

# ***II Encontro .NET em Brasília***

Janeiro de 2008

## **AVANÇOS NA LINGUAGEM C# 3.0 E SUA INTEGRAÇÃO COM O LINQ**

Rogério Moraes de Carvalho

Consultor e Instrutor de Tecnologias da Informação

[rogeriom@gmx.net](mailto:rogeriom@gmx.net)

# Agenda

- ◆ Introdução
  - ◆ A evolução da linguagem C#
  - ◆ Objetivos do projeto C# 3.0
- ◆ LINQ (*Language INtegrated Query*)
  - ◆ O que é LINQ?
  - ◆ LINQ to Objects
  - ◆ LINQ to XML
  - ◆ LINQ to DataSets
  - ◆ LINQ to SQL
  - ◆ LINQ to Entities

# Agenda

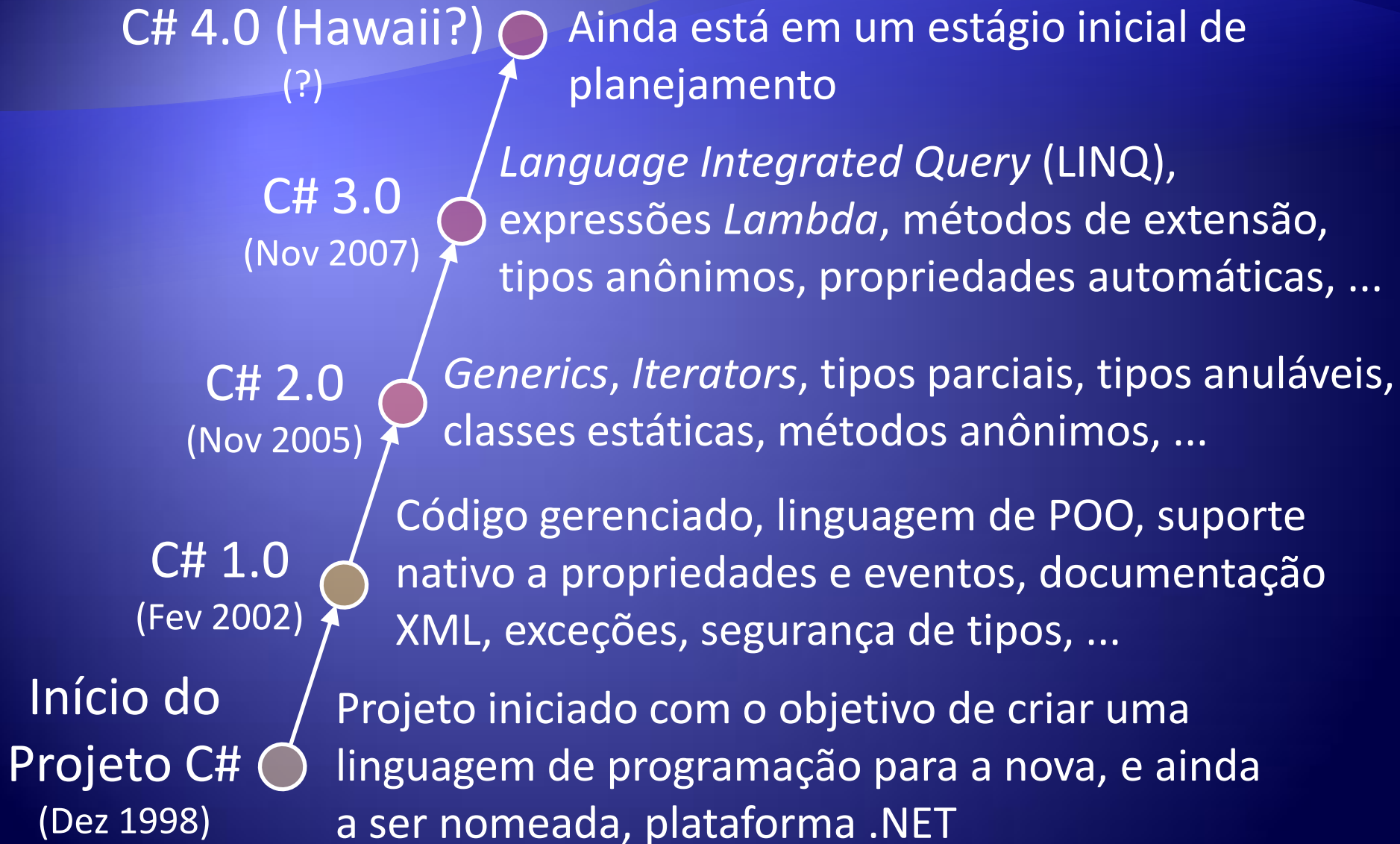
- ◆ Novidades na linguagem C# 3.0
  - ◆ Propriedades implementadas automaticamente
  - ◆ Iniciadores de objetos
  - ◆ Iniciadores de coleções
  - ◆ Variáveis locais e *arrays* com tipos deduzidos implicitamente
  - ◆ Métodos de extensão
  - ◆ Expressões *lambda*
  - ◆ Árvores de expressão
  - ◆ Tipos anônimos
  - ◆ Expressões de consulta
  - ◆ Métodos parciais

# Agenda

- ◆ Conclusão
  - ◆ Inovações na linguagem C#3.0
  - ◆ Onde obter informações adicionais
  - ◆ Perguntas & Respostas

# *Introdução*

# A evolução da linguagem C#



# Objetivos do projeto C# 3.0

- ◆ Integrar objetos, dados relacionais e XML
- ◆ Tornar a linguagem ainda mais concisa
- ◆ Adicionar construtores de programação funcional
- ◆ Não vincular a linguagem a APIs específicas
- ◆ Manter 100% de compatibilidade com a versão 2.0
- ◆ Integrar a linguagem C# com o LINQ

***LINQ***

***(Language INtegrated Query)***



# O que é LINQ?

C# 3.0

VB 9.0

Outras linguagens...

.NET LINQ (*Language INtegrated Query*)

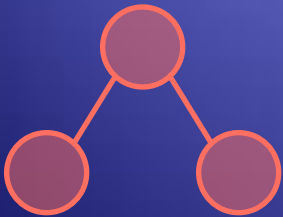
LINQ to  
Objects

LINQ to  
DataSets

LINQ to  
SQL

LINQ to  
Entities

LINQ to  
XML



Objetos



Bancos de dados  
relacionais

```
<agenda>
  <contato>
    <nome>... </nome>
    <email> ... </email>
    <fone/> ... </fone>
  </contato>
  ...
</agenda>
```

XML

# LINQ to Objects

- ◆ Linguagem integrada de consulta a objetos
  - ◆ Permite consultas diretas a coleções do tipo *System.Collections.Generic.Enumerable<T>*
  - ◆ Não necessita do uso de um provedor LINQ intermediário
- ◆ Nova abordagem na consulta a coleções
  - ◆ Elimina a necessidade de escrever laços **foreach** complexos para consultar coleções de objetos
  - ◆ Permite escrever códigos declarativos que descrevem o que deve ser recuperado
    - ◆ Sintaxe similar ao *Structured Query Language* (SQL)

# LINQ to XML

- ◆ Linguagem integrada de consulta para XML
  - ◆ Poder fornecido pelo XPath /XQuery
  - ◆ Porém, integrada às linguagens C# 3.0 e Visual Basic 9.0
- ◆ Experiência bem superior ao DOM
  - ◆ Construção funcional
  - ◆ Centralizado em elementos e não em documentos
  - ◆ Suporte simplificado a *namespaces* XML
  - ◆ Anotações
  - ◆ Mais rápido e menor

# LINQ to DataSets

- ◆ Linguagem integrada de consulta a DataSets
  - ◆ Simplifica e agiliza consultas a dados mantidos em *cache* com objetos DataSet
  - ◆ Permite realizar consultas coleções de objetos *DataRow*
    - ◆ Os membros da classe *DataRow* podem ser usados para criar consultas complexas e ricas
  - ◆ Assim como outras fontes de dados com suporte a LINQ, pode ser usada com a sintaxe baseada em métodos ou com a baseada em expressões de consulta

# LINQ to SQL

- ◆ Linguagem integrada para acesso a dados
  - ◆ Mapeia tabelas e registros para classes e objetos
  - ◆ Construído sobre o ADO.NET e .NET Transactions
- ◆ Mapeamento
  - ◆ Mapeados por atributos ou arquivo XML externo
  - ◆ Relacionamentos mapeados para propriedades
- ◆ Persistência
  - ◆ Rastreamento automático de alterações
  - ◆ Atualizações via SQL ou procedimentos armazenados
- ◆ Componente do .NET Framework 3.5
  - ◆ Atualmente funciona somente com o SQL Server

# LINQ to Entities

## ◆ ADO.NET Entity Framework

- ◆ Permite o desenvolvimento de aplicações com acesso a dados com ênfase no modelo conceitual
- ◆ Libera as aplicações de dependências de *engines* de bancos de dados particulares
- ◆ O mapeamento entre o modelo conceitual da aplicação e os esquemas específicos de bancos de dados podem mudar sem impactos para a aplicação
- ◆ Atualmente em versão Beta 3

## ◆ Componentes

- ◆ EDM (Entity Data Model)
  - ◆ Corresponde a um Modelo Entidade-Relacionamento (MER)
- ◆ LINQ to Entities

*Novidades na  
linguagem C# 3.0*

# Propriedades implementadas automaticamente

- ◆ Frequentemente as propriedades têm implementações triviais que seguem a um padrão
  - ◆ O *get* e o *set* fornecem acesso de leitura e escrita para um campo privado
    - ◆ O *get* retorna o valor atual do campo privado
    - ◆ O *set* atribui um novo valor para o campo privado
- ◆ Propriedades implementadas automaticamente fornecem uma sintaxe mais concisa
  - ◆ O compilador do C# gera campos privados de suporte (*backing fields*) automaticamente



# Propriedades implementadas automaticamente

```
public class Produto
{
    public string Nome;
    public decimal Preço;
}
```

É uma má prática de POO  
definir campos públicos

Sintaxe tradicional

```
public class Produto
{
    private string nome;
    private decimal preco;

    public string Nome {
        get { return nome; }
        set { nome = value; }
    }

    public decimal Preço {
        get { return preco; }
        set { preco = value; }
    }
}
```

# Propriedades implementadas automaticamente

```
public class Produto
{
    public string Nome { get; set; }
    public decimal Preco { get; set; }
}
```

## Sintaxe Concisa - Propriedades implementadas automaticamente

O compilador do C# gera campos privados de suporte para cada propriedade implementada automaticamente de acordo com o seguinte padrão:

`<Propriedade>k__BackingField`

É obrigatório ter o *get* e o *set*

```
public class Produto
{
    public int ID { get; private set; }
    public string Nome { get; set; }
    public decimal Preco { get; set; }
}
```

É possível usar modificadores para limitar a acessibilidade do *get* ou do *set*

An orange 3D-style arrow pointing to the right, with a slight shadow and a beveled edge. It is positioned in the upper left quadrant of the slide.

***DEMO***

A yellow rounded rectangle with a slight gradient and a shadow, centered horizontally in the lower half of the slide.

***Propriedades implementadas  
automaticamente***

# Iniciadores de objetos

- ◆ Um iniciador de objetos consiste de uma seqüência de iniciadores de membros
  - ◆ Os iniciadores de membros são delimitados por chaves e separados por vírgulas
  - ◆ Cada iniciador de membro pode atribuir um valor para uma propriedade ou um campo do objeto
- ◆ A nova sintaxe combina a criação e iniciação de um objeto num único passo
  - ◆ A sintaxe de iniciação concisa é semanticamente equivalente à invocação do construtor e posterior atribuição de valores para cada um dos membros

# Iniciadores de objetos

```
public class Ponto {  
    public Ponto() { }  
    public Ponto(int x) : this(x, 0) { }  
    public Ponto(int x, int y) {  
        X = x;  
        Y = y;  
    }  
  
    public int X { get; set; }  
    public int Y { get; set; }  
}
```

Parênteses  
opcionais

Atribuições para  
propriedades ou  
campos

Ponto p1 = new Ponto { X = 1, Y = 2 };

Ponto p1 = new Ponto();  
p1.X = 1; p1.Y = 2;

Chamada a um  
construtor com  
parâmetro(s)

Ponto p2 = new Ponto(1) { Y = 2 };

Ponto p2 = new Ponto(1);  
p2.Y = 2;

An orange 3D-style arrow pointing to the right, with a slight shadow and a beveled edge. It is positioned in the upper left quadrant of the slide.

***DEMO***

A yellow rounded rectangle with a slight gradient and a soft shadow, centered horizontally in the lower half of the slide.

***Iniciadores de objetos***

# Iniciadores de coleções


- ◆ Um iniciador de coleção pode ser aplicado para iniciar os elementos de um objeto que:
  - ◆ seja de um tipo que implemente a interface *System.Collections.Generic.IEnumerable<T>*
  - ◆ tenha um método público *Add*
- ◆ Os elementos num iniciador de coleção devem ser delimitados por chaves e separados por vírgulas
- ◆ O iniciador instancia a coleção vazia e acrescenta cada elemento usando o método *Add*
- ◆ Pode-se utilizar qualquer construtor da coleção na expressão de iniciação

# Iniciadores de coleções

Deve implementar a interface *IEnumerable<T>*

Deve ter um método público *Add*

```
List<int> quadrados = new List<int> { 1, 4, 9, 16, 25 };
```



```
List<int> quadrados = new List<int>();  
numbers.Add(1);  
numbers.Add(4);  
numbers.Add(9);  
numbers.Add(16);  
numbers.Add(25);
```

O método *Add* pode ter mais de um parâmetro

```
Dictionary<int, string> extenso = new Dictionary<int, string> {  
    { 0, "Zero" }, { 1, "Um" }, { 2, "Dois" }, { 3, "Três" } };
```



An orange arrow pointing to the right, with the word "DEMO" written inside in a stylized, italicized font. The arrow has a 3D effect with a shadow.

***DEMO***

A yellow rounded rectangle with a 3D effect and a shadow, containing the text "Iniciadores de coleções" in a bold, italicized font.

***Iniciadores de coleções***

# Variáveis locais e *arrays* com tipos deduzidos implicitamente

- ◆ Recurso que permite a omissão do tipo de dado na declaração de uma variável local
  - ◆ Variáveis locais de tipo implícito são declaradas com o tipo **var**
  - ◆ O tipo é deduzido a partir da expressão de inicialização da variável em tempo de compilação
    - ◆ Portanto, uma declaração de tipo implícito **deve** incluir uma expressão de inicialização
  - ◆ Variáveis somente podem ser declaradas com tipos definidos implicitamente dentro de um escopo local
- ◆ *Arrays* com tipos implícitos podem ser inicializados omitindo-se o tipo usando-se a sintaxe: **new [ ] { ... }**

# Variáveis locais e *arrays* com tipos deduzidos implicitamente

```
int i = 10;  
string s = "Teste";  
double d = 1.0;  
int[] pares = new int[] { 2, 4, 6, 8, 10 };  
Dictionary<int, Contato> contatos = new Dictionary<int, Contato>();
```

```
var i = 10;  
var s = "Teste";  
var d = 1.0;  
var pares = new [] { 2, 4, 6, 8, 10 };  
var contatos = new Dictionary<int, Contato>();
```

O tipo do *array* é deduzido a partir dos tipos dos elementos listados entre chaves

O tipo é deduzido a partir da expressão de inicialização do lado direito



***DEMO***

***Variáveis locais e arrays com tipos  
deduzidos implicitamente***

# Métodos de extensão

- ♦ Métodos de extensão permitem estender as funcionalidades de tipos existentes com a definição de novos métodos que são invocados usando a sintaxe normal de métodos de instância
  - ♦ São métodos estáticos declarados com uso da palavra chave **this** como modificador em seus primeiros parâmetros
    - ♦ Devem ser declarados em classes estáticas não genéricas
  - ♦ Somente ficam disponíveis se o *namespace* associado for trazido para o escopo
  - ♦ Um método de instância tem prioridade em relação a um método de extensão com a mesma assinatura

# Métodos de extensão

```
namespace ExtensoesString
```

```
{
```

```
    public static class StringAvancado
```

```
    {
```

```
        public static string Direita(this string str, int comprimento) {  
            return str.Substring(str.Length - comprimento);
```

```
        }
```

```
    }
```

```
}
```

Classe estática não genérica

Método de extensão

```
using ExtensoesString;
```

Traz as extensões para o escopo

```
string novidades = "Novidades do C# 3.0";  
string linguagem = novidades.Direita(6);
```

*IntelliSense*

novidades.Direita(6)

↓

StringAvancado.Direita(  
 novidades, 6)

An orange 3D-style arrow pointing to the right, with a slight shadow and a beveled edge. It is positioned in the upper left quadrant of the slide.

***DEMO***

A yellow rounded rectangle with a slight gradient and a soft shadow, centered horizontally in the middle of the slide.

***Métodos de extensão***

# Expressões lambda

- ◆ O C# 2.0 introduziu os métodos anônimos
  - ◆ Eles permitem escrever código *inline* onde *delegates* são esperados
- ◆ O C# 3.0 introduziu as expressões lambda
  - ◆ Elas fornecem uma sintaxe mais concisa para escrever métodos anônimos



# Expressões lambda

```
public delegate bool Predicate<T>(T obj);

public class List<T>
{
    public List<T> FindAll(Predicate<T> match) {
        List<T> result = new List<T>();
        foreach (T item in this) {
            if (match(item)) {
                result.Add(item);
            }
        }
        return result;
    }
    ...
}
```

# Expressões lambda

```
public class Teste
{
    static void Main() {
        Agenda agenda = new Agenda();
        List<Contato> contatos = agenda.Contatos;
        List<Contato> mulheres =
            contatos.FindAll(
                new Predicate<Contato>(filtrarMulheres)
            );
    }

    private static bool filtrarMulheres(Contato c) {
        return c.Sexo == Sexo.Feminino;
    }
}
```

Instanciando o *delegate* **Predicate<T>** com uma referência para o método **filtrarMulheres**

Método compatível com o *delegate* **Predicate<Contato>**

# Expressões lambda

```
public class Teste
{
    static void Main() {
        Agenda agenda = new Agenda();
        List<Contato> contatos = agenda.Contatos;
        List<Contato> mulheres =
            contatos.FindAll(
                delegate(Contato c) {
                    return c.Sexo == Sexo.Feminino;
                }
            );
    }
}
```

Método anônimo

# Expressões lambda

```
public class Teste
{
    static void Main() {
        Agenda agenda = new Agenda();
        List<Contato> contatos = agenda.Contatos;
        List<Contato> mulheres =
            contatos.FindAll(c => c.Sexo == Sexo.Feminino);
    }
}
```

Expressão lambda



***DEMO***



***Expressões lambda***

# Tipos anônimos

XXX

```
var contato = new {  
    Nome = "Maria da Silva Pereira",  
    Sexo = Sexo.Feminino,  
    Email = "mariasp@gmail.com",  
    DataNascimento = new DateTime(1992, 7, 23)  
};
```

```
public class XXX  
{  
    public string Nome { get; set; }  
    public Sexo Sexo { get; set; }  
    public string Email { get; set; }  
    public DateTime DataNascimento { get; set; }  
}
```

An orange 3D-style arrow pointing to the right, with a slight shadow and a beveled edge. It contains the word "DEMO" in a stylized font.

***DEMO***

A yellow rounded rectangle with a slight gradient and a shadow, containing the text "Tipos anônimos".

***Tipos anônimos***

# Expressões de consulta

Inicia com  
**from**

Zero ou mais **from**,  
**join**, **let**, **where** ou  
**orderby**

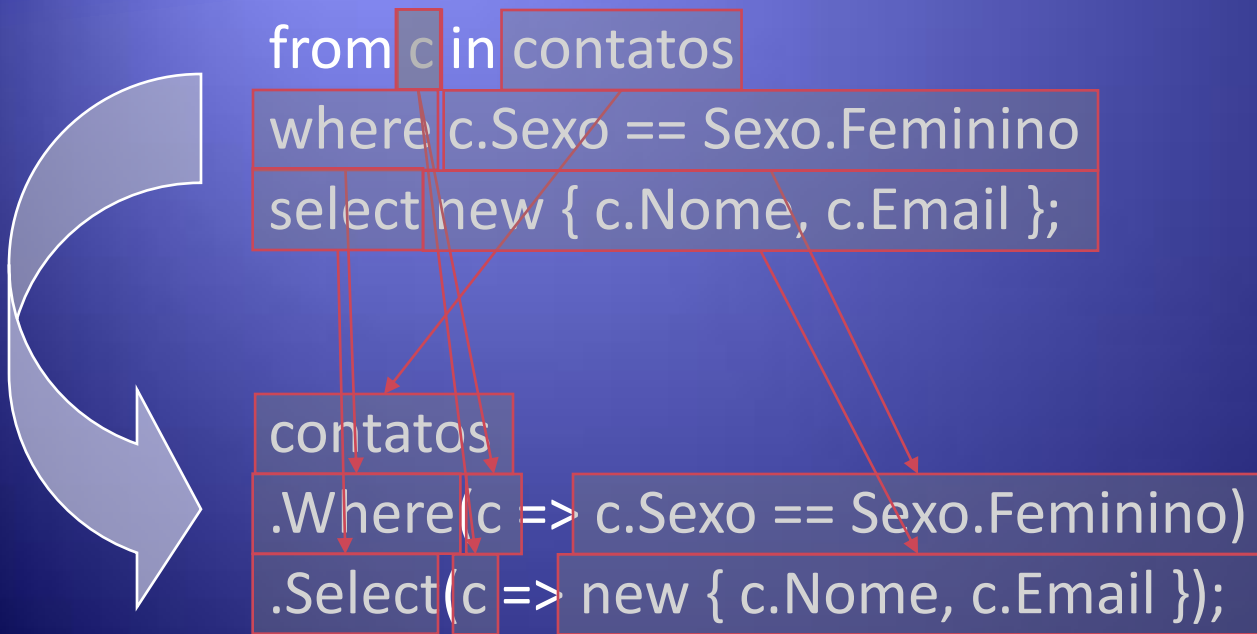
```
from id in fonte  
{ from id in fonte |  
  join id in fonte on expr equals expr [ into id ] |  
  let id = expressão |  
  where condição |  
  orderby id1, id2, ... [ ascending | descending ] }  
select expressão | group expressão by chave  
[ into id ]
```

Termina com  
**select** ou **group by**



# Expressões de consulta

- ◆ Expressões de consulta são traduzidas para invocações de métodos de extensão
  - ◆ Where, Join, OrderBy, Select, GroupBy, ...



An orange arrow pointing to the right, with the word "DEMO" written inside in a stylized, italicized font. The arrow has a 3D effect with a shadow.

***DEMO***

A yellow rounded rectangle with a 3D effect and a shadow, containing the text "Expressões de consulta" in a bold, italicized font.

***Expressões de consulta***

*Conclusão*

# Inovações na linguagem C# 3.0

```
var mulheres =
```

```
from c in contatos
```

```
where c.Sexo == Sexo.Feminino
```

```
select new { c.Nome, c.Email };
```

Expressões de consulta

Tipos implícitos em variáveis locais

```
var mulheres =  
contatos
```

Expressões lambda

```
.Where(c => c.Sexo == Sexo.Feminino)
```

```
.Select(c => new { c.Nome, c.Email });
```

Árvores de expressão

Propriedades automáticas

Métodos parciais

Métodos de extensão

Tipos anônimos

Iniciadores de objetos

# Onde obter informações adicionais

- ◆ Visual C# Developer Center
  - ◆ <http://msdn2.microsoft.com/en-us/vcsharp/> (en-US)
  - ◆ <http://www.microsoft.com/brasil/msdn/csharp/> (pt-BR)
- ◆ Visual C# 2008 Express Edition
  - ◆ <http://www.microsoft.com/express/vcsharp/>
- ◆ Microsoft Visual C# 2008 Step by Step (MS Press)
  - ◆ <http://www.microsoft.com/mspress/books/11298.aspx>
- ◆ Pro LINQ  
Language Integrated Query in C# 2008 (Apress)
  - ◆ <http://www.apress.com/book/view/1590597893>

# Perguntas & Respostas

- ◆ Demonstrações motivadas por perguntas dos participantes
- ◆ Informações para contato
  - ◆ Consultoria e Treinamento
    - ◆ Rogério Moraes de Carvalho
    - ◆ [rogeriom@gmx.net](mailto:rogeriom@gmx.net)