# Executando consultas com LINQ e expressões Lambda

Fonte: <http://msdn.microsoft.com/pt-br/library/jj128159.aspx>

Este artigo apresenta o uso de sintaxe LINQ e expressões Lambda para execução de consultas utilizando recursos de seleção, filtro e ordenação com modelos de dados do Entity Framework.

Um modelo de dados do Entity Framework é uma ponte entre a sua aplicação e sua base de dados. Este modelo de dados nos fornece a habilidade de interagir com a base de dados consultando, inserindo, atualizando e removendo registros. O uso do modelo de dados do Entity Framework faz com que não nos preocupemos com a estrutura de nossa base de dados, apenas devemos nos preocupar com a lógica de nossa aplicação. O Entity Framework ficará responsável por processar seus comandos e transformá-los em instruções de consulta para seu banco de dados. Depois que o banco de dados executar a consulta, o resultado será retornado para o Entity Framework e transformado em classes do seu modelo de dados. Uma das maiores vantagens do uso do Entity Framework é que uma vez o banco de dados mapeado, podemos interagir com os dados através do uso de objetos e propriedades. Quando executamos consultas com LINQ ou com expressões Lambda, não precisamos mais nos preocupar com a escrita de queries SQL, deixamos o Entity Framework ficar responsável pela construção e execução da consulta SQL.

O LINQ (Language Integrated Query), como o nome sugere, é uma linguagem tipada de consulta integrada ao .Net Framework. O termo tipada significa que a query, é definida utilizando as classes e propriedades que constituem seu modelo.

A execução de consultas utilizando expressões Lambda é feita a partir do uso de method-based queries, que nada mais são do que a construção de consultas através do acumulo de chamadas a métodos, tais métodos são chamados de standard query operators.

A linguagem de consulta LINQ e as expressões Lambda foram adicionadas as linguagens Visual Basic e C# na versão 3.0 do .Net Framework, enquanto que o Entity Framework foi incluído na versão 3.5 do .Net Framework.

Como principais vantagens do uso de consultas com LINQ:   
Estamos utilizando uma linguagem tipada para execução de nossas consultas;

Temos a vantagem de o IntelliSence nos auxiliar na codificação de nossas consultas;

Validação em tempo de compilação da estrutura das consultas;

No fim deste artigo você terá adquirido conhecimento para consultar, filtrar e ordenar dados utilizando queries LINQ e expressões Lambda.

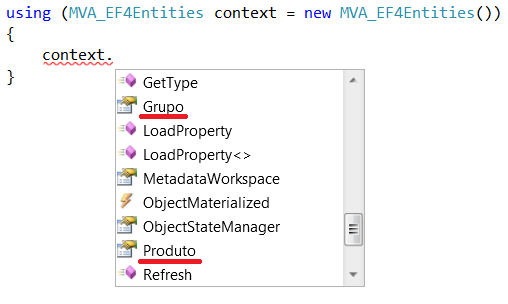
## Executando consultas

Para que as consultas sejam executadas é preciso que seja criada uma instância de um ObjectContext, este objeto funciona como elo de ligação entre a aplicação que estiver sendo desenvolvida e o seu respectivo banco de dados. Neste objeto ficam mapeadas as tabelas, views e stored procedures que fizerem parte de nosso modelo de dados do Entity Framework. Neste exemplo, a classe ObjectContext que será utilizada se chama “MVA\_EF4Entities”. A criação de sua instância pode ser feita desta forma:

C#

MVA\_EF4Entities context = new MVA\_EF4Entities();

Se olharmos as propriedades deste ObjetContext, veremos que nossas tabelas Grupo e Produto estão mapeadas e prontas para serem utilizadas.



Os exemplos a seguir demonstram como é possível executar consultas, filtros e ordenação de dados, tanto com a linguagem de consulta LINQ, quanto com expressões Lambda, baseados em modelos de dados do Entity Framework.

