

Linq

Language Integrated Query

Linq

Significa “Language Integrated Query”. É um conjunto de métodos e classes introduzidos na versão 3.5 da .NET Framework da Microsoft (~2007).

Estende a plataforma (VB, C#) com mecanismos de filtragem (Where), projeção (Select, SelectMany), agrupamento (GroupBy), agregação (Aggregate, Sum) e ordenação (OrderBy), entre outros operadores de manipulação de dados.

Duas sintaxes: “method syntax” (mais parecido com código comum) e “query syntax” (mais parecido com SQL).

Suporte para fazer queries a várias fontes de dados: SQL, XML, objetos simples, etc.

As duas sintaxes do Linq

Query syntax é utilizado mais quando se “tem uma ideia” do comando SQL equivalente. Devido à sua sintaxe semelhante ao SQL “puro” (from, where, select, join group by, etc.), é por vezes mais fácil fazer a query para obter os dados pretendidos. Quem é fluente com o SQL de Bases de Dados I pode achar este método mais fácil para começar.

Method syntax é mais flexível do que *query syntax*, mas apenas em situações onde é necessário usar operadores não disponíveis na *query syntax* (ex: Sum), ou quando necessitamos de fazer múltiplos Where ou Select.

É possível combinar os dois. Uma query com *query syntax* é compilada para *method syntax*.

Namespace “System.Linq”

Para trabalhar com Linq, é obrigatório importar o namespace “System.Linq”. Sem este namespace, nem a *query syntax* nem o *method syntax* funcionarão.

Para importar um namespace em C#, usa-se o “using” no início do ficheiro:
using System.Linq;

Um dos dois erros à direita poderão surgir se não estiver importado. Versões mais recentes do Visual Studio (2015+) já conseguem ajudar se fizerem Ctrl+. (ponto) com o cursor sobre o erro.

```
static void Main(string[] args)
{
    var nums = new[] { 1, 2, 3, 4, 5 };
    var media = nums.Average(x => x);
    Console.WriteLine(media);
}
```

'int[]' does not contain a definition for 'Average' and no extension method 'Average' is visible. Show potential fixes (Alt+Enter or Ctrl+.)

```
static void Main(string[] args)
{
    var nums = new[] { 1, 2, 3, 4, 5 };
    var dobroDeCadaNumeroPar = from n in nums
                                where n % 2 == 0
                                select n * 2;
}
```

(local variable) int[] nums
Could not find an implementation of the query pattern for source type 'int[]'. 'Where' not found. Are you missing a reference to 'System.Linq'? Show potential fixes (Alt+Enter or Ctrl+.)

```
13
14
15
16 static void Main(string[] args)
17 {
18     var nums = new[] { 1, 2, 3, 4, 5 };
19
20     var dobroDeCadaNumeroPar = from n in nums
21                                where n % 2 == 0
22                                select n * 2;
23 }
24
25 using System.Linq;
26
27
```

CS1935 Could not find an implementation of the query pattern for source type 'int[]'. 'Where' not found. Are you missing a reference to 'System.Linq' or a using directive for 'System.Linq'?

...
using System.Collections.Generic;
using System.Linq;
using System.Web;
...

Preview changes

Sequências - IEnumerable<T> e IQueryable<T>

Linq opera sobre a interface IEnumerable<T>. Esta interface genérica dá suporte ao ciclo “**foreach**”, usado para iteração em C#. O T é o tipo de cada elemento na sequência (ex: int, string, Agente, Multa).

Arrays, Listas, Dicionários, Sets, entre outros objetos da .net Framework implementam esta interface e podem, portanto, ser usados com Linq.

Side note: Objetos que implementam IQueryable<T> (como os objetos de Base de Dados da Entity Framework) fazem uso desta interface para poderem traduzir código C# nas queries dos sistemas respectivos (ex: SQL). Infelizmente, isso significa que por vezes estamos limitados ao *target* das nossas queries em Linq.

Arrow Functions / Lambda Functions / etc.

O *method syntax* do Linq faz forte uso da sintaxe das *funções lambda*. Isto é apenas um nome *fancy* para funções com o seguinte aspecto:

(parâmetro_1, ..., parâmetro_N) => resultado

Quando as chavetas após o => são omitidas, o operador => significa “return”. Quando eu coloco chavetas, sou obrigado a especificar o return quando quero devolver dados:

(num1, num2) => { return num1 + num2; }

Só as funções com chavetas a seguir ao => suportam coisas como “if”, “for”, e múltiplas operações.

Notas sobre os exemplos seguintes

Os *prints* seguintes são do LinqPad, um programa gratuito que serve para testar código C# ou VB.

Verão nos exemplos o código “.Dump()”; isto não é uma funcionalidade do Linq, e é específico a este programa. Serve apenas para fins de debug (por prints).

O link para o programa: <https://www.linqpad.net/>

Nota: A versão gratuita não suporta autocomplete.

Operações comuns - Filtrar

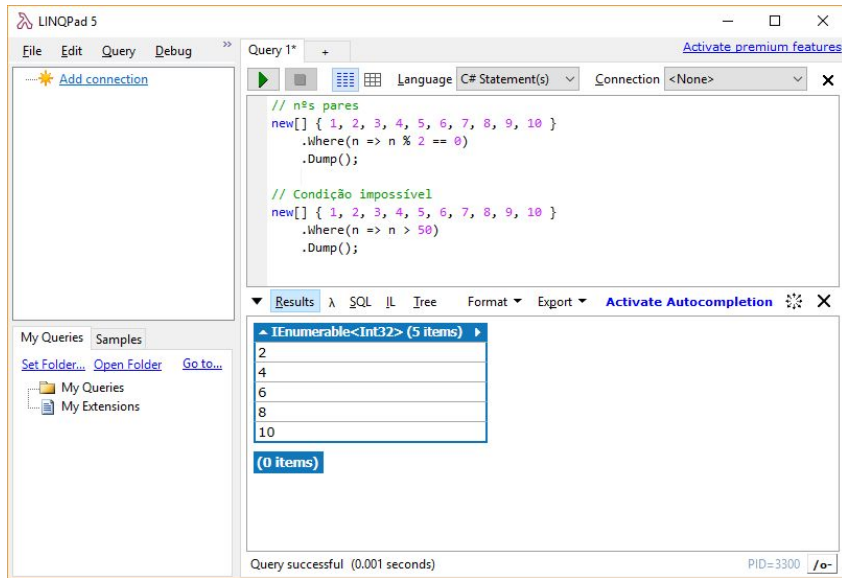
Where

O Where, tal como em SQL, serve para *filtrar* os objetos numa *sequência*.

Recebe, por parâmetro, uma função, que será executada sobre cada um dos objetos presentes na sequência, e que deve devolver o resultado de uma condição booleana.

Se nenhum dos objetos presentes na sequência passar na condição especificada por parâmetro, o resultado é uma *sequência vazia*.

Em JavaScript, o análogo é o “filter”, presente nos Arrays.



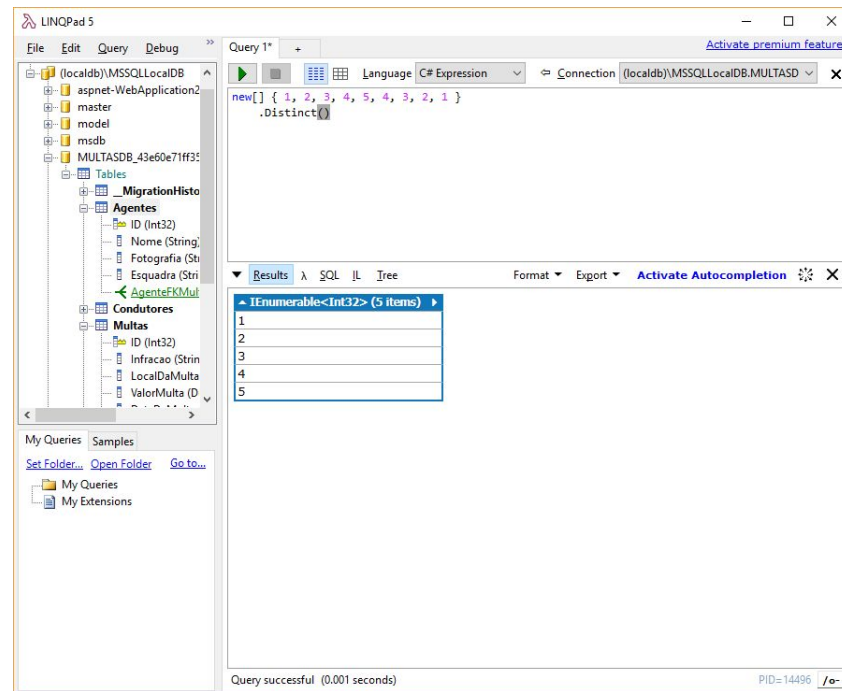
Distinct

Serve para obter uma sequência de itens distintos.

A condição de igualdade dependerá do “target”; se for Entity Framework, provavelmente será usado o “distinct” do SQL, se forem objetos comuns, será usado o GetHashCode ou Equals.

Infelizmente, não suporta a possibilidade de especificar o campo no qual fazer a igualdade.

Em casos raros, para objectos comuns, permite especificar um objeto que vai fazer a comparação dos objetos (provavelmente não suportado com Entity Framework).



Operações comuns - Projeção

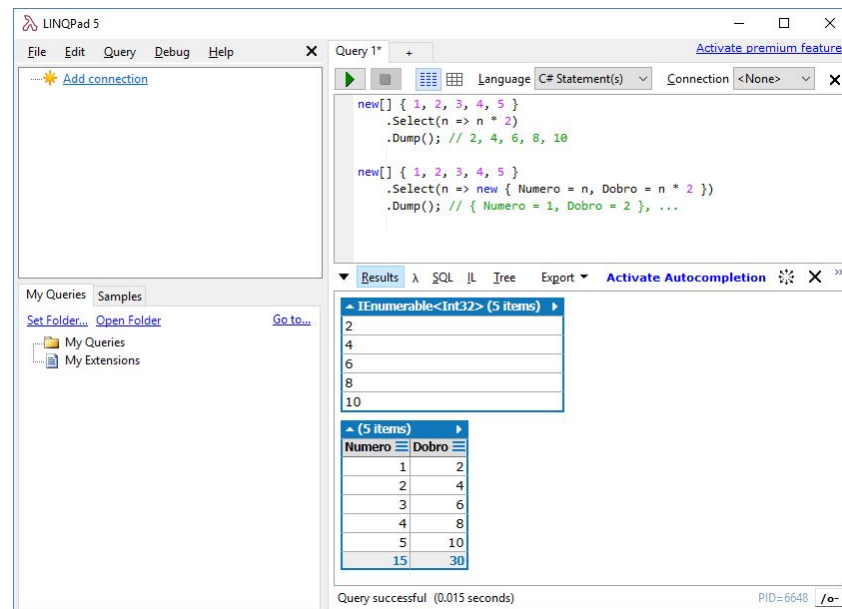
Select

Serve para projetar (transformar) uma sequência de valores noutra sequência de outros valores. Pode ser usado para especificar campos, ou até cálculos.

Recebe, por parâmetro, uma função, que será invocada para cada elemento na sequência.

O resultado (return) da invocação será colocado na sequência de resultado, pela mesma ordem da sequência de input.

Em JavaScript, o análogo é o “map”, presente nos Arrays.



Operações comuns - Ordenação

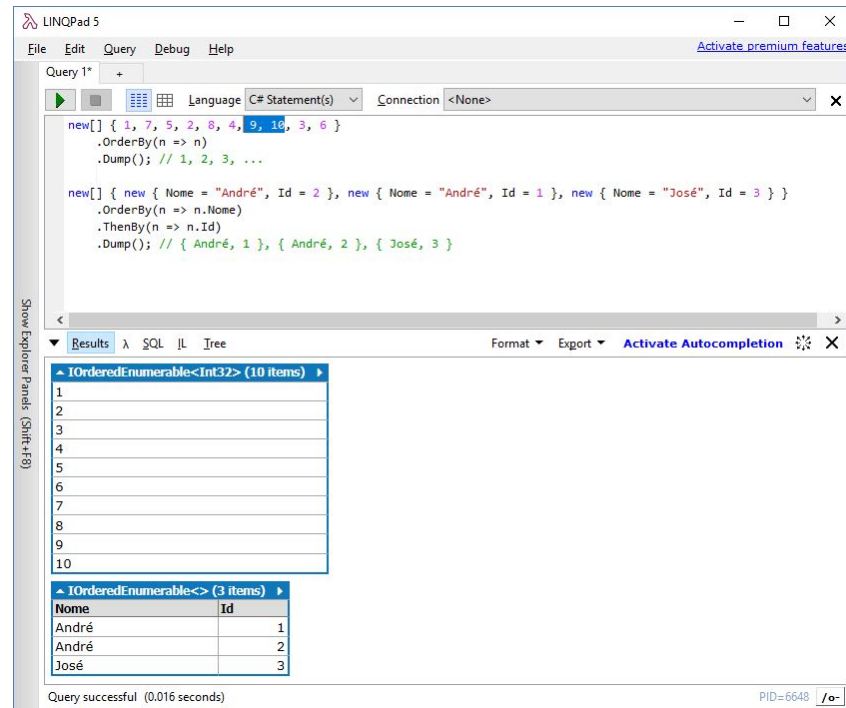
OrderBy / ThenBy / OrderByDescending / ThenByDescending

O OrderBy permite fazer a ordenação ascendente de uma sequência através de um campo ou expressão.

O campo é especificado através da função que lhe é passada por parâmetro. Os valores do campo especificado são depois usados para produzir uma nova sequência com os elementos ordenados.

O ThenBy serve para especificar níveis adicionais de ordenação (ex: “pelo nome, e depois pelo id”), e usa a mesma sintaxe que o OrderBy.

As versões “Descending” são usadas para fazer ordenação decrescente.



```
new[] { 1, 7, 5, 2, 8, 4, 9, 10, 3, 6 }
.OrderBy(n => n)
.Dump(); // 1, 2, 3, ...

new[] { new { Nome = "André", Id = 2 }, new { Nome = "André", Id = 1 }, new { Nome = "José", Id = 3 } }
.OrderBy(n => n.Nome)
.ThenBy(n => n.Id)
.Dump(); // { André, 1 }, { André, 2 }, { José, 3 }
```

Results

10 items

Id
1
2
3
4
5
6
7
8
9
10

3 items

Nome	Id
André	1
André	2
José	3

Query successful (0.016 seconds)

