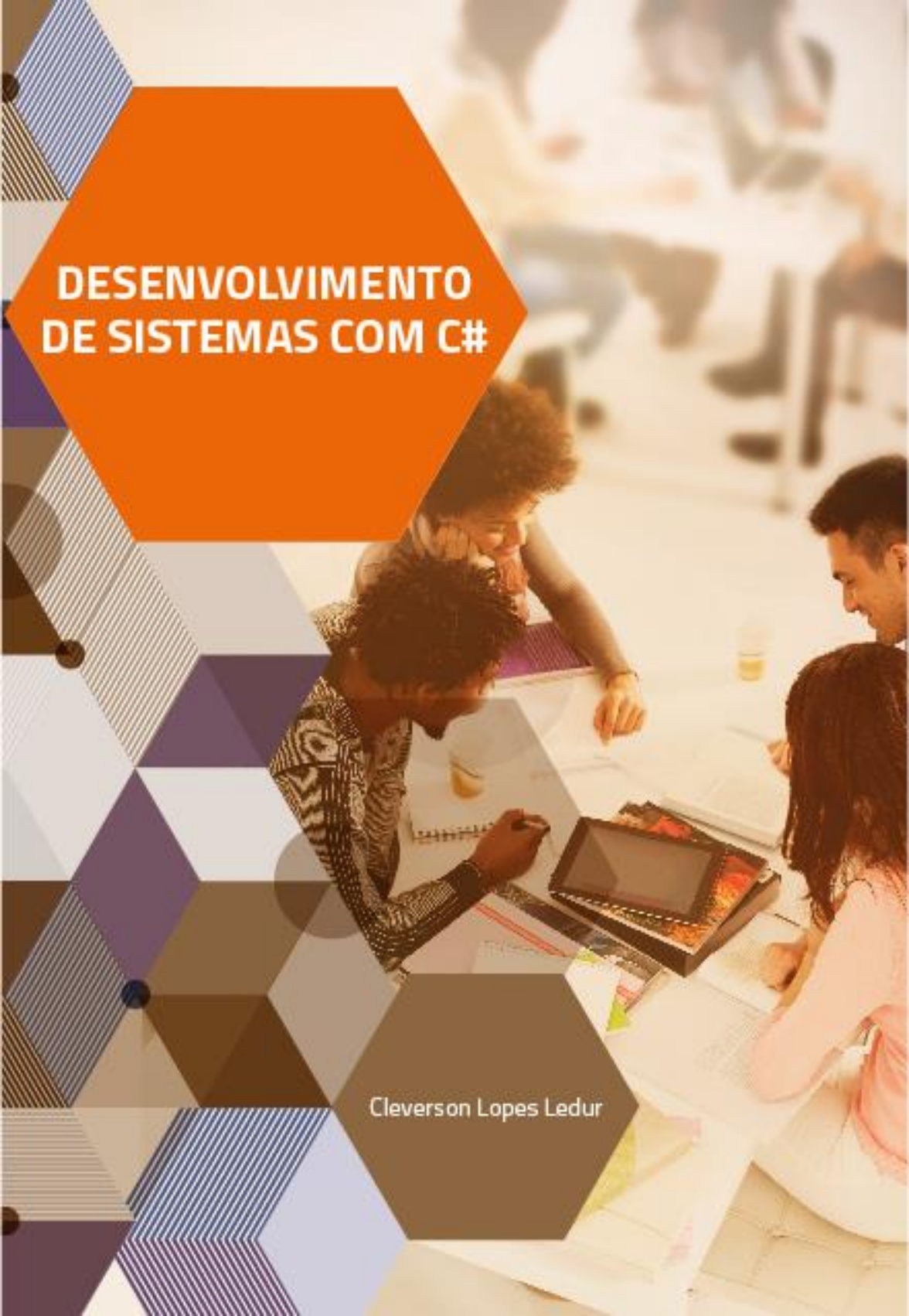
DESENVOLVIMENTO DE SISTEMAS COM C#



Cleverson Lopes Ledur

Revisão técnica:

Jeferson Faleiro Leon
Desenvolvimento de Sistemas
Especialista Formação Pedagógica de Professores
Professor do curso Técnico em informática

L475d Ledur, Cleverson Lopes.
Desenvolvimento de sistemas com #C [recurso eletrônico] / Cleverson Lopes Ledur; [revisão técnica: Jeferson Faleiro Leon]. – Porto Alegre : SAGAH, 2018.

