Dell IT Academy

Desenvolvimento de Sistemas

INTRODUÇÃO A LINQ

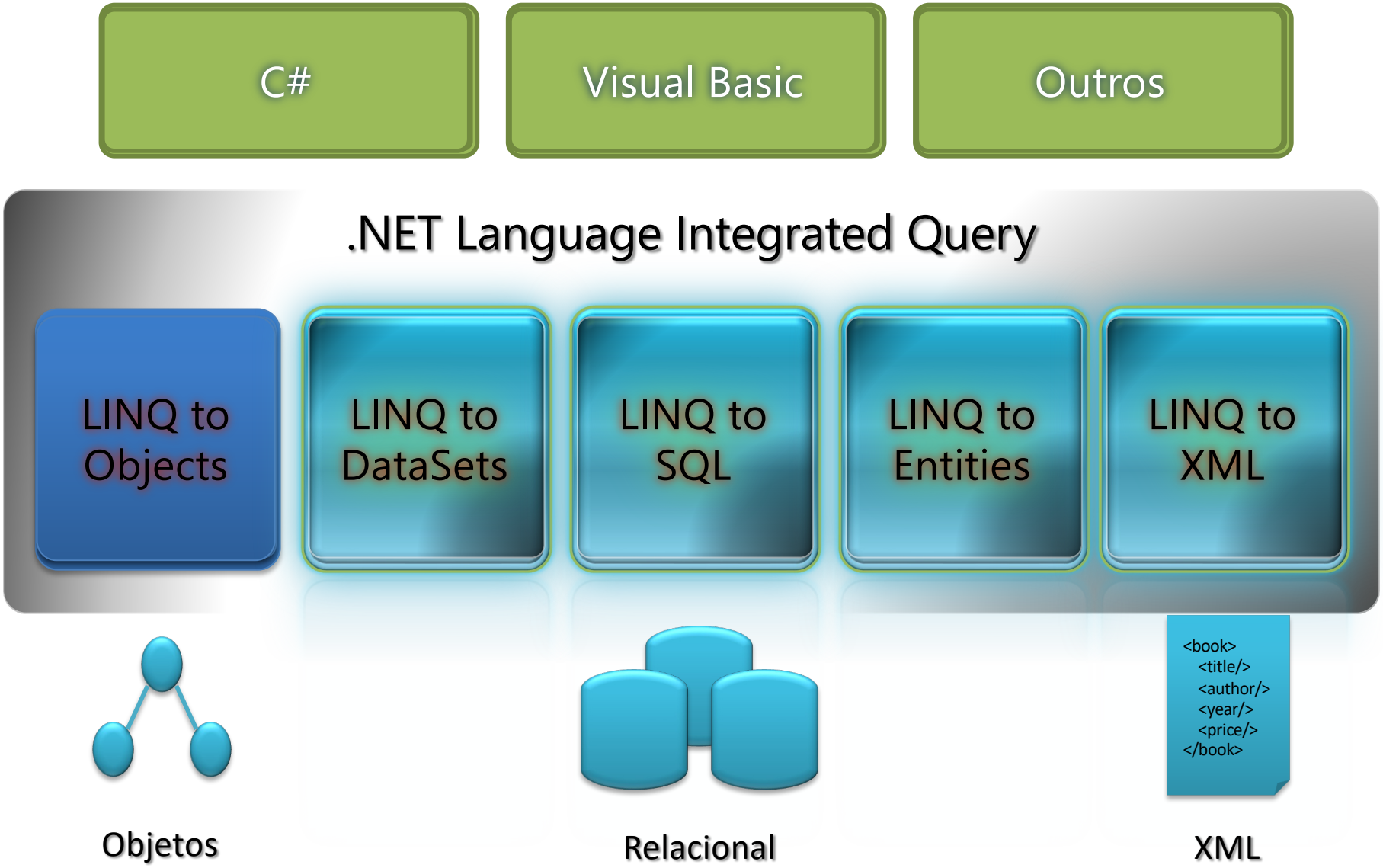
LINQ

- *Language Integrated Query*
  - Linguagem de consulta integrada à linguagem de programação
- Documentação:
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/
  - https://docs.microsoft.com/en-us/dotnet/csharp/linq/

LINQ

- Três idéias básicas:
  - Dados são Objetos
  - Mudança de programação imperativa para programação declarativa
  - Trabalha sobre diversos elementos (objetos, bases relacionais, xml)

