulta de execução tardia ou adiada - como vimos anteriormente - é útil, pois sempre que buscarmos a consulta no banco de dados, teremos uma informação atualizada sobre os dados! Isso é interessante para o caso de executarmos a consulta e termos de volta sempre informações recentes. Para isto, basta armazenar a consulta em uma `query` e ela será chamada quando for necessária - assim, teremos a garantia de que as informações estão atualizadas!