Operações comuns - Agregação

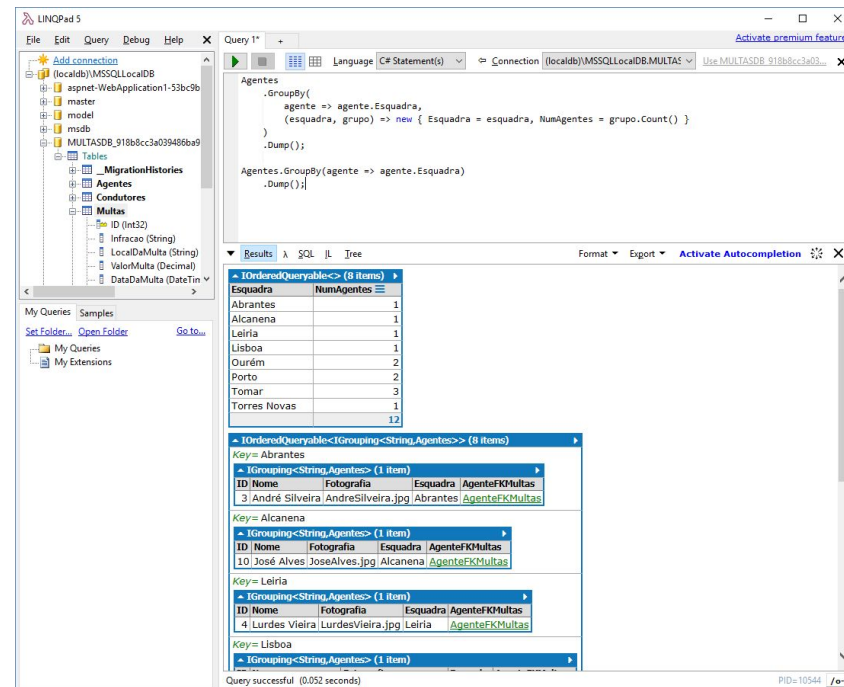
GroupBy

O GroupBy permite agrupar elementos por uma chave, tal como em SQL.

A chave é especificada através da função passada no primeiro parâmetro do GroupBy.

Além disso, é possível especificar uma segunda função para transformar os resultados.

Essa função é chamada com dois parâmetros: chave e elementos agrupados para essa chave, e pode ser usada para operações como contar o nº de elementos por chave.



The screenshot shows the LINQPad 5 interface. The left pane displays a file explorer with a project named 'aspnet-WebApplication1-53bc9b' containing a 'model' folder with a 'multas' table. The main editor shows a C# query using GroupBy to count agents by team. The results pane shows a table with 8 items, grouped by team (Esquadra) and agent count (NumAgentes).

```
Agents
    .GroupBy(
        agente => agente.Esquadra,
        (esquadra, grupo) => new { Esquadra = esquadra, NumAgentes = grupo.Count() }
    )
    .Dump();

Agents.GroupBy(agente => agente.Esquadra)
    .Dump();
```

Esquadra	NumAgentes
Abrantes	1
Alcanena	1
Leiria	1
Lisboa	1
Ourém	2
Porto	2
Tomar	3
Torres Novas	1
	12

The results are also displayed as a series of grouped tables for each team:

- Key = Abrantes**

ID	Nome	Fotografia	Esquadra	AgenteFKMultas
3	André Silveira	AndreSilveira.jpg	Abrantes	AgenteFKMultas
- Key = Alcanena**

ID	Nome	Fotografia	Esquadra	AgenteFKMultas
10	José Alves	JoseAlves.jpg	Alcanena	AgenteFKMultas
- Key = Leiria**

ID	Nome	Fotografia	Esquadra	AgenteFKMultas
4	Lurdes Vieira	LurdesVieira.jpg	Leiria	AgenteFKMultas
- Key = Lisboa**

ID	Nome	Fotografia	Esquadra	AgenteFKMultas

Query successful (0.052 seconds)

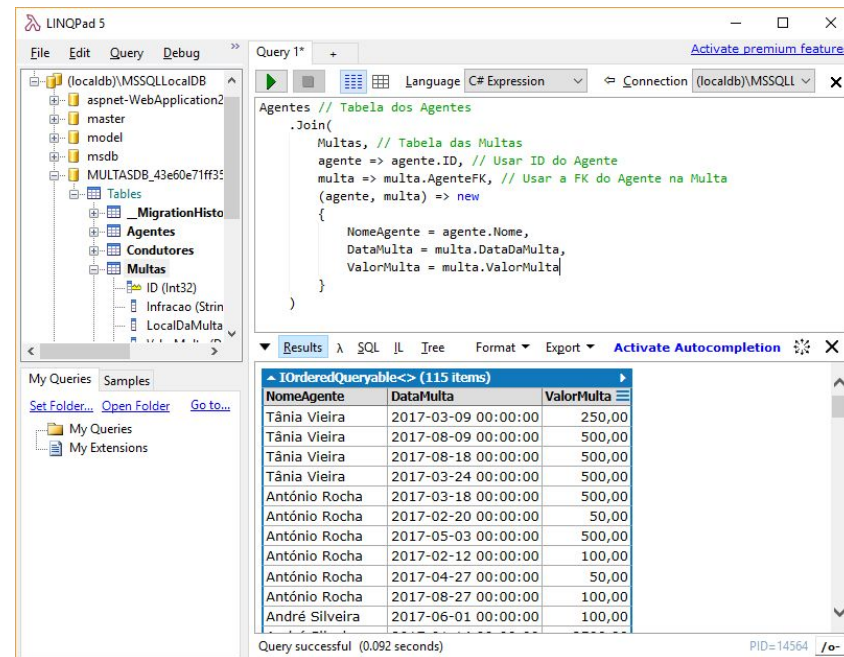
Join

O Join permite combinar duas sequências, tal como o Inner Join do SQL.

Esta função recebe, por parâmetro, a sequência que vai ser junta.

Além disso, recebe duas funções que servem para extrair as chaves das sequências (SQL: on a.id = b.id).

Por último, recebe uma função que recebe os elementos combinados, e serve para produzir um resultado com cada elemento.



The screenshot displays the LINQPad 5 interface. The left pane shows a project tree with a database connection to (localdb)\MSSQLLocalDB. The main editor shows a C# query using the `Join` method to combine the `Agentes` and `Multas` tables. The query selects agent names, multa dates, and values. The results pane shows 115 items with columns `NomeAgente`, `DataMulta`, and `ValorMulta`.

```
Agentes // Tabela dos Agentes
.Join(
    Multas, // Tabela das Multas
    agente => agente.ID, // Usar ID do Agente
    multa => multa.AgenteFK, // Usar a FK do Agente na Multa
    (agente, multa) => new
    {
        NomeAgente = agente.Nome,
        DataMulta = multa.DataDaMulta,
        ValorMulta = multa.ValorMulta
    }
)
```

NomeAgente	DataMulta	ValorMulta
Tânia Vieira	2017-03-09 00:00:00	250,00
Tânia Vieira	2017-08-09 00:00:00	500,00
Tânia Vieira	2017-08-18 00:00:00	500,00
Tânia Vieira	2017-03-24 00:00:00	500,00
António Rocha	2017-03-18 00:00:00	500,00
António Rocha	2017-02-20 00:00:00	50,00
António Rocha	2017-05-03 00:00:00	500,00
António Rocha	2017-02-12 00:00:00	100,00
António Rocha	2017-04-27 00:00:00	50,00
António Rocha	2017-08-27 00:00:00	100,00
André Silveira	2017-06-01 00:00:00	100,00

Query successful (0.092 seconds) PID=14564

Operações comuns - Finalização

Operadores de finalização

Estes operadores permitem obter um único resultado de uma sequência, ou guardar uma sequência num List, Dictionary, ou Array, para que possa ser reutilizado.

O Linq é *lazy* (preguiçoso), os operadores Select, Where, GroupBy, OrderBy, etc., não serão invocados até que os valores sejam precisos. Isto é feito por questões de desempenho e memória, especialmente com sequências grandes.

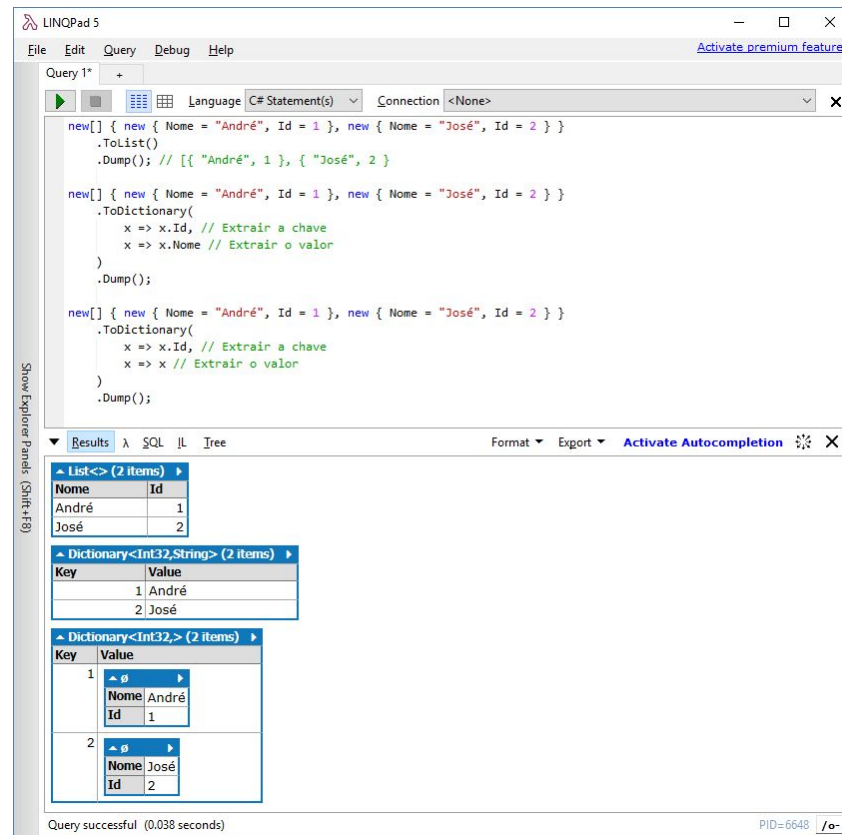
Uma coisa a ter em conta é que múltiplas iterações (foreach) numa sequência podem implicar a múltiplas queries. O uso destes operadores guarda os resultados em objetos de forma a serem reutilizáveis.

ToList / ToDictionary / ToArray

Estes operadores executam a query guardam o resultado em memória.

O ToList coloca todos os elementos da sequência num List. O ToArray é semelhante, excepto que guarda num Array. É preferível usar o ToList, por questões de desempenho e flexibilidade.

O ToDictionary permite construir um Dictionary<K,V> a partir dos dados de uma sequência, e de duas funções, uma para extrair as chaves, e outra para extrair os valores.



The screenshot shows the LINQPad 5 interface with a C# query and its results. The query defines three data structures: a List, a Dictionary<Int32, String>, and a Dictionary<Int32, > (where > represents a complex object). The results pane shows the output of these queries.

Query 1*

```
new[] { new { Nome = "André", Id = 1 }, new { Nome = "José", Id = 2 } }
.ToList()
.Dump(); // [{ "André", 1 }, { "José", 2 } ]

new[] { new { Nome = "André", Id = 1 }, new { Nome = "José", Id = 2 } }
.ToDictionary(
    x => x.Id, // Extrair a chave
    x => x.Nome // Extrair o valor
)
.Dump();

new[] { new { Nome = "André", Id = 1 }, new { Nome = "José", Id = 2 } }
.ToDictionary(
    x => x.Id, // Extrair a chave
    x => x // Extrair o valor
)
.Dump();
```

Results

- List<T> (2 items)**

Nome	Id
André	1
José	2
- Dictionary<Int32, String> (2 items)**

Key	Value
1	André
2	José
- Dictionary<Int32, > (2 items)**

Key	Value				
1	<table border="1"><thead><tr><th>Nome</th><th>Id</th></tr></thead><tbody><tr><td>André</td><td>1</td></tr></tbody></table>	Nome	Id	André	1
Nome	Id				
André	1				
2	<table border="1"><thead><tr><th>Nome</th><th>Id</th></tr></thead><tbody><tr><td>José</td><td>2</td></tr></tbody></table>	Nome	Id	José	2
Nome	Id				
José	2				

Query successful (0.038 seconds) PID=6648

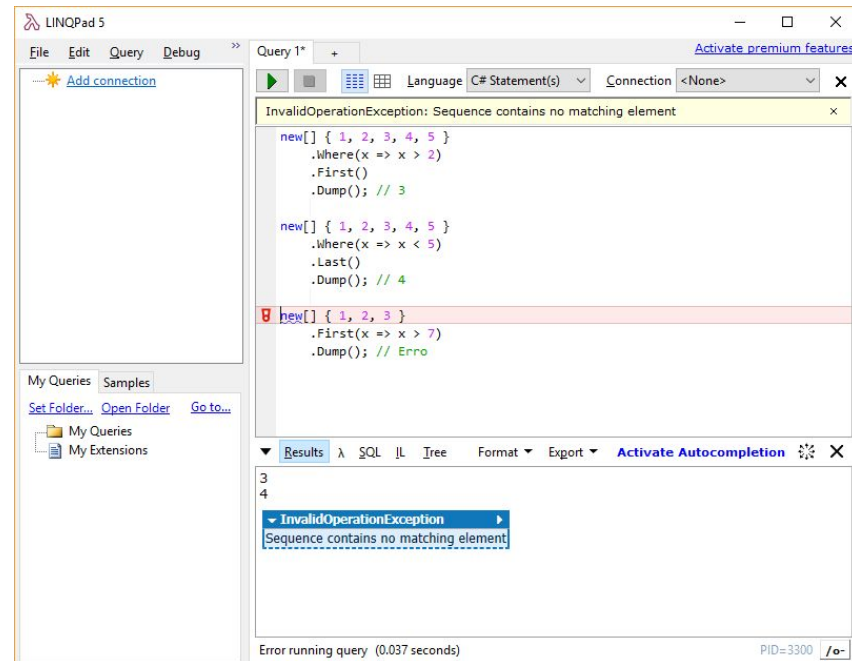
First / Last

O First e o Last servem para obter o primeiro ou o último elemento de uma sequência.

Tal como o Where, permite filtrar os elementos na sequência com uma função. Se for omitida, não é aplicada qualquer operação de filtragem.

Usar o Where seguido de First / Last é o mesmo que usar o First / Last com a condição no Where (e mais eficiente).

Se a sequência estiver vazia, ou nenhum elemento respeitar a condição da função passada por parâmetro, estas funções rebentam com um erro.



FirstOrDefault / LastOrDefault

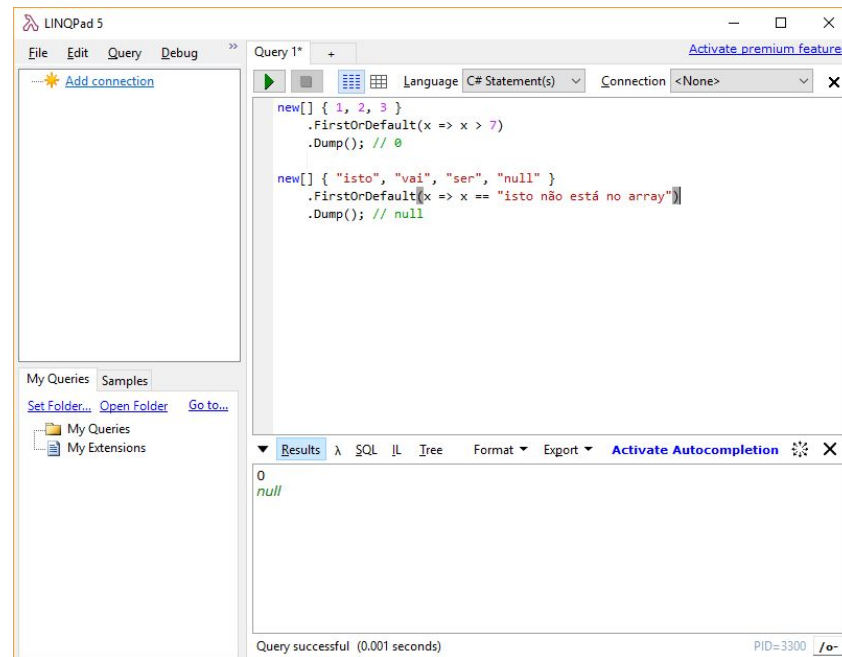
Variação do First / Last que, em vez de rebentar com um erro, devolve o valor por defeito para o tipo da sequência.