LINQ

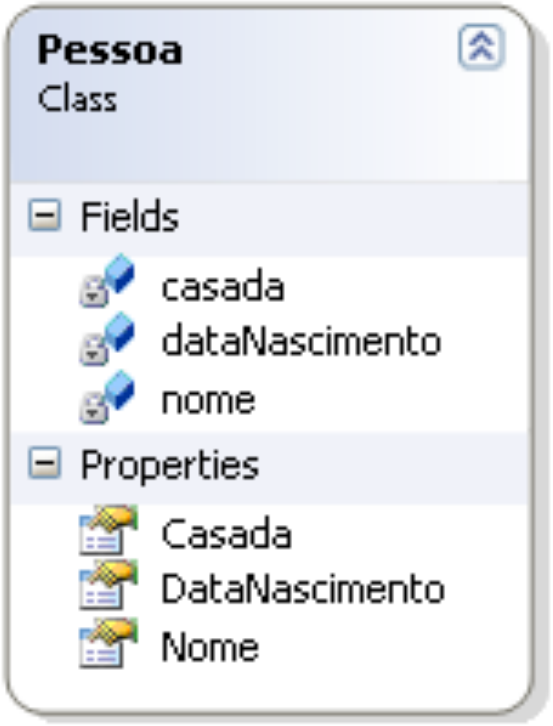

C#
Visual Basic
Outros
.NET Language Integrated Query
LINQ to Objects
LINQ to DataSets
LINQ to SQL
LINQ to Entities
LINQ to XML
<book>
<title/>
<author/>
<year/>
<price/>
</book>
Objetos
Relacional
XML

Demonstração

- Utilizaremos a ferramenta LINQPad
- Disponível gratuitamente em http://linqpad.net/

Demonstração

- Um objeto Pessoa com propriedades para nome, data de nascimento e informação se é casada ou não



Demonstração

- Uma coleção de objetos é criada como uma lista de pessoas
  - Ana, 14/03/1980, casada
  - Paulo, 23/10/1978, casado
  - Maria, 10/01/2000, solteira
  - Carlos, 12/12/1999, solteiro

Demonstração

- Deseja-se obter uma lista de pessoas que são casadas

Demonstração

- Deseja-se obter uma lista de pessoas que são casadas
- Solução tradicional
  - Envolve laços de repetição, enumeradores, condicionais, etc
  - Focada em “como executar a consulta”

```
foreach (var p in pessoas)
 {
  if (p.Casada) {
    casadas.Add(p);
  }
}
```

Demonstração

- Deseja-se obter uma lista de pessoas que são casadas
- Solução Linq
  - Programação declarativa
  - Focada em “quais são os dados da consulta”

```
var casadasLinq =
  from p in pessoas
  where p.Casada
  select p;
```

LINQ

- Importante!
- LINQ depende do provedor
  - LINQ to _____
  - Distribuição do .NET não possui provedores de acesso para fonte de dados genéricas
  - É necessário buscar o provedor de acesso desejado

LINQ to Objects

- Operações envolvem três partes:
  - Obter a fonte de dados
    - Neste caso estamos trabalhando com coleções de objetos que implementam a interface IEnumerable<T>
  - Criar a consulta
    - Utilizando sintaxe do Linq ou métodos de extensão
  - Utilizar a consulta
    - Usualmente através de código que “percorre” os dados do resultado da consulta

