LINQ

Guinther Pauli

MCP, MCAD, MCSD, MCTS, MCPD

LINQ

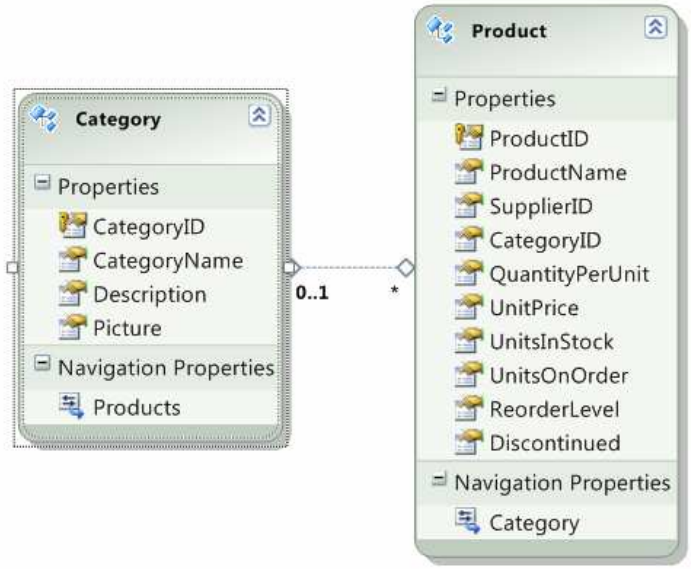
* Language Integrated Query (LINQ);
* Permite usar uma mesma linguagem, com semântica semelhante, para múltiplos propósitos;
* Compilada, não resulta em erros de esquema;
* LINQ to Objects (ao invés de for eachs e ifs)
* LINQ to XML (ao invés de XPath e XQuery);
* LINQ to Entities (para consultas a modelos de entidade);
* LINQ to SQL (ao invés de SQL);
* LINQ to ... (há dezenas de providers disponíveis).

LINQ

```
// array de inteiros
var nums = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
// retorna só os pares
var pares = from n in nums
            where (n % 2) == 0
            select n;
```

LINQ

```
// retorna todos os produtos que são bebidas começando com "P"
var ctx = new NorthwindEntities();
var query = from p in ctx.Products
            where p.Category.CategoryName == "Beverages"
                  & p.ProductName.StartsWith("P")
            select p;
```



LINQ

* O objetivo da Microsoft com a criação da Language-Integrated Query (LINQ) foi aproximar o mundo dos objetos do mundo dos dados. O LINQ corresponde a uma sintaxe unificada, inicialmente incorporada às linguagens C# e Visual Basic, para consultas em fontes de dados variadas.

LINQ

* A sintaxe de consulta da LINQ foi inspirada na da Structured Query Language (SQL), que é uma linguagem padrão para comunicação com bancos de dados relacionais. Assim como na linguagem SQL, as expressões de consulta LINQ permitem a construção de instruções variadas para extração de informações.

LINQ

* Tradicionalmente, as consultas a bancos de dados em aplicações eram expressas por strings sem verificação de sintaxe nem em tempo de projeto e nem em tempo de compilação, além de não haver suporte a IntelliSense.
* Erros de sintaxe na consulta somente eram identificados em tempo de execução.
* Além disto, o desenvolvedor precisava aprender linguagens de consulta específicas para cada tipo de fonte de dados, como: bancos de dados relacionais, coleções de objetos na memória, documentos XML, dentre outros tipos de fontes de dados

LINQ

* A ideia com a LINQ foi tornar as consultas como um recurso de primeira classe nas linguagens de programação da plataforma .NET, sendo incorporado inicialmente nas linguagens C# e Visual Basic.

LINQ

* A LINQ foi introduzida nas linguagens Visual Basic 9.0 (ou Visual Basic 2008) e C# 3.0 (Visual C# 2008), em novembro de 2007, com o .NET Framework 3.5 e o Visual Studio 2008. Diversos novos recursos foram introduzidos na versão 3.0 da linguagem de programação C#, sendo que grande parte deles voltados para suportar a inclusão da LINQ na linguagem.

LINQ

* Vemos a seguir uma lista dos recursos que estão diretamente ligados com o suporte à LINQ:
  * Iniciadores de objetos
  * Tipos implícitos em variáveis locais (var)
  * Métodos de extensão
  * Expressões lambda
  * Árvores de expressão
  * Tipos anônimos
  * Expressões de consulta

LINQ Providers

* O padrão LINQ torna simples a consulta em fontes de dados para as quais o LINQ esteja habilitado, uma vez que a sintaxe e o padrão de consultas não muda.
* Um LINQ Provider para uma determinada fonte de dados permite não somente a operação de consulta, mas também operações de inclusão, atualização e exclusão, além de mapeamento para tipos definidos pelo usuário.
* O .NET Framework suporta os seguintes LINQ Providers:

LINQ Providers

* **LINQ to Objects** - Componente, introduzido no .NET Framework 3.5, que permite consultar coleções de objetos do tipo IEnumerable<T> diretamente.
* **LINQ to XML** - Componente, introduzido no .NET Framework 3.5, que fornece uma interface de manipulação de XML em memória. Corresponde a uma interface de programação muito mais avançada e simples de manipulação XML na memória que a interface do padrão W3C Document Object Model (DOM).

LINQ Providers

- **LINQ to SQL** - Componente, introduzido no .NET Framework 3.5, que fornece infraestrutura para gerenciar dados relacionais como objetos. Este componente permite fazer mapeamento objeto-relacional (ORM – Object-Relational Mapping), ou seja, permite mapear um modelo de dados de um banco de dados relacional para um modelo de objetos. Atualmente, a LINQ to SQL somente contém um provider para o servidor de banco de dados SQL Server, desenvolvido pela Microsoft. Como o LINQ to SQL não fornece uma boa abstração do banco de dados, não houve interesse no desenvolvimento de providers para outros bancos de dados.

LINQ Providers

- **LINQ to Entities**, que é parte do ADO.NET Entity Framework O ADO.NET Entity Framework é um componente do .NET Framework, introduzido no .NET Framework 3.5 SP1, que, assim como a LINQ to SQL, fornece infraestrutura para gerenciar dados relacionais como objetos. Porém, ao contrário da LINQ to SQL, permite trabalhar com dados num alto nível de abstração do engine de armazenamento de dados e do esquema relacional.

LINQ Providers

- O Entity Framework suporta o Entity Data Model (EDM) para definição dos dados num nível conceitual. Há uma separação clara entre as informações do modelo conceitual, do modelo de armazenamento e do mapeamento entre estes dois modelos. O LINQ to Entities é um recurso do Entity Framework que permite escrever consultas ao modelo conceitual usando as linguagens C# e Visual Basic. Atualmente, o ADO .NET Entity Framework está na versão 6.1 e corresponde à principal tecnologia de mapeamento objeto-relacional suportada pela Microsoft.

Standard Query Operators

* Os Standard Query Operators são métodos de extensão que formam o padrão LINQ. Eles fornecem capacidades de consulta incluindo projeção, filtragem, ordenação, agregação e outras.
* Existem dois conjuntos de Standard Query Operators LINQ: os que atuam em objetos do tipo System.Collections.Generic.IEnumerable<T> e os que atuam em objetos do tipo System.Linq.IQueryable<T>. Os métodos de extensão que constituem cada conjunto são definidos nas classes estáticas System.Linq.Enumerable e System.Linq.Queryable, respectivamente.

Standard Query Operators

* Alguns dos Standard Query Operators usados mais frequentemente possuem palavras-chaves na sintaxe da linguagem C# para permitir que eles sejam chamados como parte de uma expressão de consulta.
* Principalmente na linguagem C#, diversos métodos de extensão importantes dos Standard Query Operators não possuem palavras-chaves equivalentes nas expressões de consulta LINQ. Deste modo, é muito importante o domínio destes métodos de extensão para ultrapassar as limitações das expressões de consulta.