ISBN 978-85-9502-314-7

1. Ciência da computação. 2. Linguagens de programação de computador. I. Título.

CDU 004.43

Catalogação na publicação: Karin Lorien Menoncin CRB-10/2147

Utilizar o Entity Framework para persistência em banco de dados – IV

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Identificar o conceito de expressões lambda para a busca de registros.
- Recuperar dados do banco de dados.
- Buscar registros com queries avançadas utilizando LINQ.

Introdução

A Microsoft criou o ORM chamado Entity Framework para que o desenvolvimento de softwares utilizando banco de dados fosse mais fácil. A empresa também desejava que não houvesse a necessidade de empreender grandes esforços na criação de bancos de dados. Entre os recursos disponíveis na plataforma .NET, um deles se destaca pela facilidade de uso e de criação de consultas no banco, o LINQ (*Language Integrated Query*). Ele é um dos recursos da plataforma que permitem a criação de **queries** para consultar o banco de dados. Em conjunto, as expressões lambda permitem a criação de lógicas que proporcionam mais possibilidades de consultas com poucas linhas de código. O entendimento dessas duas tecnologias é muito importante para quem, como você, deseja desenvolver sistemas na plataforma .NET.

Neste texto, você irá estudar e praticar as consultas no banco de dados utilizando as expressões lambda e o LINQ.

Expressões lambda

Uma expressão lambda é uma função anônima que você pode usar para criar deletages ou outros tipos de expressão. Ao usar expressões lambda, você

consegue escrever funções locais que podem ser passadas como argumentos ou retornadas como o valor das chamadas de função. As expressões lambda são particularmente úteis para escrever expressões de consultas no Entity Framework (MICROSOFT, 2017).

Para criar uma expressão lambda, você especifica os parâmetros de entrada (se houver) no lado esquerdo do operador lambda => e coloca o bloco de expressão ou declaração no outro lado.

Saiba mais

O token **=>** é chamado de **operador lambda**. É usado em expressões lambda para separar as variáveis de entrada no lado esquerdo do corpo lambda no lado direito. As expressões lambda são expressões em linha semelhantes aos métodos anônimos, mas são mais flexíveis. Elas são usadas extensivamente em consultas LINQ que são expressas na sintaxe do método (MICROSOFT, 2017).

No exemplo a seguir, você pode ver duas maneiras de encontrar e exibir o comprimento da sequência mais curta em uma matriz de strings. A primeira parte do exemplo aplica uma expressão lambda (`w => w.Length`) a cada elemento da matriz de palavras e, em seguida, usa o método `Min` para encontrar o menor tamanho. Para comparação, a segunda parte do exemplo mostra uma solução mais longa, que usa sintaxe de consulta para fazer a mesma coisa.

```
string[] palavras = { "cherry", "apple", "blueberry" };

//Versão mais curta com o uso de lambda.
int menoresPalavras= palavras .Min(w => w.Length);
Console.WriteLine(menoresPalavras);

//Versão mais longa sem o uso de lambda.
var query = from w in words
            select w.Length;

int menoresPalavras2 = query.Min();
Console.WriteLine(menoresPalavras2 );
```

Outro exemplo do uso de expressões lambda é para a criação de delegates. Por exemplo, a expressão lambda `x => x * x` especifica um parâmetro chamado × e retorna o valor de × quadrado. Você pode atribuir essa expressão a um tipo de delegate, conforme mostrado a seguir.

```
delegate int del(int i);
static void Main(string[] args)
{
    del myDelegate = x => x * x;
    int j = myDelegate(5); //j = 25
}
```

No exemplo anterior, foi criado um delegate, que pode ser entendido como uma função na qual você passa um valor e ele realiza determinada operação. Nesse caso, as expressões lambda são utilizadas para criar uma execução/cálculo para determinado elemento. Veja que, sem o uso do delegate e da expressão, seria preciso criar toda uma estrutura de método para realizar a mesma operação. Também é possível usar expressões lambda para criar buscas no banco de dados. Você pode ver no exemplo a seguir um método que realiza uma busca no banco de dados de acordo com a idade do usuário.

```
public List<Usuario> Busca(int idade)
{
      return context.Usuarios.Where(p => p.Idade == idade).ToList();
}
```

Você não usa expressões lambda diretamente na sintaxe de consulta, mas você as usa em chamadas de método. As expressões de consulta, por sua vez, podem conter chamadas de método. Na verdade, algumas operações de consulta só podem ser expressas na sintaxe do método.