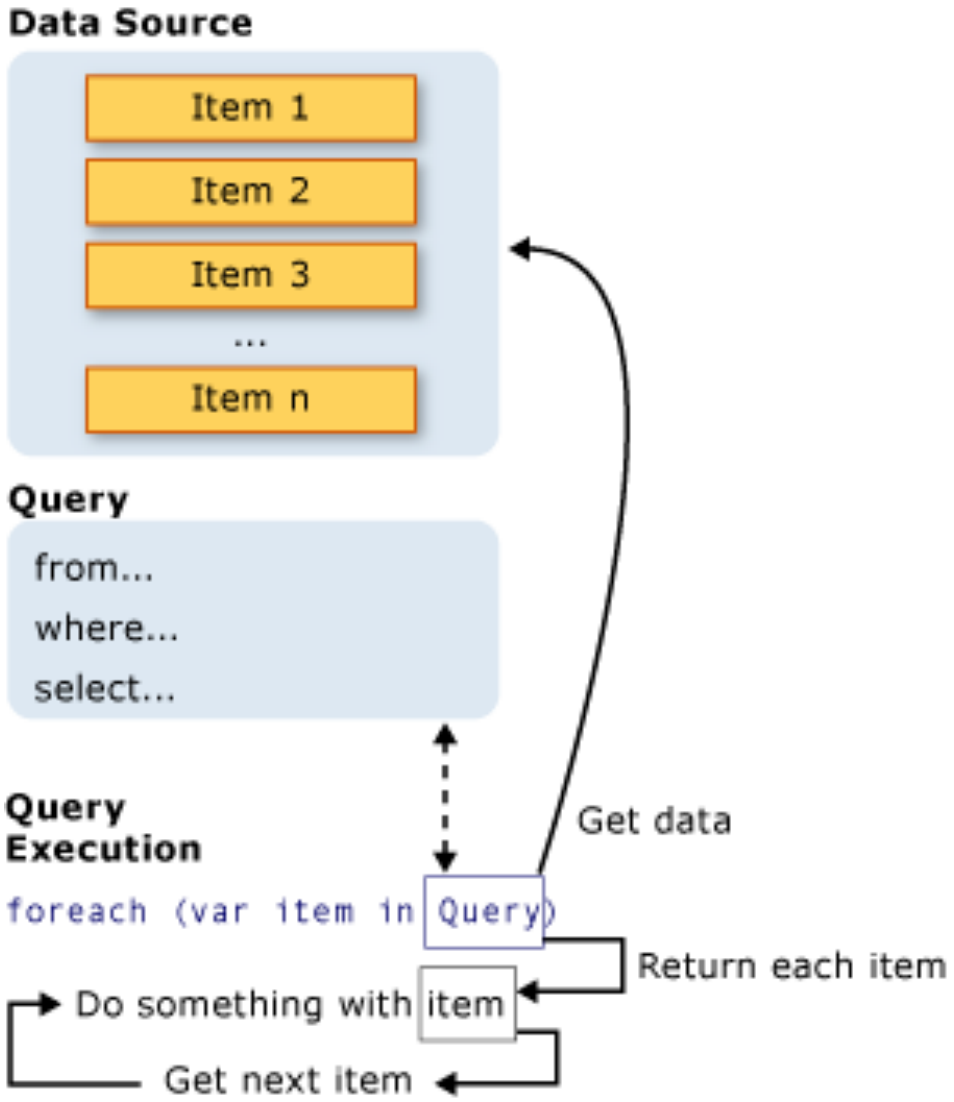
LINQ to Objects

- Três elementos principais:
  - *from* – especifica a fonte de dados
  - *where* – aplica um filtro
  - *select* – especifica o tipo do retorno (projeção)

LINQ to Objects

- Importante!!!
  - A definição da consulta LINQ não executa a consulta
  - Usualmente, uma enumeração da coleção realiza essa tarefa ou um método de extensão do LINQ

LINQ to Objects



LINQ to Objects

- Dois tipos de sintaxes:
  - Método (também chamada de *method syntax*)
    - Utiliza *lambda expressions*
  - Consulta (também chamada de *query syntax*)
- Importante:
  - Toda consulta em uma *query syntax* é executada pelo framework como chamadas de métodos
  - Algumas consultas somente podem ser realizadas na *method syntax*

LINQ to Objects

- Query syntax

```
string[ ] palavras = new [ ] {“livro”, “mesa”, “abajur”};
var filtradas = from p in palavras
                where p.Length > 4
                select p;
foreach(string s in filtradas) {…}
```

LINQ to Objects

- Method syntax

```
string[ ] palavras = new [ ] {“livro”, “mesa”, “abajur”};
var filtradas = palavras.Where(p => p.Length > 4);
foreach(string s in filtradas) {…}
```

Where é um método de extensão que aplica a expressão a cada palavra do arranjo

Esta expressão define um delegate que retorna um bool e recebe como parâmetro uma string

Demonstração

- Utilize o projeto com a classe Pessoa

LINQ to Objects

- Filtragem
  - Obter as pessoas casadas que nasceram após 01/01/1980

```
var resultado =
  from p in pessoas
  where p.Casada && p.DataNascimento >= new
     DateTime(1980,1,1)
  select p;
```

LINQ to Objects

- Seleção de uma propriedade
  - Obter o nome das pessoas casadas

```
var resultado =
  from p in pessoas
  where p.Casada
  select p.Nome;
```

Resultado é uma coleção de string

LINQ to Objects

- Seleção de várias propriedades
  - Obter o nome e data de nascimento das pessoas casadas

```
var resultado =
  from p in pessoas
  where p.Casada
  select new {p.Nome, p.DataNascimento};
```

Resultado é uma coleção de objetos anônimos com duas propriedades

LINQ to Objects

- Ordenação simples
  - Obter as pessoas ordenadas pelo nome

```
var resultado =
  from p in pessoas
  orderby p.Nome
  select p;
```

LINQ to Objects

- Ordenação simples
  - Obter as pessoas ordenadas pelo nome em ordem inversa

```
var resultado =
  from p in pessoas
  orderby p.Nome descending
  select p;
```

LINQ to Objects

- Ordenação múltipla
  - Obter as pessoas ordenadas pela data de nascimento e por nome

```
var resultado =
  from p in pessoas
  orderby p.DataNascimento, p.Nome
  select p;
```

LINQ to Objects

- Agrupamento simples
  - Obter as pessoas agrupadas em casadas e solteiras

```
var resultado =
  from p in pessoas
  group p by p.Casada;
```

Resultado é uma hierarquia de objetos pessoa agrupados por uma chave do tipo bool

LINQ to Objects

- Agrupamento simples com projeção
  - Obter as pessoas agrupadas em casadas e solteiras

```
var resultado =
  from p in pessoas
  group p by p.Casada into grupoPessoas
  select new {Categoria=grupoPessoas.Key,
     Pessoas=grupoPessoas};
```

LINQ to Objects

- Função de agregação
  - Obter o número de pessoas casadas

```
var resultado =
 (from p in pessoas
  where p.Casada
  select p).Count();
```

Operadores

- LINQ apresenta um conjunto de operadores disponibilizados via métodos de extensão
- Nem todo operador possui uma versão equivalente na notação *query expression*
- Ver https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/standard-query-operators-overview