Expressões de consulta LINQ

* Para um desenvolvedor que escreve consultas LINQ, a parte mais visível da integração com a linguagem C# fornecida pela LINQ são as expressões de consulta. Expressões de consulta são escritas com uma sintaxe declarativa que foi introduzida na versão 3.0 da linguagem C#. A sintaxe de consulta permite realizar, com um mínimo de código, diversas operações em dados, como:
  * filtragem;
  * projeção;
  * ordenação;
  * agrupamento;
  * junção e outras.

Expressões de consulta LINQ

* As mesmas expressões de consulta LINQ são usadas para consultar coleções de objetos na memória (LINQ to Objects), dados em bancos de dados relacionais (LINQ to SQL e LINQ to Entities), objetos DataSet na memória (LINQ to DataSet), documentos XML na memória (LINQ to XML) ou, ainda, qualquer outra fonte de dados que tenha um LINQ Provider disponível.

Sintaxe de Expressão de consulta LINQ

```
      from id in fonteDados
    { from id in fonteDados |
      join id in fonteDados on expressão equals expressão
[ into id ] |
      let id = expressão |
      where condição |
      orderby id1, id2, … [ascending | descending] }
      select expressão | group expressão by chave
    [ into id ]
```

Expressões de consulta LINQ

* As expressões de consulta devem iniciar com from, depois podem ter um ou mais from, join, let, where ou orderby e deve terminar com select ou group by.
* Podemos conferir na tabela a seguir as descrições das funções das palavras-chaves, ou cláusulas, para expressões de consulta LINQ introduzidas na versão 3.0 da linguagem de programação C#.

Expressões de consulta LINQ

Cláusula	Descrição
from	Especifica a fonte de dados e uma variável de série (similar a uma variável de iteração num laço).
where	Filtra elementos da fonte de dados baseada em uma ou mais expressões booleanas.
select	Faz projeções, permitindo especificar o tipo e o formato dos elementos do resultado da consulta. Frequentemente usada em conjunto com o recurso de tipos anônimos.
join	Junta duas fontes de dados baseado em comparações de igualdade entre dois elementos de comparação especificados.
in	Palavra-chave contextual usada numa cláusula **join**.
on	Palavra-chave contextual usada numa cláusula **join**.
equals	Palavra-chave contextual usada numa cláusula **join**. Observe que se deve usar a palavra-chave **equals** ao invés do operador == na comparação da cláusula **join**.

Expressões de consulta LINQ

Cláusula	Descrição
group	Agrupa os resultados de uma consulta de acordo com valores específicos de uma chave.
by	Palavra-chave contextual usada numa cláusula **group**.
into	Fornece um identificador para servir de referência para os resultados de uma cláusula de junção (**join**), agrupamento (**group**) ou projeção (**select**).
orderby	Classifica os resultados em ordem ascendente ou descendente.
ascending	Palavra-chave contextual usada numa cláusula **orderby** para determinar a classificação em ordem ascendente, que é a classificação padrão em caso de omissão.
descending	Palavra-chave contextual usada numa cláusula **orderby** para determinar a classificação em ordem descendente.
let	Introduz uma variável para armazenar resultados de expressões intermediárias numa expressão de consulta. Deste modo, o resultado armazenado na variável pode ser reutilizado na consulta.

Cláusulas from e select

* A cláusula from é usada para especificar a fonte de dados e uma variável de série usada para referenciar cada elemento da fonte de dados.
* A cláusula select é usada para fazer projeções, permitindo especificar o tipo e o formato dos elementos do resultado da consulta.

Cláusulas from e select

* O exemplo a seguir consulta dos nomes completos dos arquivos na pasta C:\Windows\System32.

```
from arquivo in Directory.GetFiles(@"C:\Windows\System32")
select arquivo;
```

* Observe, no resultado da consulta LINQ anterior, que são apresentados os nomes completos dos arquivos, incluindo o caminho. Por exemplo: "C:\Windows\System32\net.exe".

Cláusulas from e select

* Para apresentar somente o nome do arquivo, sem o caminho, e mais a informação da extensão, pode-se fazer uma projeção. O recurso dos tipos anônimos, acrescentado na versão 3.0 da linguagem C#, permite fazer uma projeção do resultado de uma consulta LINQ sem a necessidade de se criar um novo tipo manualmente.

```
from arquivo in Directory.GetFiles(@"C:\Windows\System32")
select new
{
  NomeArquivo = Path.GetFileName(arquivo),
  Extensao = Path.GetExtension(arquivo)
};
```

Cláusulas let, orderby, ascending e descending

* A cláusula let permite introduzir uma variável para armazenar resultados de expressões intermediárias numa expressão de consulta. Deste modo, o resultado armazenado na variável pode ser reutilizado na consulta.
* A cláusula orderby classifica os resultados em ordem ascendente, definido pelo uso opcional da cláusula ascending, ou descendente, definido pelo uso obrigatório da cláusula descending.

Cláusulas let, orderby, ascending e descending

* A cláusula orderby pode ser usada para classificar o resultado da consulta a seguir em ordem crescente de extensão (opcional o uso da cláusula ascending) e decrescente de nome de arquivo (obrigatório o uso da cláusula descending). A cláusula let pode ser usada para evitar a repetição das operações de extração do nome de arquivo e da extensão dos nomes completos dos arquivos.

Cláusulas let, orderby, ascending e descending

```
from arquivo in Directory.GetFiles(@"C:\Windows\System32")
let nomeArquivo = Path.GetFileName(arquivo)
let extensao = Path.GetExtension(arquivo).ToUpper()
orderby extensao, nomeArquivo descending
select new
{
  NomeArquivo = nomeArquivo,
  Extensao = extensao
};
```

Cláusulas let, orderby, ascending e descending

* Observe que foram criadas duas variáveis (nomeArquivo e extensao) porque os resultados das expressões foram usados na cláusula orderby para ordenação e na cláusula select para projeção. Observe também o uso do método ToUpper(), da classe System.String, para converter a extensão para letras maiúsculas, uma vez que há distinção entre letras maiúsculas e minúsculas na ordenação de strings.

Cláusula where

* A cláusula where permite filtrar elementos da fonte de dados baseada em uma ou mais expressões booleanas separadas pelos operadores lógicos && (e) ou || (ou).
* Por exemplo, vamos supor que desejamos obter uma relação com todos os arquivos executáveis da pasta C:\Windows\System32 com tamanho superior a 1 MB, classificados em ordem crescente do tamanho do arquivo.

Cláusula where

```
from arquivo in Directory.GetFiles(@"C:\Windows\System32")
let infoArquivo = new FileInfo(arquivo)
let tamanhoArquivoMB = infoArquivo.Length / 1024M / 1024M
where tamanhoArquivoMB > 1M &&
  infoArquivo.Extension.ToUpper() == ".EXE"
orderby tamanhoArquivoMB
select new
{
  Nome = infoArquivo.Name,
  TamanhoMB = tamanhoArquivoMB
};
```

Cláusulas group, by e into

* A cláusula group agrupa os resultados de uma consulta de acordo com valores específicos de chaves. Ela é usada em conjunto com as palavras-chave by e into.
* A cláusula group retorna uma sequência de objetos do tipo System.Linq.IGrouping<TKey, TElement>, que contém zero ou mais itens que correspondem ao valor da chave para o grupo.

Cláusulas group, by e into

* O LINQ fornece uma série de Standard Query Operators para executar operações de agregação em dados agrupados, assim como ocorrem com o SQL. A tabela a seguir apresenta um resumo das operações de agregação disponíveis, expostas por meio de métodos de extensão definidos nas classes estáticas System.Linq.Enumerable e System.Linq.Queryable.