Por “valor por defeito”, depende do “T”.

Se for algo como um “int”, “double”, etc., será provavelmente 0.

Se for “string”, então é null (não uma string vazia).

Para outros objetos, é provável que seja null (depende se for class ou struct, structs não permitem nulls).



Average / Sum / Max / Min

Estas funções servem para fazer operações matemáticas sobre uma sequência de números (int, double, decimal, etc.).

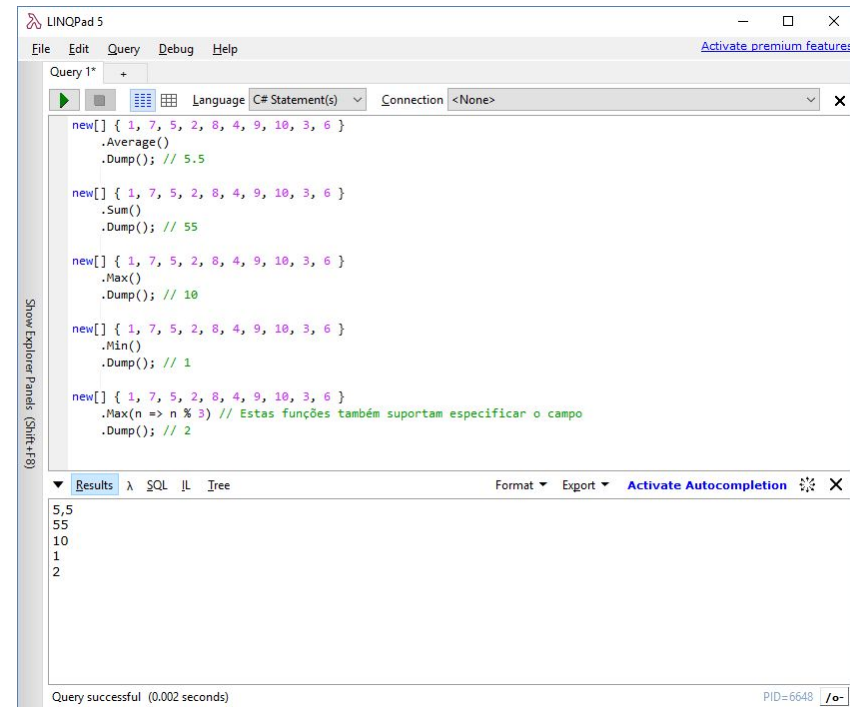
Caso as sequências não sejam numéricas, pode-se especificar uma função que permite projectar cada elemento da sequência num valor numérico (como no Select).

O Average serve para fazer médias.

O Sum para somatórios.

O Max para obter o valor máximo da sequência.

O Min para obter o valor mínimo da sequência.



```
new[] { 1, 7, 5, 2, 8, 4, 9, 10, 3, 6 }  
    .Average()  
    .Dump(); // 5.5  
  
new[] { 1, 7, 5, 2, 8, 4, 9, 10, 3, 6 }  
    .Sum()  
    .Dump(); // 55  
  
new[] { 1, 7, 5, 2, 8, 4, 9, 10, 3, 6 }  
    .Max()  
    .Dump(); // 10  
  
new[] { 1, 7, 5, 2, 8, 4, 9, 10, 3, 6 }  
    .Min()  
    .Dump(); // 1  
  
new[] { 1, 7, 5, 2, 8, 4, 9, 10, 3, 6 }  
    .Max(n => n % 3) // Estas funções também suportam especificar o campo  
    .Dump(); // 2
```

Results

5,5
55
10
1
2

Query successful (0.002 seconds) PID=6648

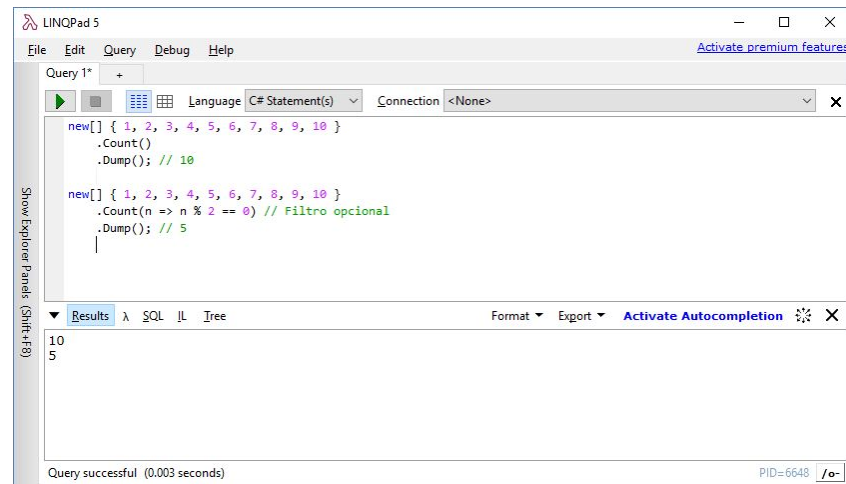
Count / Any

O Count serve quando se quer contar o nº de elementos numa sequência, e devolve um int.

O Any verifica se a sequência tem algum elemento, e devolve um booleano a indicar o resultado da condição (isto é, se o Count > 0).

Tanto o Count como o Any podem receber funções por parâmetro para filtrar elementos, tal como no Where.

Usar o Where seguido de Count / Any é o mesmo que fazer Count / Any com a condição no Where.



Operações comuns - Partição

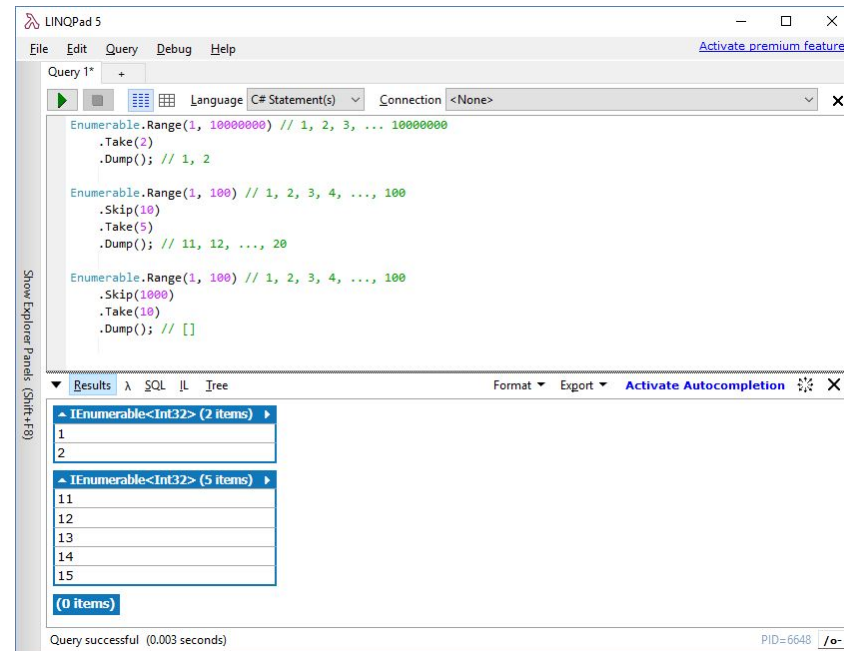
Take / Skip

O Take e o Skip permitem limitar o nº de itens numa sequência.

O Take(N) limita a sequência a N itens. Se a sequência tiver menos itens que N, então ficará apenas com esses. Se a origem estiver vazia, o resultado também será vazio.

O Skip(N) permite omitir os primeiros N items. Se a sequência tiver menos itens que N, então ficará vazia.

Se combinados pela ordem Skip > Take, podem ser usados para paginação.



The screenshot shows the LINQPad 5 interface with three queries executed. The first query uses `Enumerable.Range(1, 100000000)` and `.Take(2)`, resulting in a table with 2 items (1, 2). The second query uses `Enumerable.Range(1, 100)`, `.Skip(10)`, and `.Take(5)`, resulting in a table with 5 items (11, 12, 13, 14, 15). The third query uses `Enumerable.Range(1, 100)`, `.Skip(1000)`, and `.Take(10)`, resulting in a table with 0 items. The status bar at the bottom indicates 'Query successful (0.003 seconds)' and 'PID=6648'.

```
Enumerable.Range(1, 100000000) // 1, 2, 3, ... 100000000
    .Take(2)
    .Dump(); // 1, 2

Enumerable.Range(1, 100) // 1, 2, 3, 4, ..., 100
    .Skip(10)
    .Take(5)
    .Dump(); // 11, 12, ..., 20

Enumerable.Range(1, 100) // 1, 2, 3, 4, ..., 100
    .Skip(1000)
    .Take(10)
    .Dump(); // []
```

IEnumerable<Int32> (2 items)	
1	
2	

IEnumerable<Int32> (5 items)	
11	
12	
13	
14	
15	

IEnumerable<Int32> (0 items)	
------------------------------	--

Imutabilidade e Composição

Imutabilidade e composição

Todos os operadores de projeção, filtragem, ordenação, agrupamento e agregação do Linq *não modificam* as sequências nem os elementos das sequências sob as quais elas operam [imutabilidade].

Se pensarem nas ferramentas UNIX (cat, grep, sed, ls, sort, uniq, etc.), e no operador “|” (pipe), o Linq funciona sobre o mesmo conceito; o resultado de uma função (ex: Where) é passado para a próxima (ex: Select) [composição].

Estes dois conceitos são o fundamento para nós podermos “concatenar” várias operações, umas após as outras.

Outros operadores

SelectMany

Variação do Select, que é usada quando trabalhamos com sequências de sequências (Arrays de Arrays).

Só pode ser usado quando o resultado da função passada por parâmetro for também uma sequência.

O que o Linq vai fazer é o “flatten” dos elementos, de forma a que todos os elementos de todas as sequências sejam colocados pela ordem que surgem: os elementos da primeira sequência, depois os da segunda, etc.

O equivalente em outras linguagens é o “flatMap”.

The screenshot shows LINQPad 5 with a C# query in the editor. The query is as follows:

```

Agentes // Tabela dos Agentes
.Select( agente => agente.AgenteFKMultas)
.Take(2) // Por questões de brevidade...
.Dump(); // Array de Arrays de Multas

Agentes
.SelectMany( agente => agente.AgenteFKMultas)
.Take(2) // Por questões de brevidade...
.Dump(); // Array de Multas

```

The Results pane shows the output of the query, which is an array of arrays of multas. The first two items are expanded, showing the flattened data as a table:

ID	Infração	LocalDaMulta	ValorMulta	DataDaMulta	AgenteFK	CondutorFK	ViaturaFK	AgenteF
1	Desrespeito da obrigação de parar	Leiria	100,00	2017-02-04 00:00:00	7	8	14	AgenteF
2	Não pararmos na Passadeira de Peões	Torres Novas	50,00	2017-07-01 00:00:00	9	26	6	AgenteF
			150,00		16	34	20	

The status bar at the bottom indicates "Query successful (0.029 seconds)" and "PID=14496".

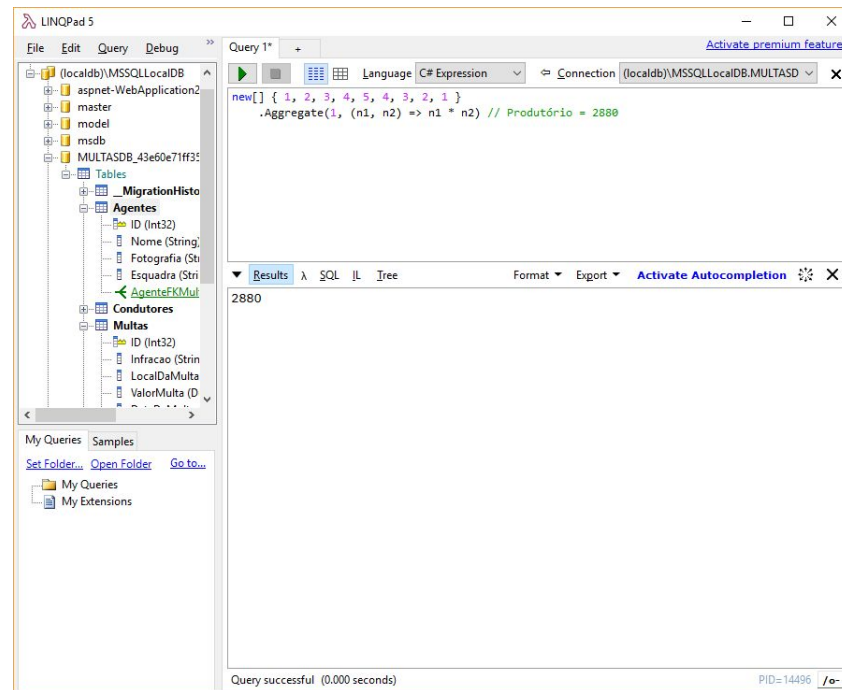
Aggregate

Função genérica para agregar uma sequência num único valor.

Recebe, por parâmetro, uma função. Esta função, ao contrário das outras, recebe dois parâmetros; cada um representa um elemento da sequência.

Opcionalmente, recebe um argumento que indica o “valor inicial”. Esse valor é colocado no início da sequência.

Esta função pode ser usada para implementar o Max, Min, Average, Sum, entre outras.

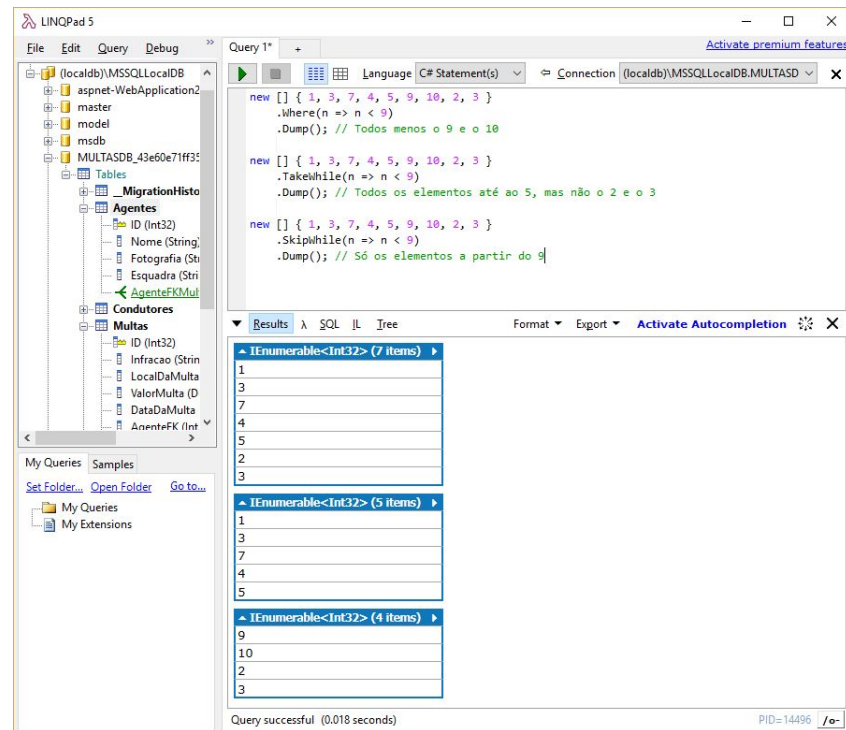


TakeWhile / SkipWhile

O TakeWhile funciona como um Where, mas deixa de produzir resultados assim que encontrar um elemento que não respeite a condição presente na função passada por parâmetro, mesmo que existam mais elementos na sequência.

O SkipWhile funciona ao contrário do TakeWhile; ignora itens até que o primeiro não respeite a condição presente na função passada por parâmetro.

Útil quando se querem obter sequências contíguas de elementos.



Intersect / Union / Except

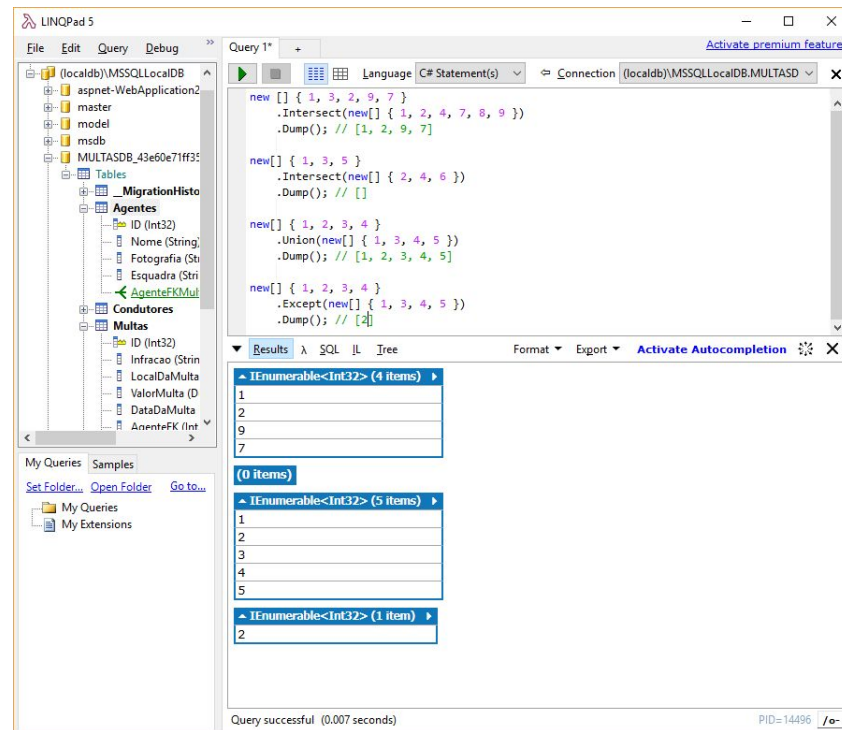
Permitem fazer a intersecção ou a união (teoria de conjuntos) entre duas sequências.

Os elementos são comparados através da sua igualdade; seja por chave, GetHashCode, ou Equals.

O Intersect produz a interseção, se nenhum elemento for comum às duas sequências, o resultado é uma sequência vazia.

O Union produz a união.

O Except exclui os elementos da primeira sequência que estejam também na segunda.



The screenshot shows the LINQPad 5 interface. On the left, a file explorer displays a project structure with a database connection to (localdb)\MSSQLLocalDB. The main editor shows a C# query titled 'Query 1*' that uses LINQ to generate a sequence of numbers from 1 to 100 and filter out non-prime numbers. The results pane at the bottom shows a table with 100 items, displaying the first 17 rows of the results.

```
// Um número, n, é primo se for maior que 1
// e se nenhum número entre 2 e n é divisor de n.
// Enumerable.Range(1, 100) dá 100 números, a partir do 1.
// Enumerable.Range(2, num - 2) dá números entre 2 e num.
Enumerable.Range(1, 100) // 1, 2, 3, ..., 100
.Select(num => new
{
    Numero = num,
    Primo = num > 1 && !Enumerable.Range(2, num - 2) // 2, ..., n
        .Any(divisor => num % divisor == 0)
})
```

Numero	Primo
1	False
2	True
3	True
4	False
5	True
6	False
7	True
8	False
9	False
10	False
11	True
12	False
13	True
14	False
15	False
16	False
17	True

Query successful (0.005 seconds) PID=14496

Calcular números primos sem ciclos for

Linq aplicado às Multas

<https://github.com/ipt-ti2-2017-2018/multas/blob/master/Multas-tA/Multas-tA/Api/MultasController.cs>

101 Linq Samples

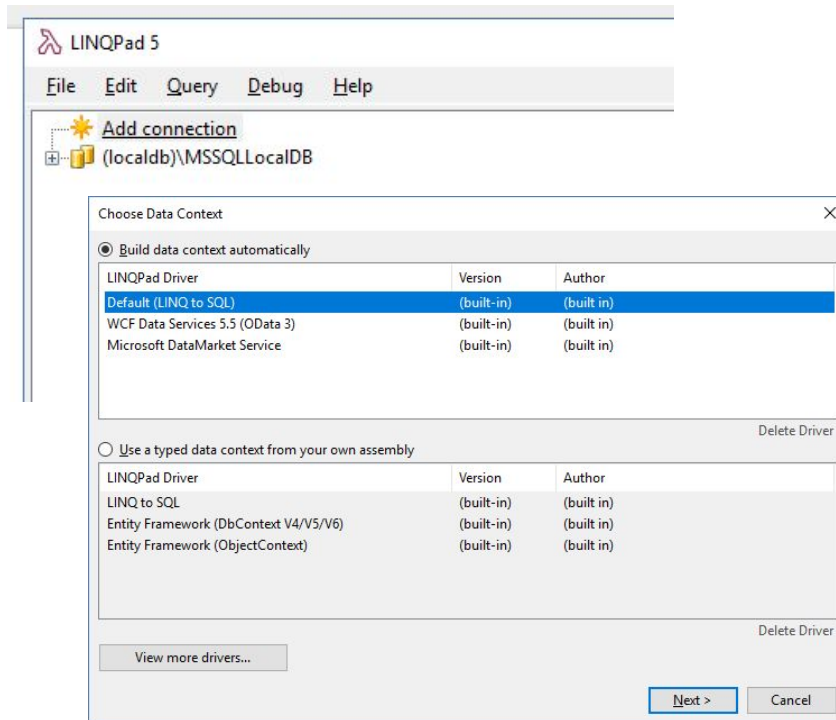
<https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>

Ligar o LinqPad a uma base de dados

O LinqPad pode ligar-se a uma base de dados através da opção “Add connection” presente no canto superior esquerdo.

Ao clicar nessa opção, aparece um pop-up com opções. O “Default (LINQ to SQL)” permite ligar a qualquer BD SQL Server.

Se se pretende usar as classes de um projeto (ex: Multas), deve-se usar a opção “Entity Framework (DbContext V4/V5/V6)”.



LinqPad e BD - Linq to SQL

A opção do Linq to SQL não requer quaisquer classes do modelo.

O que se tem que fazer é especificar a ligação ao servidor. Se estivermos a usar LocalDB, que é um SQL Server para desenvolvimento, colocamos “(localdb)\MSSQLLocalDB”, com autenticação do Windows.

Opcionalmente, podemos escolher um ficheiro para a base de dados.

Clicar em OK guarda a ligação.

The image shows the 'LINQ to SQL Connection' dialog box. It has several sections: 'Provider' with radio buttons for 'SQL Server' (selected), 'SQL CE 3.5', 'SQL CE 4.0', and 'SQL Azure'; 'Server' with a text box containing '(localdb)\MSSQLLocalDB'; 'Log on details' with radio buttons for 'Windows Authentication' (selected) and 'SQL Authentication'; 'Database' with radio buttons for 'Display all in a TreeView' (selected), 'Attach database file' (with a 'Browse' button), and 'Specify new or existing database' (with a dropdown menu); a 'Create database' button; an 'Include additional databases' checkbox; 'Data Context Options' with checkboxes for 'Display columns in alphabetical order', 'Pluralize EntitySet and Table properties' (checked), 'Capitalize property names' (checked), and 'Include Stored Procedures and Functions' (checked); a 'Remember this connection' checkbox (checked) and a 'Contains production data' checkbox (unchecked); and buttons for 'Advanced...', 'Test', 'OK' (highlighted with a blue border), and 'Cancel'.

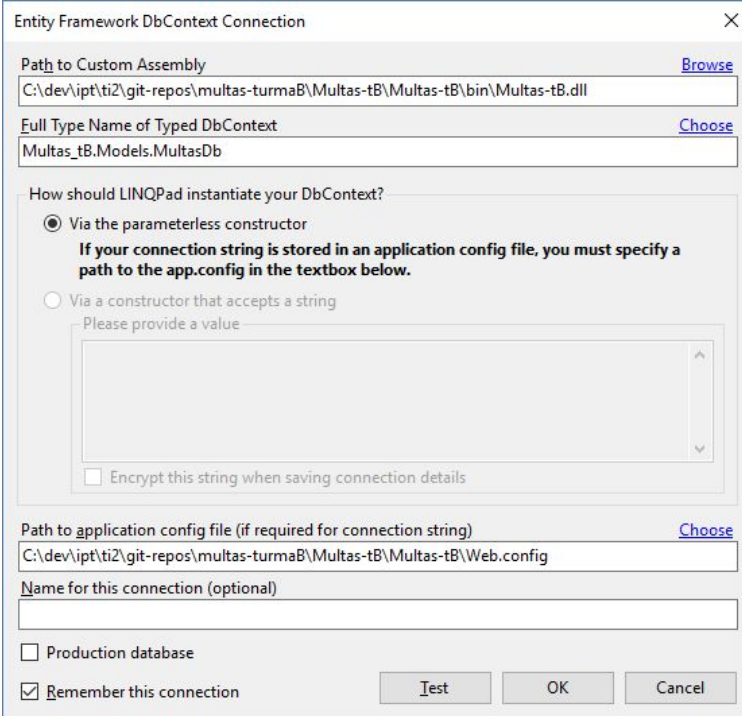
LinqPad e BD - Entity Framework

Para Entity Framework, devemos escolher a localização do DLL (após compilação) onde estão as nossas classes da base de dados, e o nome completo da classe da base de dados.

Teremos também que escolher o ficheiro de configuração com uma connection string.

Ao clicar em OK, o LinqPad guarda a ligação.

Atenção: O LinqPad por vezes não consegue usar a connection string que está no Web.config. Se isso acontecer, usem Linq to SQL.



The image shows the 'Entity Framework DbContext Connection' dialog box. It has a title bar with a close button. The dialog contains several fields and options:

- Path to Custom Assembly:** A text box containing 'C:\dev\ipt\ti2\git-repos\multas-turmaB\Multas-tB\Multas-tB\bin\Multas-tB.dll'. A 'Browse' link is to the right.
- Full Type Name of Typed DbContext:** A text box containing 'Multas_tB.Models.MultasDb'. A 'Choose' link is to the right.
- How should LINQPad instantiate your DbContext?:** Two radio buttons are present:
 - ☒ Via the parameterless constructor. Below it, a note says: 'If your connection string is stored in an application config file, you must specify a path to the app.config in the textbox below.'
 - ☐ Via a constructor that accepts a string. Below it, a text box labeled 'Please provide a value' is empty. A checkbox 'Encrypt this string when saving connection details' is below the text box.
- Path to application config file (if required for connection string):** A text box containing 'C:\dev\ipt\ti2\git-repos\multas-turmaB\Multas-tB\Multas-tB\Web.config'. A 'Choose' link is to the right.
- Name for this connection (optional):** An empty text box.
- ☐ Production database
- ☒ Remember this connection
- Buttons: 'Test', 'OK', and 'Cancel'.

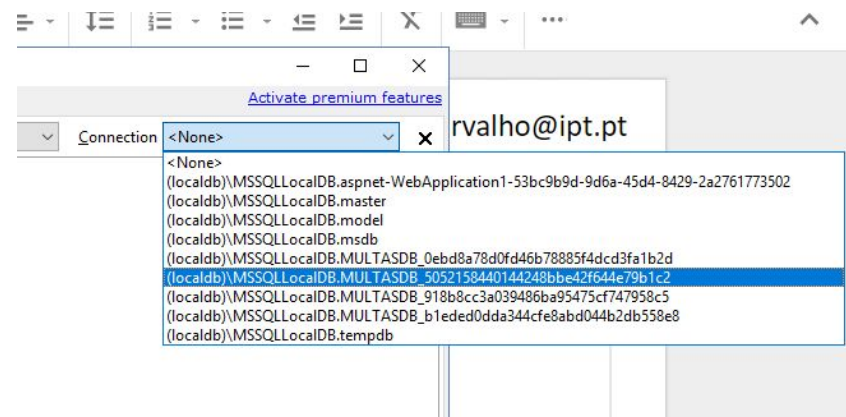
LinqPad e BD - Escolher ligação

Para escolher a ligação, usem a drop-down presente no canto superior direito.

Ao seleccionar a base de dados, os nomes das tabelas estarão disponíveis como variáveis.

Atenção que os nomes das colunas poderão não ser iguais aos nomes escolhidos pela Entity Framework, devem olhar para os campos da tabela no canto superior esquerdo para mais informações.

(Ver slide seguinte)



Query à BD das Multas, tabela dos Agentes

The screenshot shows the LINQPad 5 interface. On the left, a file explorer displays a project structure with folders 'model' and 'msdb'. Under 'msdb', there are two subfolders: 'MULTASDB_0ebd8a78d0fd46b78885f4dcd3fa1b2d' and 'MULTASDB_5052158440144248bbe42f644e79b1c2'. The 'Tables' folder is expanded, showing 'MigrationHistories', 'Agentes', 'Condutores', 'Multas', and 'Viaturas'. The 'Agentes' table is selected, showing its schema: ID (Int32), Nome (String), Fotografia (String), Esquadra (String), and AgenteFKMultas.

The main query area shows the query: `Agentes.Take(100)`. The results are displayed in a table with 10 items:

ID	Nome	Fotografia	Esquadra	AgenteFKMultas
1	Tânia Vieira	TaniaVieira.jpg	Ourém	AgenteFKMultas
2	António Rocha	AntonioRocha.jpg	Ourém	AgenteFKMultas
3	André Silveira	AndreSilveira.jpg	Abrantes	AgenteFKMultas
4	Lurdes Vieira	LurdesVieira.jpg	Leiria	AgenteFKMultas
5	Cláudia Pinto	ClaudiaPinto.jpg	Porto	AgenteFKMultas
6	Rui Vieira	RuiVieira.jpg	Tomar	AgenteFKMultas
7	Paulo Vieira	PauloVieira.jpg	Torres Novas	AgenteFKMultas
8	Augusto Carvalho	AugustoCarvalho.jpg	Lisboa	AgenteFKMultas
9	Beatriz Pinto	BeatrizPinto.jpg	Porto	AgenteFKMultas
10	José Alves	JoseAlves.jpg	Alcanena	AgenteFKMultas

The status bar at the bottom indicates 'Query successful (1.003 seconds)' and 'PID=17400'.