Operadores

- Operadores são classificados em diferentes meios de execução
  - Execução Imediata
  - Execução Postergada

Operadores

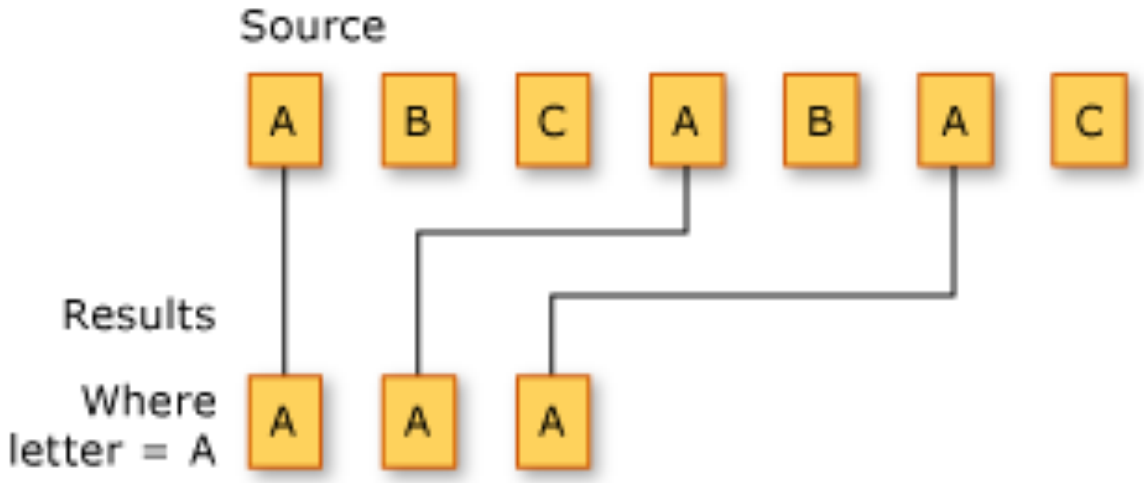
- Execução imediata
  - Fonte de dados processada no ponto de inserção do operador
  - Ex.: Count

Operadores

- Execução postergada
  - Fonte de dados processada quando os elementos são enumerados
  - Dois tipos:
    - Streaming – não necessita ler toda a fonte de dados antes de retornar resultados
      - Ex.: Select
    - Não-streaming – necessita ler toda a fonte de dados antes de retornar resultados
      - Ex.: OrderBy

Filtragem

- Operadores: OfType, Where



Filtragem

- Filtragem
  - Obter as pessoas casadas que nasceram após 01/01/1980

```
var resultado =
  from p in pessoas
  where p.Casada && p.DataNascimento >= new
     DateTime(1980,1,1)
  select p;
```

Filtragem

- Exemplos:
  - Ex1, Ex2

Projeção

- Operadores:
  - Select – projeta valores com base em uma função de transformação, um resultado para cada fonte
    - Usa select na *query expression*
  - SelectMany - projeta valores com base em uma função de transformação e “achata” o resultado em uma sequência única
    - Usa múltiplos from na *query expression*

Projeção

- Exemplos Select:
  - Ex3, Ex4, Ex5

Projeção

- Projeção simples
  - Obter as flores de um bouquet

```
List<Bouquet> bouquets = new List<Bouquet>() {
        new Bouquet { Flowers = new List<string> {
    "sunflower", "daisy", "daffodil", "larkspur" }},
        new Bouquet{ Flowers = new List<string> { "tulip",
    "rose", "orchid" }},
        new Bouquet{ Flowers = new List<string> {
    "gladiolis", "lily", "snapdragon", "aster", "protea"
    }},
        new Bouquet{ Flowers = new List<string> {
    "larkspur", "lilac", "iris", "dahlia" }}
    };
```

Projeção

- Projeção simples
  - Obter as flores de um bouquet

```
var query1 = bouquets.Select(bq => bq.Flowers);
foreach (IEnumerable<String> collection in query1)
        foreach (string item in collection)
            Console.WriteLine(item);

var query2 = bouquets.SelectMany(bq => bq.Flowers);
foreach (string item in query2)
        Console.WriteLine(item);
```

Projeção

- Projeção com SelectMany
  - Obter as palavras de uma lista de frases

```
List<String> frases = …;
var query = from frase in frases
            form palavra in frase.Split(‘ ‘)
            select palavra;
```

Ordenação

- Operadores: OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse

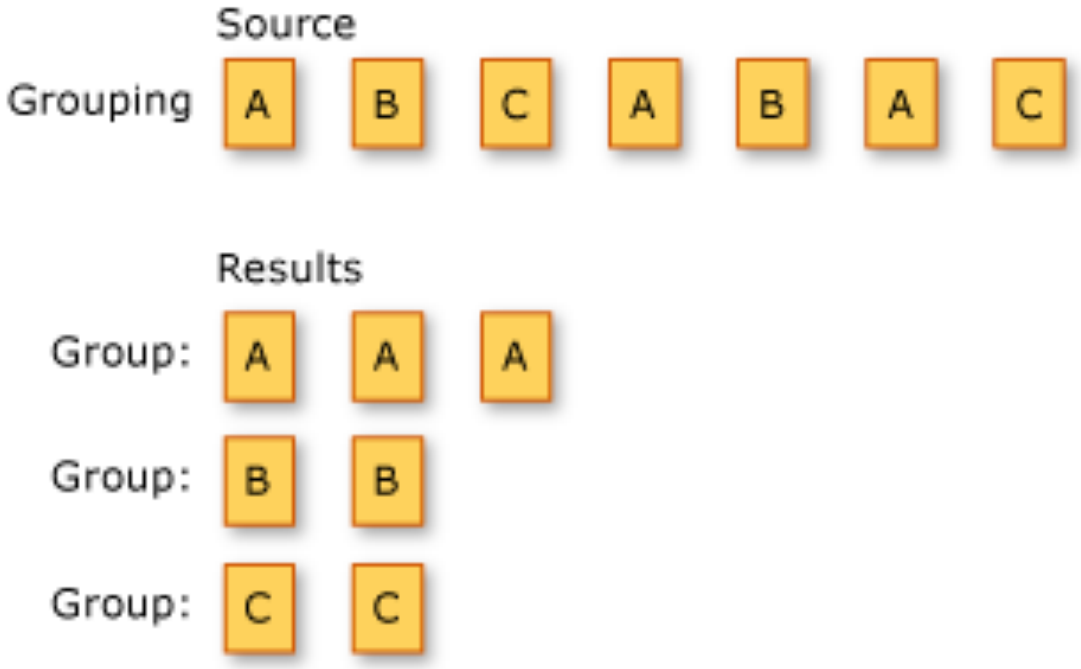
Ordenação

- Ordenação simples
  - Obter as pessoas ordenadas pelo nome em ordem inversa

```
var resultado =
  from p in pessoas
  orderby p.Nome descending
  select p;
```

Agrupamento

- Operadores: GroupBy, ToLookup



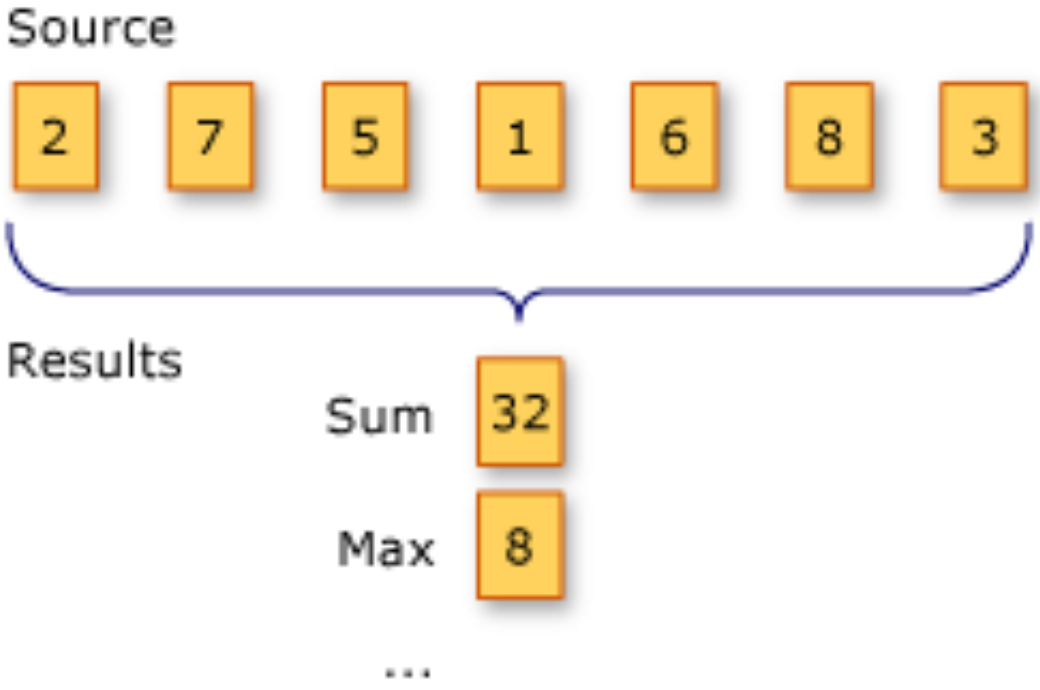
Agrupamento

- Agrupamento simples com projeção
  - Obter as pessoas agrupadas em casadas e solteiras

```
var resultado =
  from p in pessoas
  group p by p.Casada into grupoPessoas
  select new {Casados=grupoPessoas.Key,
     Pessoas=grupoPessoas};
```

Agregação

- Operadores: Aggregate, Average, Count, LongCount, Max, Min, Sum



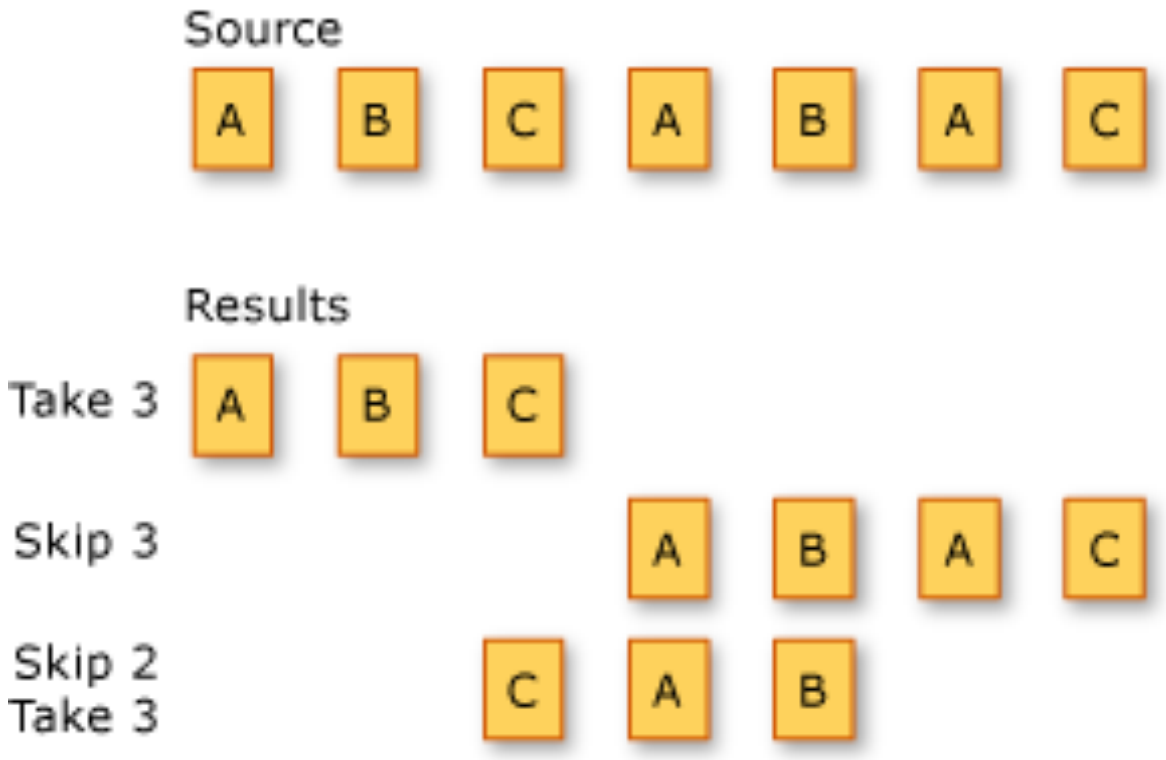
Agregação

- Função de agregação
  - Obter o número de pessoas casadas

```
var resultado =
 (from p in pessoas
  where p.Casada
  select p).Count();
```

Particionamento

- Operadores: Skip, SkipWhile, Take, TakeWhile



Particionamento

- Função de particionamento
  - Obter as primeiras 3 pessoas

```
var resultado =
 (from p in pessoas
  select p).Take(3);
```

Elementos

- Operadores: ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault

Elementos

- Acesso com elemento
  - Obter o primeiro mês de uma lista de meses

```
List<int> months = …;
int firstMonth = months.First();
```

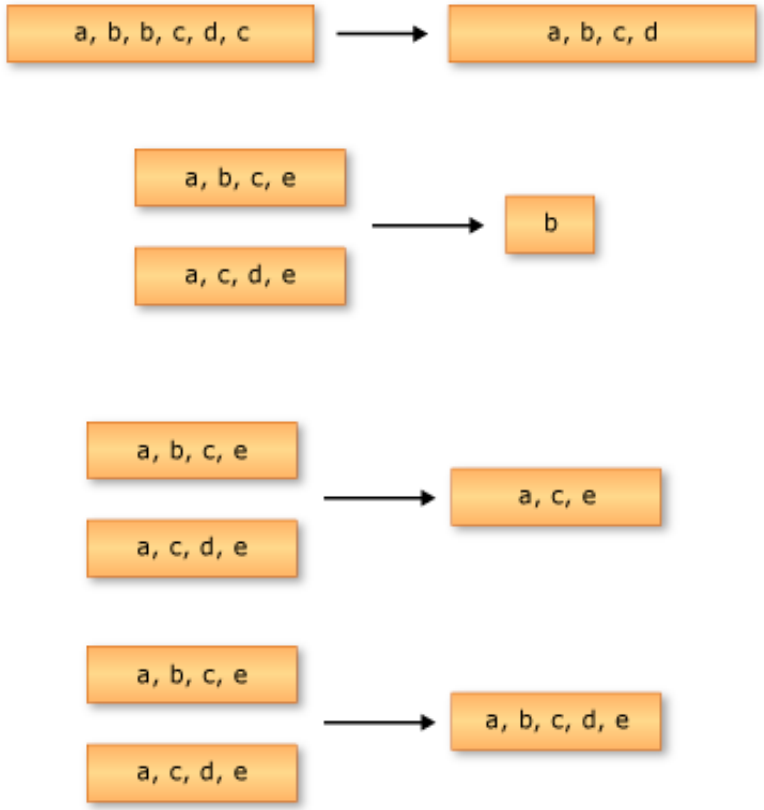
Elementos

- Acesso com elemento padrão
  - Obter o primeiro mês de uma lista de meses ou 0 se a lista for vazia

```
List<int> months = new List<int> { };
int firstMonth = months.FirstOrDefault();
```

Conjunto

- Operadores: Distinct, Except, Intersect, Union



Conjunto

- Diferença
  - Encontrar a diferença entre dois conjuntos de strings

```
var resultado =
  conjunto1.Except(conjunto2);
```

Concatenação

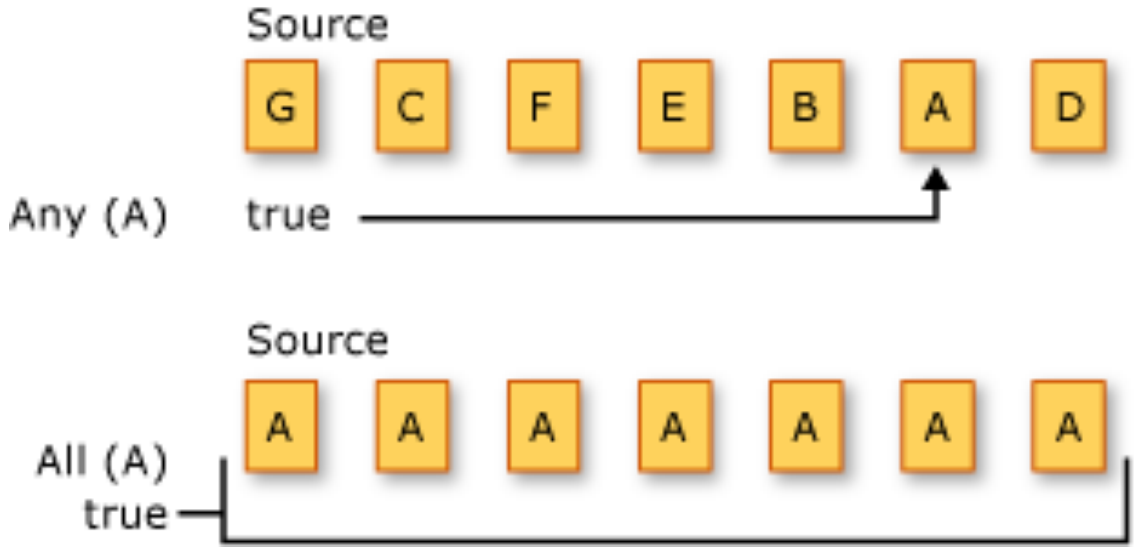
- Operadores: Concat

Igualdade

- Operadores: SequenceEqual
  - Objetos devem implementar uma interface de comparação, como *IEquatable<>*

Quantificador

- Operadores: All, Any, Contains



Geração

- Operadores:
  - DefaultIfEmpty – substitui uma coleção vazia com uma coleção de valor padrão singleton
  - Empty – retorna uma coleção vazia
  - Range – retorna uma coleção que possui uma sequência de números
  - Repeat – retorna uma coleção com um valor repetido

Geração

```
Pet defaultPet = new Pet { Name = "Default Pet", Age = 0 };
List<Pet> pets2 = new List<Pet>();
foreach (Pet pet in pets2.DefaultIfEmpty(defaultPet)) {...}
```