### Consultando todos os registros de uma tabela

Para o primeiro exemplo serão consultados todos os registros da tabela de produtos. A sintaxe é bastante simples, pois não se deve utilizar nenhum método adicional para filtragem, ordenação ou agrupamento de registros, pois o desejado é apenas que se obtenham todos os registros contidos em uma tabela da base de dados, para isso fazemos da seguinte maneira:

Exemplo com LINQ

C#

using (MVA\_EF4Entities context = new MVA\_EF4Entities())

{

var query = from c in context.Produto select c;

foreach (var item in query)

{

Console.WriteLine("{0} - {1}",item.Id,item.Nome);

}

Console.Read();

}

Exemplo com Lambda

C#

using (MVA\_EF4Entities context = new MVA\_EF4Entities())

{

var query = contect.Produto;

foreach (var item in query)

{

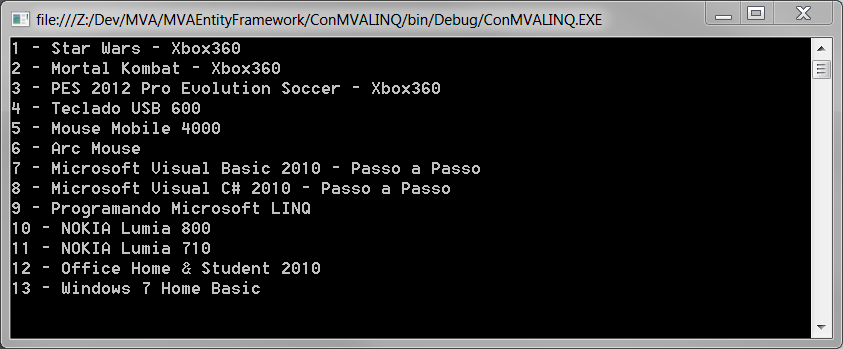
Console.WriteLine("{0} - {1}",item.Id,item.Nome);

}

Console.Read();

}

Como resultado previsto, devemos obter uma lista com todos os itens registrados na base de dados, como a figura a seguir apresenta.



Este exemplo é bastante simples e logo notamos o quão produtivo é o uso de consultas com o Entity Framework, pois com apenas duas linhas de código conseguimos acessar a base de dados e consultar todos os registros que ali estão inseridos. Além disso, conseguimos perceber a vantagem de trabalhar com resultados mapeados em classes do nosso modelo, pois conseguimos acessar o valor das colunas “Id” e “Nome” dos produtos de forma bastante simples e transparente, não exigindo nenhum conhecimento adicional de como a tabela esta construída.

### Consultando os cinco primeiros registros de uma tabela

Neste exemplo faremos uso do método Take que retorna os N primeiros registros de uma consulta e descarta o restante. Este método é semelhante ao operador TOP da linguagem SQL.

Exemplo com LINQ

C#

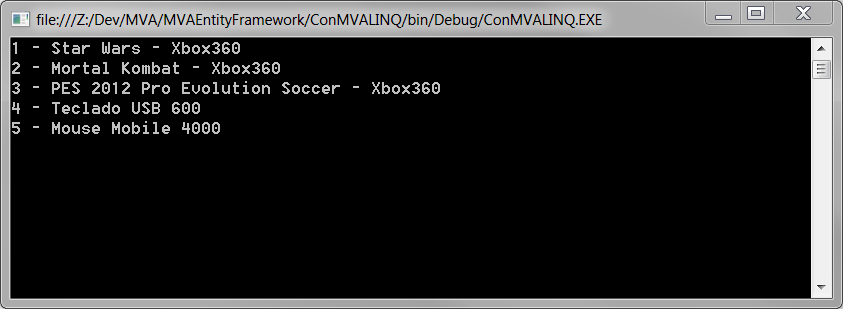
var query = (from p in contexto.Produto select p).Take(5);

Exemplo com Lambda

C#

var query = contexto.Produto.Take(5);

Como resultado esperado, devemos ter como resposta uma lista com os cinco primeiros registros da tabela de produtos.



### Obtendo a quantidade de itens retornados por uma consulta

Um método bastante útil é o Count, que retorna a quantidade de registros retornados por nossa consulta. Este método como resposta retorna um inteiro que corresponde a quantidade de registros retornados.

Exemplo com LINQ

C#

int quantidadeRegistros = (from p in context.Produto select p).Count();

Exemplo com Lambda

C#

int quantidadeRegistros = context.Produto.Count();

### Obtendo o primeiro registro de uma consulta

Caso seja preciso obter o primeiro registro retornado de uma consulta, podemos fazer uso do método First. Este método tem a função de selecionar dentre os resultados da consulta apenas o primeiro registro, descartando os demais.

Exemplo com LINQ

C#

var resultado = (from p in context.Produto select p).First();

Exemplo com Lambda

C#

var resultado = context.Produto.First();

Consultando registros através de campos texto

Consultas com LINQ e expressões Lambda permitem a utilização de filtros para execução de seleções mais aprimoradas, que a partir de condições de comparação retornam apenas os registros que estiverem de acordo com o domínio de dados desejado. Um exemplo bastante comum de consulta é a seleção de registros a partir do campo Nome. Assim, neste exemplo, faremos uma consulta na base de dados a fim de retornar todos os produtos que tiverem o nome “NOKIA Lumia 800”.

Exemplo com LINQ

C#

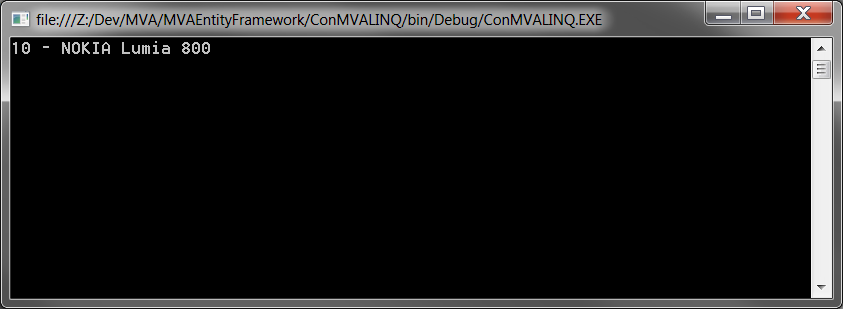
var query = from p in context.Produto where p.Nome == " NOKIA Lumia 800" select p;

Exemplo com Lambda

C#

var query = context.Produto.Where(p => p.Nome == " NOKIA Lumia 800");

O resultado esperado é a exibição de apenas um registro, como na figura abaixo:



O interessante de consultas envolvendo campos de texto, é que neste caso a comparação de valores não é case sensitive, isto é, não importa se os dados estão salvos na base de dados com letras maiúsculas ou minúsculas, o resultado será sempre o mesmo.

Mas, existem alguns cenários nos quais não queremos filtrar os registros pelo valor exato que existe no campo, e preferimos fazer a consulta utilizando condições mais abrangentes, por exemplo: consultar todos os produtos que contém a palavra “NOKIA” em seu nome. Para execução deste exemplo podemos utilizar o método Contains que verifica se um campo contém um valor esperado, conforme o código abaixo.

Exemplo com LINQ

C#

var query = from p in context.Produto where p.Nome.Contains(" NOKIA") select p;

Exemplo com Lambda

C#

var query = context.Produto.Where(p => p.Nome.Contains("NOKIA"));

### Consultando registros através de operadores relacionais

Outra opção de consulta é através do uso de operadores relacionais (i.e., <, <=, >, >=, != e ==). Estes operadores são bastante úteis para seleção de registros utilizando filtros em campos numéricos (i.e., inteiros, decimais, ponto flutuante, etc). No exemplo abaixo fazemos uma consulta a fim de descobrir quais produtos possuem valores de custo maior que R$ 100,00.

Exemplo com LINQ

C#

var query = from p in context.Produto where p.Custo > 100 select p;

foreach (var item in query)

{

Console.WriteLine("{0} – {1} – {2:C}",item.Id,item.Nome, item.Custo);

}

Console.Read();

Exemplo com Lambda

C#

using (MVA\_EF4Entities context = new MVA\_EF4Entities())

{

var query = context.Produto.Where(p => p.Custo > 100);

foreach (var item in query)

{

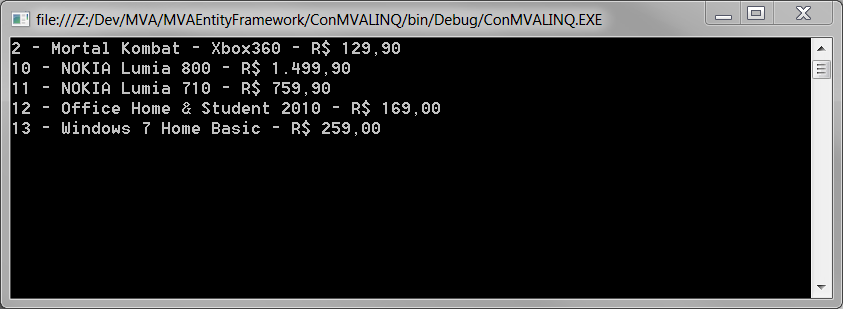
Console.WriteLine("{0} – {1} – {2:C}",item.Id,item.Nome, item.Custo);

}

Console.Read();

}

O resultado esperado é a listagem de todos os produtos com valor de custo maior que R$ 100,00 - como apresentado pela figura abaixo:



### Consultar registros utilizando operadores lógicos

Podemos encarar cenários que exijam mais de uma condição para o filtro de registros. Nestas situações é preciso combinar condições através do uso de operadores lógicos (i.e., AND e OR). Como exemplo, faremos uma consulta filtrando os registros que possuem “XBOX360” em seu nome e (AND) que cujo valor de custo seja menor que R$ 100,00.

Exemplo com LINQ

C#

var query = from p in context.Produto where p.Custo < 100 && p.Nome.Contains("xbox360") select p;

Exemplo com Lambda

C#

var query = context.Produto.Where(p => p.Custo < 100 && p.Nome.Contains("xbox360"));

Neste exemplo utilizamos o operador lógico AND (&&), também podemos fazer uso do operador lógico OR (||). Para demonstrar o uso do operador lógico OR, faremos uma consulta selecionando os produtos que possuem as palavras “XBOX” ou (OR) “NOKIA” em seu nome.

Exemplo com LINQ

C#

var query = from p in context.Produto where p.Nome.Contains("xbox") || p.Nome.Contains("nokia") select p;

Exemplo com Lambda

C#

var query = context.Produto.Where(p => p.Nome.Contains("xbox")|| p.Nome.Contains("nokia"));

### Utilizando dados de um relacionamento para filtrar registros

Outro recurso bastante poderoso do Entity Framework é permitir que se utilizem dados de uma entidade relacionada como parte de um dos critérios de filtragem de nossa consulta. Neste exemplo, faremos uma consulta filtrando todos os produtos que fazem parte do grupo “Games”.

Exemplo com LINQ

C#

var query = from p in context.Produto where p.Grupo.Nome == Games" select p;

Exemplo com Lambda

C#

var query = context.Produto.Where(p => p.Grupo.Nome == " Games");

Se olharmos com mais detalhe os códigos deste exemplo, veremos que acessamos a propriedade “Nome”, da entidade “Grupo”, associada com cada produto para efetuarmos a consulta de produtos cadastrados no banco de dados.

### Ordenando registros

Um recurso bastante importante ao apresentarmos dados para um usuário é a sua ordem. Constantemente encaramos requisitos que definem a ordem na qual os dados devem ser apresentados. Neste exemplo será demonstrado como utilizar os recursos de ordenação do Entity Framework.

Inicialmente, começaremos com o exemplo mais simples, no qual iremos consultar todos os produtos ordenando-os por nome.

Exemplo com LINQ

C#

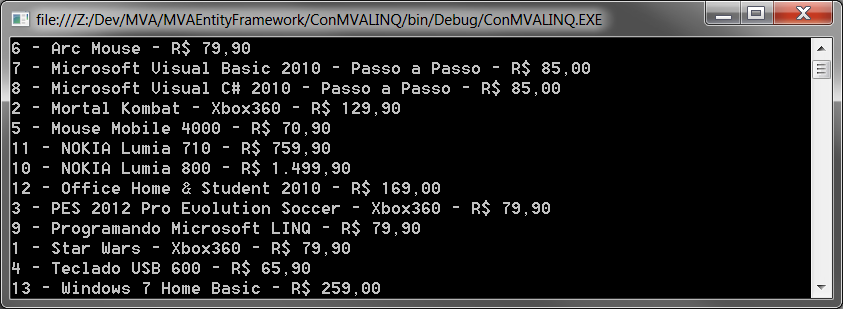
var query = from p in context.Produto orderby p.Nome select p;

Exemplo com Lambda

C#

var query = context.Produto.OrderBy(p => p.Nome);

O resultado esperado para esta consulta é a lista de produtos ordenada pelo nome, conforme a figura abaixo:



Outro requisito comum é a ordenação decrescente (do maior para o menor), e o Entity Framework também provê recursos para essa necessidade. Mas, para este exemplo, utilizaremos a propriedade “Custo” como chave de ordenação de nossa consulta.

Exemplo com LINQ

C#

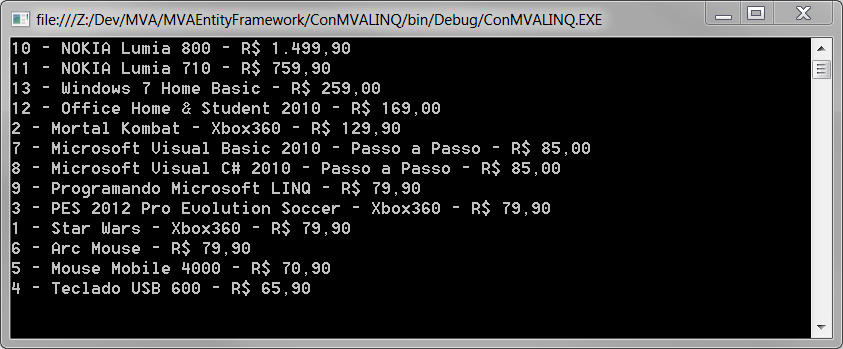
var query = from p in context.Produto orderby p.Custo descending select p;

Exemplo com Lambda

C#

var query = context.Produto.OrderByDescending(p => p.Custo);

O resultado esperado desta consulta é a listagem de produtos ordenada pelo valor de custo, de forma decrescente.



Outro recurso muitas vezes necessário é a combinação de diferentes operadores de ordenação, isto é, a utilização de mais de um campo para ordenação de registros. Para demonstrar este cenário, faremos a ordenação de produtos por ordem decrescente de valor de custo e então ordenada de forma secundária pelo nome dos produtos, em ordem alfabética.

Exemplo com LINQ

C#

var query = from p in context.Produto orderby p.Custo descending p.Nome ascending select p;

foreach (var item in query)

{

Console.WriteLine("{0} – {1} – {2:C}",item.Id,item.Nome, item.Custo);

}

Console.Read();

Exemplo com Lambda

C#

var query = context.Produto.OrderByDescending(p =>p.Custo).ThenByDescending(person => person.Age);

foreach (var item in query)

{

Console.WriteLine("{0} – {1} – {2:C}",item.Id,item.Nome, item.Custo);

}

Console.Read();

O resultado esperado é apresentado na imagem abaixo.