A seguir, você pode ver outro exemplo que não utiliza dados de um banco de dados, mas sim uma lista simples de valores (scores). Por meio do método `Where`, é passada uma expressão lambda que é utilizada para contar todos os números maiores que 80.

```
class LambdaSimples
{
    static void Main ()
    {
        int[] valores = { 90, 71, 82, 93, 75, 82 };

        int maioresnumeros = valores.Where(n => n > 80).Count();

        Console.WriteLine("{0} números são maiores que 80", maioresnumeros);
    }
}
```

A seguir, você pode analisar um exemplo que expressa e deixa claro o escopo de variável em expressões lambda. As lambdas podem se referir a variáveis externas que estão no escopo do método que define a função lambda. Também podem estar no escopo do tipo que contém a expressão lambda. Logo, você pode considerar que as variáveis capturadas assim são armazenadas localmente para uso na expressão lambda, mesmo que, de alguma forma, elas saiam do escopo. Portanto, uma variável externa deve ser definitivamente atribuída para que possa ser consumida em uma expressão lambda. Você pode ver isso claramente no próximo exemplo, que mostra o uso de expressões lambda atribuídas para `del` e `del2`.

```
delegate bool D();
delegate bool D2(int i);

class Test
{
    D del;
    D2 del2;
    public void TestMethod(int input)
    {
        int j = 0;

        del = () => { j = 10;  return j > input; };

        del2 = (x) => {return x == j; };

        Console.WriteLine("j = {0}", j);
        bool boolResult = del();

        Console.WriteLine("j = {0}. b = {1}", j, boolResult);
    }

    static void Main()
    {
        Test test = new Test();
        test.TestMethod(5);

        bool result = test.del2(10);

        Console.WriteLine(result);

        Console.ReadKey();
    }
}
```

Lambda para recuperação de dados

As expressões lambda são muito utilizadas na busca de dados por meio do método `Where`, como você viu anteriormente. Uma expressão lambda dentro de um operador de consulta é avaliada por esse operador sob demanda e funciona continuamente em cada um dos elementos na sequência de entrada, e não na sequência completa. Os desenvolvedores podem, com uma expressão lambda, criar sua própria lógica nos operadores de consulta padrão. No código a seguir, o desenvolvedor usou o operador `Where` para recuperar os valores ímpares de uma lista dada, utilizando uma expressão lambda.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace exemplo{


   class Program {


      static void Main(string[] args) {


         int[] fibNum = { 1, 1, 2, 3, 5, 8, 13, 21, 34 };
         double valorMedio = fibNum.Where(num ⇒ num % 2 == 1).Average();


         Console.WriteLine(valorMedio);
         Console.ReadLine();
      }
   }
}
```

Você pode verificar que o uso de expressões lambda é muito útil para a redução de linhas de código e a simplificação de métodos de busca. Outra forma de realizar buscas de dados é utilizando os mecanismos do LINQ. A seguir, você vai ver, então, como funciona o LINQ!

LINQ

O LINQ é um componente do Microsoft.NET Framework que adiciona capacidades de consulta de dados nativos para idiomas .NET. Ele amplia o idioma C# pela adição de expressões de consulta, que são semelhantes às instruções

SQL. Além disso, pode ser usado para extrair e processar dados convenientemente de matrizes, classes enumeráveis, documentos XML, bancos de dados relacionais e fontes de dados de terceiros. Outros usos, que utilizam expressões de consulta como uma estrutura geral para compilar facilmente cálculos, incluem a construção de manipuladores de eventos (MICROSOFT, 2015).

O LINQ também define um conjunto de nomes de métodos (chamados **operadores de consulta padrão** ou **operadores de sequência padrão**). Além disso, estabelece as regras de tradução usadas pelo compilador para traduzir expressões de consulta de estilo fluente em expressões que usam esses nomes de métodos, expressões lambda e tipos anônimos.

Saiba mais

Muitos dos conceitos que o LINQ introduziu foram originalmente testados no projeto de pesquisa CQ da Microsoft. O LINQ foi lançado como uma parte importante do .NET Framework 3.5 em 19 de novembro de 2007.

O exemplo LINQ do código a seguir utiliza o método `Select` junto a uma expressão lambda. Desse modo, há um array de elementos criados a partir da própria expressão utilizando as informações contidas no retorno da query. Veja como o uso desses recursos permite a criação de queries avançadas que, se fossem traduzidas para SQL, iriam necessitar de maior esforço de programação.