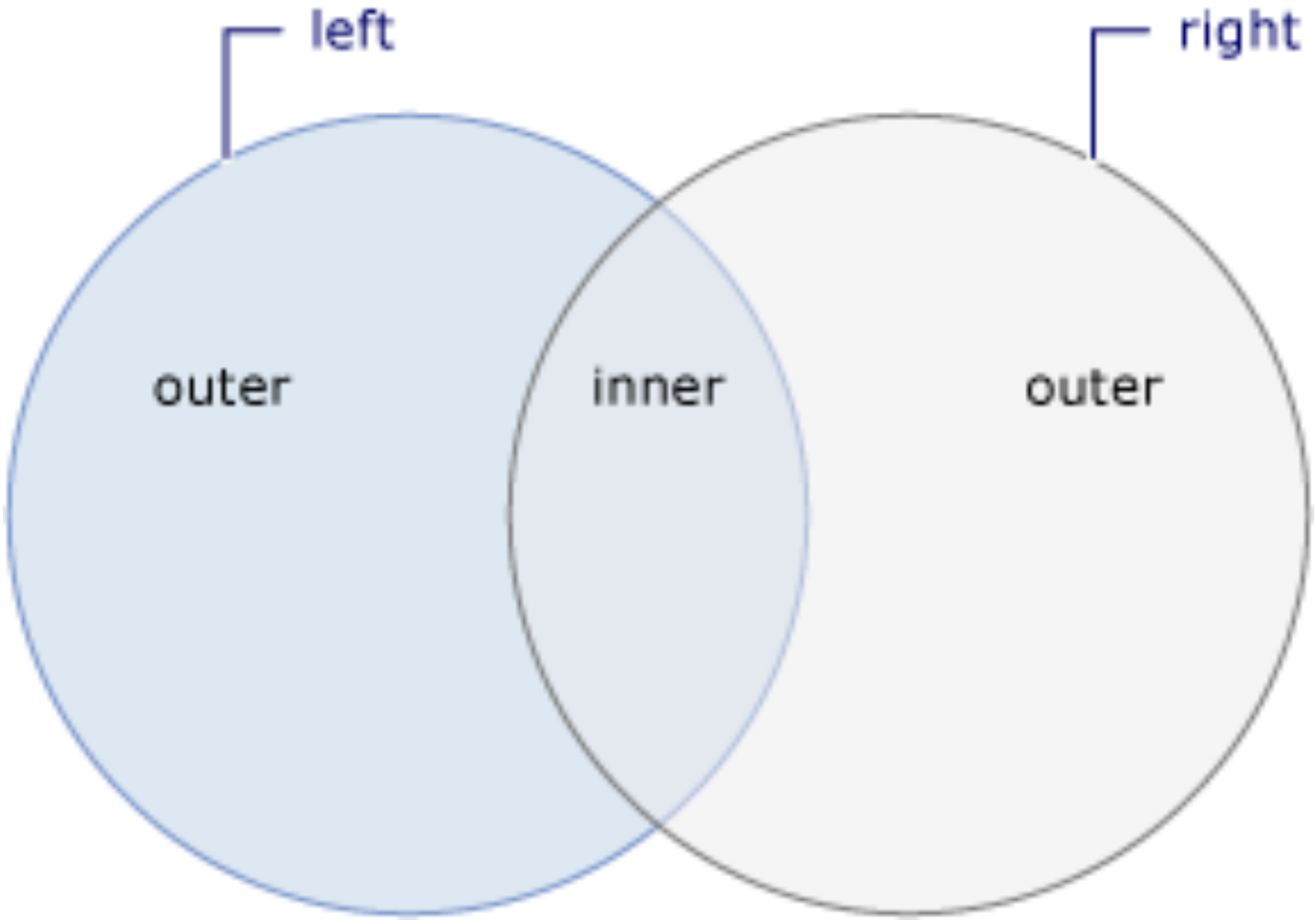
Conversão

- Operadores: AsEnumerable, AsQueryable, Cast, OfType, ToArray, ToDictionary, ToList, ToLookup

Join

- Operadores:
  - Executam equijoins
  - Join – junção de duas sequências com base em igualdade das chaves, extraindo pares de valores
    - Resulta em *inner join*
    - Usa join ... in ... on ... equals ... na *query expression*
  - GroupJoin – junção de duas sequências com base em igualdade das chaves, agrupando os resultados
    - Resulta em um superconjunto de *inner join* e *left outer join*
    - Usa join ... in ... on ... equals ... into ... na *query expression*

Join



Join

- Inner join
  - Obter uma coleção de pares de pessoas com seus animais de estimação

```
Person magnus = new Person { FirstName = "Magnus", LastName = "Hedlund" };
    Person terry = new Person { FirstName = "Terry", LastName = "Adams" };
    Person charlotte = new Person { FirstName = "Charlotte", LastName =
      "Weiss" };
    Person arlene = new Person { FirstName = "Arlene", LastName = "Huff" };
    Person rui = new Person { FirstName = "Rui", LastName = "Raposo" };
Pet barley = new Pet { Name = "Barley", Owner = terry };
    Pet boots = new Pet { Name = "Boots", Owner = terry };
    Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };
    Pet bluemoon = new Pet { Name = "Blue Moon", Owner = rui };
    Pet daisy = new Pet { Name = "Daisy", Owner = magnus };
List<Person> people = new List<Person> { magnus, terry, charlotte, arlene,
      rui };
List<Pet> pets = new List<Pet> { barley, boots, whiskers, bluemoon, daisy };
```

Join

- Inner join
  - Obter uma coleção de pares de pessoas com seus animais de estimação

```
var resultado =
  from person in people
  join pet in pets on person equals pet.Owner
  select new { OwnerName = person.FirstName, PetName =
     pet.Name };
```

Join

- Left outer join
  - Obter uma coleção de pares de pessoas com seus animais de estimação, mesmo que não possua um animal de estimação

```
var resultado =
  from person in people
  join pet in pets on person equals pet.Owner into gj
  from subpet in gj.DefaultIfEmpty()
  select new { person.FirstName, PetName = (subpet == null
    ? String.Empty : subpet.Name) };
```

Subconsultas

- Operadores LINQ podem ser aplicados em sequência
- Lembrar que as consultas somente são executadas quando necessárias
- Composição com *into* e *let*