Cláusulas group, by e into

Método de extensão	Descrição
Aggregate	Realiza uma operação de agregação personalizada nos valores de uma coleção.
Average	Calcula a média aritmética simples dos valores de uma coleção.
Count	Conta os elementos de uma coleção. Opcionalmente, pode ser usada para contar somente os elementos que satisfazem uma determinada condição (predicado).
LongCount	Conta os elementos de uma grande coleção, que ultrapassa o limite de armazenamento do tipo int. Opcionalmente, pode ser usada para contar somente os elementos que satisfazem uma determinada condição (predicado).
Max	Determina o valor máximo em uma coleção.
Min	Determina o valor mínimo em uma coleção.
Sum	Calcula a soma dos valores em uma coleção.

Cláusulas group, by e into

* Vamos supor que desejamos obter uma relação com informações agregadas por extensão dos arquivos na pasta C:\Windows\System32.

Cláusulas group, by e into

```
from arquivo in Directory.GetFiles(@"C:\Windows\System32")
let infoArquivo = new FileInfo(arquivo)
group infoArquivo by infoArquivo.Extension.ToUpper() into g
let extensao = g.Key
orderby extensao
select new
{
  Extensao = extensao,
  NumeroArquivos = g.Count(),
  TamanhoTotalArquivosKB = g.Sum(fi => fi.Length) / 1024M,
  TamanhoMedioArquivosKB = g.Average(fi => fi.Length) / 1024D,
  TamanhoMenorArquivoKB = g.Min(fi => fi.Length) / 1024M,
  TamanhoMaiorArquivoKB = g.Max(fi => fi.Length) / 1024M
};
```

Cláusulas group, by e into

* Observe a linha com a cláusula group na consulta. Entre a cláusula group e a palavra-chave by deve-se colocar os objetos que serão agrupados, que neste caso são objetos do tipo System.IO.FileInfo. Entre as palavras-chaves by e into deve-se colocar a chave de agrupamento, que neste caso são as extensões dos arquivos em letras maiúsculas. Finalmente, após a palavra-chave into deve-se colocar um identificador para receber os dados agrupados, que neste caso foi definido como “g”. Observe que as operações de agregação na cláusula select foram feitas por meio de métodos de extensão aplicados nos agrupamentos representados pela variável “g”.

Cláusulas group, by e into

* Agora, vamos supor que exista a necessidade de se gerar um relatório bem mais complexo com informações agregadas de todos os arquivos abaixo de uma dada pasta. Por exemplo, a partir de uma pasta base é necessário gerar um relatório com informações de todos os arquivos na própria pasta base e em todos os níveis de subpastas. Este relatório deve estar agrupado por pasta e por extensão, portanto trata-se de agrupamento com chave composta, ao invés de um agrupamento com chave simples.

Cláusulas group, by e into

* Para cada conjunto de pasta e de extensão, se devem fornecer as informações da quantidade e do tamanho total dos arquivos, em KB, em cada pasta e de cada extensão. Os dados devem estar classificados em ordem crescente de nome das pastas e em ordem decrescente do tamanho total dos arquivos.
* Observe que para se realizar um agrupamento com uma chave composta pode-se definir um tipo anônimo após a palavra-chave “by” com propriedades definidas para cada integrante da chave.

Cláusulas group, by e into

```
from arquivo in obterTodosArquivosDotNet4()
let infoArquivo = new FileInfo(arquivo)
group infoArquivo
  by new
  {
    Pasta = infoArquivo.DirectoryName,
    Extensao = infoArquivo.Extension.ToUpper()
  }
  into infoArquivosPorPastaExtensao
  let tamanhoKB =
  infoArquivosPorPastaExtensao.Sum(ia => ia.Length) / 1024M
  orderby infoArquivosPorPastaExtensao.Key.Pasta,
    tamanhoKB descending
  select new
  {
    infoArquivosPorPastaExtensao.Key.Pasta,
    infoArquivosPorPastaExtensao.Key.Extensao,
    NumeroArquivos = infoArquivosPorPastaExtensao.Count(),
    TamanhoKB = tamanhoKB
  };
```

Cláusulas join, in, on e equals

* A cláusula join junta duas fontes de dados baseada em comparações de igualdade entre dois critérios de comparação especificados. A comparação de igualdade é feita com a palavra-chave equals, ao invés do operador de igualdade (==).
* Para fechar com chave de ouro, vamos acrescentar uma complexidade a mais na consulta anterior. Deve-se acrescentar mais uma informação na projeção final da consulta: a porcentagem do tamanho ocupado pelos arquivos de uma dada extensão em uma dada pasta em relação ao tamanho total de todos os arquivos nesta pasta, independente da extensão.

Cláusulas join, in, on e equals

```
string[] arquivosDotNet4 = obterTodosArquivosDotNet4();
var tamanhosTotaisPorPasta =
  from arquivo in arquivosDotNet4
  let infoArquivo = new FileInfo(arquivo)
  group infoArquivo by infoArquivo.DirectoryName into g
  select new
  {
    Pasta = g.Key,
    TamanhoTotalKB = g.Sum(ia => ia.Length) / 1024M
  };
```

Cláusulas join, in, on e equals

```
from arquivo in arquivosDotNet4
let infoArquivo = new FileInfo(arquivo)
group infoArquivo
  by new
  {
    Pasta = infoArquivo.DirectoryName,
    Extensao = infoArquivo.Extension.ToUpper()
  }
  into infoArquivosPorPastaExtensao
  join tamanhoTotalPorPasta in tamanhosTotaisPorPasta
    on infoArquivosPorPastaExtensao.Key.Pasta equals tamanhoTotalPorPasta.Pasta
    into juncaoComTamanhoTotalKB
  let tamanhoTotalKB = juncaoComTamanhoTotalKB.Single().TamanhoTotalKB
  let tamanhoKB = infoArquivosPorPastaExtensao.Sum(ia => ia.Length) / 1024M
  orderby infoArquivosPorPastaExtensao.Key.Pasta,
    tamanhoKB descending
  select new
  {
    infoArquivosPorPastaExtensao.Key.Pasta,
    infoArquivosPorPastaExtensao.Key.Extensao,
    NumeroArquivos = infoArquivosPorPastaExtensao.Count(),
    TamanhoKB = tamanhoKB,
    Porcentagem = 100 * (tamanhoKB / tamanhoTotalKB)
  };
```

Cláusulas join, in, on e equals

* No exemplo entre a cláusula “join” e a palavra-chave “in” há um identificador (tamanhoTotalPorPasta) que representa cada elemento da fonte de dados (tamanhosTotaisPorPasta). Em seguida, a palavra-chave “on” identifica a condição de junção, com uso obrigatório da palavra-chave equals para realizar a comparação de igualdade.

Cláusulas join, in, on e equals

* Observe que a junção é realizada pela comparação das pastas em ambas as coleções. Finalmente, o resultado da junção é armazenado na variável cujo identificador é definido após a palavra-chave into (juncaoComTamanhoTotalKB). Neste caso, a junção sempre retorna um único elemento correspondente da fonte de dados tamanhosTotaisPorPasta. Porém, o resultado sempre é armazenado numa coleção, uma vez que a comparação poderia trazer vários elementos. Sendo assim, foi usado o Standard Query Operator Single() para extrair o elemento único e acessar a sua propriedade com o tamanho total em KB (tamanhoTotalKB).

Conclusão

* O acréscimo do recurso LINQ à linguagem C# permite ao desenvolvedor fazer consultas complexas de uma forma bem mais simples, permitindo que ele foque no resultado desejado sem se preocupar em como os dados estão sendo extraídos nos bastidores. O recurso permite usar um raciocínio similar ao fornecido pela linguagem SQL, em bancos de dados relacionais, para fazer consultas em fontes de dados variadas diretamente na linguagem C#.

Contato

 http://www.facebook.com/guintherpauli

 http://www.twitter.com/guintherpauli

 http://www.gpauli.com

 guintherpauli@gmail.com

 http://guintherpauli.blogspot.com

 guinther.pauli

 guintherpauli@gmail.com