```
using (Context context = new Context())
{
    var query = context.Products
        .Select(product => new
        {
            ProductId = product.ProductID,
            ProductName = product.Name
        });

    Console.WriteLine("Product Info:");
    foreach (var productInfo in query)
    {
        Console.WriteLine("Product Id: {0} Product name: {1} ",
            productInfo.ProductId, productInfo.ProductName);
    }
}
```

Nesse código, você pode verificar o uso de uma query `Select` utilizando o LINQ para selecionar apenas dois dos campos encontrados e adicioná-los em um array. Após, você pode notar que há um `foreach` que realiza um loop nesse array e imprime no console o nome e o ID de cada um dos produtos selecionados.

Você também pode criar queries por meio de um formato muito similar com as consultas SQL. Com elas, de forma geral, todas as operações de consulta LINQ consistem em três ações distintas:

- obter a fonte de dados;
- criar a consulta;
- executar a consulta.

No próximo exemplo, você pode ver como as três partes de uma operação de consulta são expressas em código-fonte. O exemplo usa uma matriz de números inteiros como fonte de dados por conveniência. No entanto, os mesmos conceitos também se aplicam a outras fontes de dados. Observe:

```
class IntroToLINQ
{
    static void Main()
    {

        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

        var numQuery =
            from num in numbers
            where (num % 2) == 0
            select num;

        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
    }
}
```

Você pode ver, nesse caso, a utilização de cláusulas `from`, `in`, `where` e `select`. Com elas, foi possível selecionar os números pares, já que é verificado o resto da divisão do número por 2.

Outra opção que você pode utilizar nas queries LINQ é o `orderby`. Ele permite que os resultados da pesquisa sejam ordenados de acordo com alguma lógica. No exemplo da a seguir, foi utilizado o valor `Key` do objeto retornado para ordenar. Você pode ordenar com qualquer outro atributo.

```
var custQuery =
    from cust in customers
    group cust by cust.City into custGroup
    where custGroup.Count() > 2
    orderby custGroup.Key
    select custGroup;
```

Você também pode utilizar o `group by` para fazer o agrupamento de registros seguindo determinada regra. No próximo código, o `group by` serve para agrupar as pessoas (people) de acordo com o carro (car).

```
var results = from p in persons
              group p.car by p.PersonId into g
              select new {
                  PersonId = g.Key,
                  Cars = g.ToList()
              };
```

Outro elemento importante é o `join`. Com ele, você pode realizar consultas utilizando várias tabelas que se relacionam entre si. Assim, deve fazer a junção dessas tabelas e selecionar determinadas informações que deseja. No último exemplo, você pode ver uma query LINQ que utiliza o `join` para selecionar dados de outra tabela e realiza duas verificações com `where`, uma em cada tabela. Logo, quando um elemento é selecionado, é criado um novo objeto de retorno com `account _ name e contact _ name`.

```
var query_where3 = from c in svcContext.ContactSet
                   join a in svcContext.AccountSet
                   on c.ContactId equals a.PrimaryContactId.Id
                   where a.Name.Contains("Contoso")
                   where c.LastName.Contains("Smith")
                   select new
                   {
                    account_name = a.Name,
                    contact_name = c.LastName
                   };
```

Além do uso de queries, o LINQ também se destina a facilitar a consulta de fontes de dados. Um dos usos mais populares do LINQ é a consulta de bancos de dados relacionais. No entanto, como você viu aqui, o LINQ pode consultar objetos. Isso não é tudo: o .NET Framework inclui bibliotecas que permitem que qualquer pessoa crie um provedor LINQ para consultar qualquer fonte de dados. Fora da caixa, a Microsoft envia LINQ para Objetos, LINQ para XML, LINQ para SQL (SQL Server) e LINQ para Entidades (Entity Framework).

Fique atento

Existem também provedores LINQ de terceiros que facilitam a consulta de fontes de dados especializadas. Por exemplo, existe um provedor do LINQ que permite consultar a API do Twitter, o LINQ no Twitter.

Referências

MICROSOFT. *Getting started with LINQ in C#*. Redmond, 2015. Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/getting-started-with-linq>. Acesso em: 02 nov. 2017.

MICROSOFT. *Lambda expressions (C# programming guide)*. Redmond, 2017. Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions>. Acesso em: 02 nov. 2017.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo: