

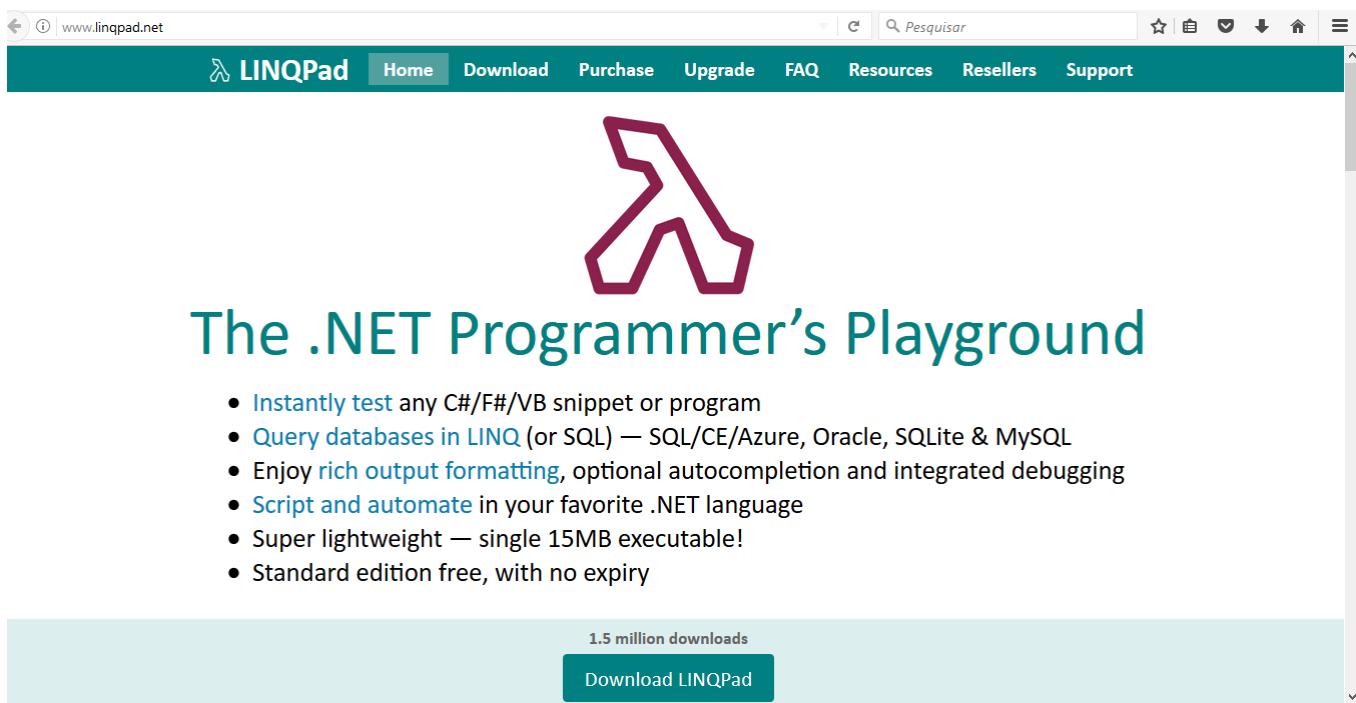
01
1 - Linqpad

Transcrição

Ao longo do curso trabalhamos com consultas para o site da **Alura Tunes**, mas ele já não é mais o mesmo e cresceu! Algo que dificulta um pouco nossa vida diante desse novo contexto é ter que rodá-lo diversas vezes para verificar as consultas realizadas.

Nesta aula, veremos uma nova ferramenta para abrir e rodar as consultas de forma eficiente e rápida obtendo os resultados de maneira mais acessível!

Essa ferramenta é o **LINQPad**, disponível para [download gratuito \(<https://www.linqpad.net/>\)](https://www.linqpad.net/):



The .NET Programmer's Playground

- Instantly test any C#/F#/VB snippet or program
- Query databases in LINQ (or SQL) — SQL/CE/Azure, Oracle, SQLite & MySQL
- Enjoy rich output formatting, optional autocompletion and integrated debugging
- Script and automate in your favorite .NET language
- Super lightweight — single 15MB executable!
- Standard edition free, with no expiry

1.5 million downloads

Download LINQPad

Para baixar a ferramenta basta clicar no botão "Download LINQPad" e optar pela versão de número 5:



Download LINQPad

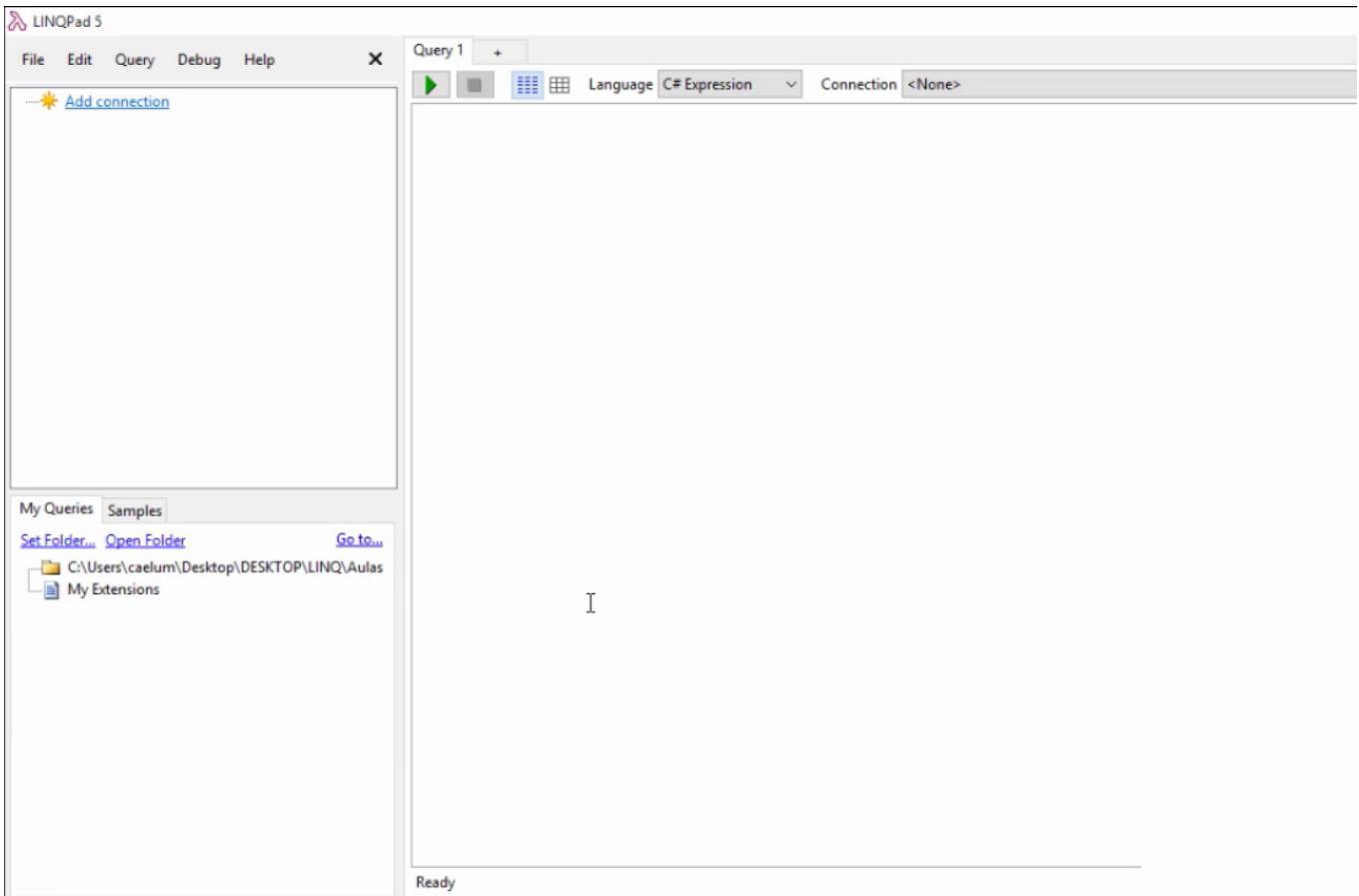
Download LINQPad 5 for .NET Framework 4.6

Download LINQPad 4 for .NET Framework 4.0 / 4.5

LINQPad 5 runs **side-by-side** with LINQPad 4.

If you have a paid edition of LINQPad 4, please read the [licensing notes](#).

O programa será baixado e instalado. Ao abri-lo, iremos nos deparar com a seguinte interface:



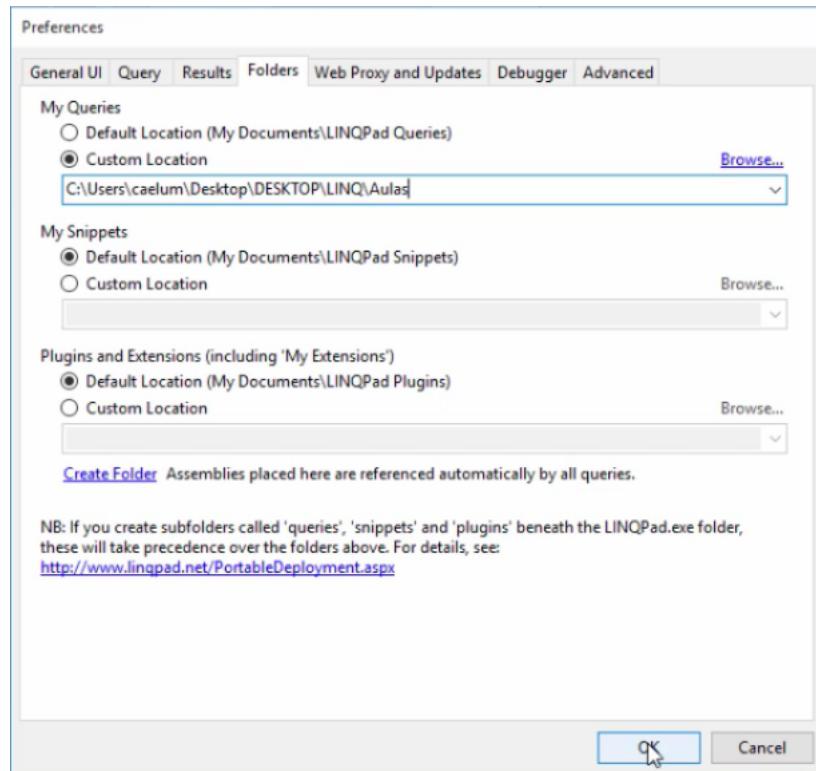
O programa permite tanto criar novas consultas quanto trabalhar com as já existentes. Para testar o programa, criaremos uma consulta simples que trará números de 1 a 10. Na parte de cima da tela é mostrado o código e, na de baixo, o resultado:

```
from n in new int[] {1,2,3,4,5,6,7,8,9,10}
select n
```

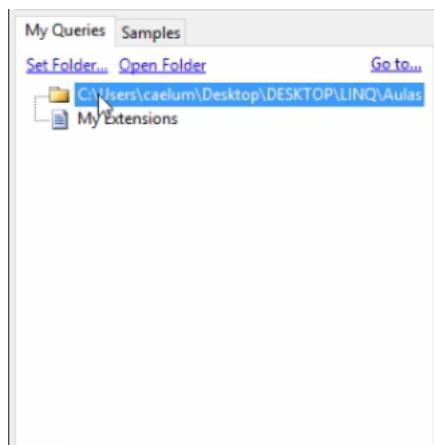
IEnumerable<Int32> (10 items)
1
2
3
4
5
6
7
8
9
10

No **Visual Studio** não será exibido nem o tipo do dado nem o `IEnumerable`, mas no **LINQPad** sim.

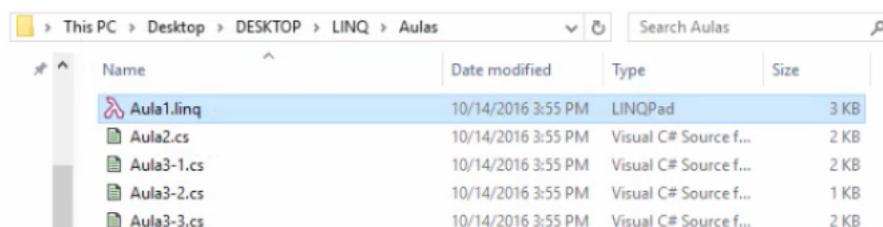
Para utilizar o `LINQPad`, vamos abrir as consultas feitas. Como salvamos uma cópia desses arquivos, vamos definir no "Set Folder" o caminho para acessar os documentos que desejamos:



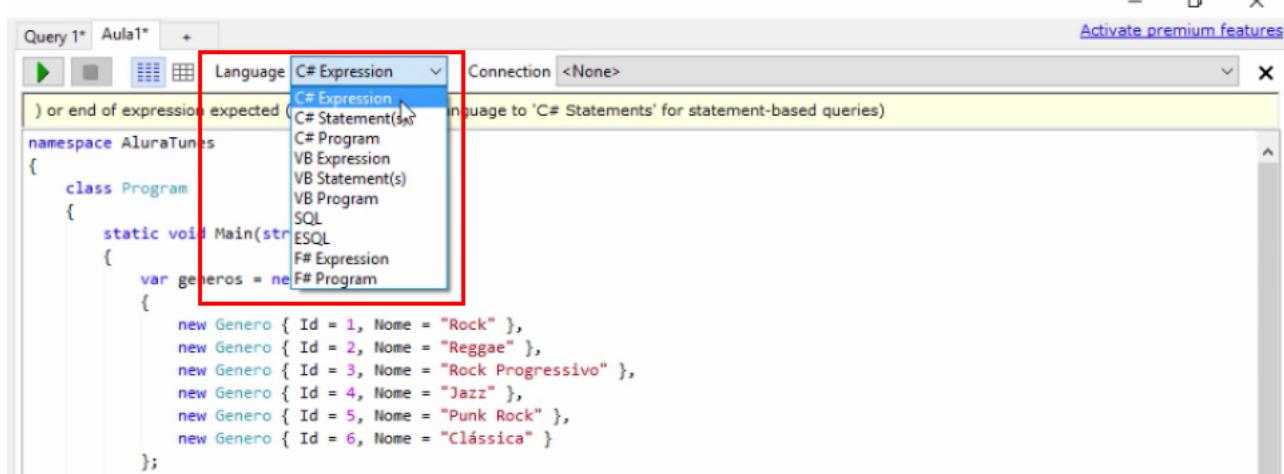
Ao fazermos isso, o resultado será uma pasta vazia e sem arquivos ou consultas:



Isso ocorre, pois os arquivos estão com uma extensão de tipo `cs`. Para solucionar esse impasse é preciso modificar a extensão dos demais documentos para que eles sejam `linq`. Vamos alterar o nome do arquivo para que apareça no LINQPad :



Mas, ainda não conseguiremos rodar a consulta `Aula 1`, pois ocorre um erro! Isso acontece uma vez que o **LINQPad** não comprehende a sintaxe da consulta, portanto, é preciso alterar a linguagem para rodar o arquivo. Para fazer isso modificaremos o campo `Language` para `C# Program` :

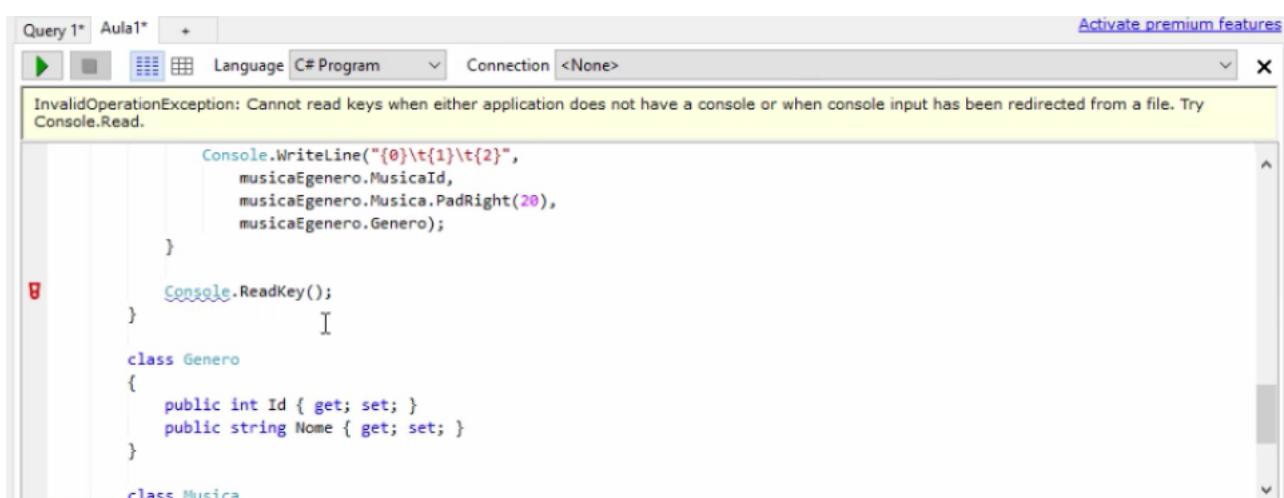


```

Query 1* Aula1* + Language C# Expression <None> Activate premium features
) or end of expression expected
namespace AluraTunes
{
    class Program
    {
        static void Main(string[] args)
        {
            var generos = new List<Genero>
            {
                new Genero { Id = 1, Nome = "Rock" },
                new Genero { Id = 2, Nome = "Reggae" },
                new Genero { Id = 3, Nome = "Rock Progressivo" },
                new Genero { Id = 4, Nome = "Jazz" },
                new Genero { Id = 5, Nome = "Punk Rock" },
                new Genero { Id = 6, Nome = "Clássica" }
            };
        }
    }
}

```

Mesmo fazendo a mudança, seguirá ocorrendo um erro de sintaxe. A falha acontece pois é preciso modificar o código para que o **LINQPad** possa compreendê-lo. Dessa forma, o primeiro passo é remover o `namespace AluraTunes`. Após ser feito isso, conseguiremos rodar:



```

Query 1* Aula1* + Language C# Program <None> Activate premium features
InvalidOperationException: Cannot read keys when either application does not have a console or when console input has been redirected from a file. Try Console.Read.
    Console.WriteLine("{0}\t{1}\t{2}",
        musicaEgenero.MusicaId,
        musicaEgenero.Musica.PadRight(20),
        musicaEgenero.Genero);
    }

    Console.ReadKey();
}

class Genero
{
    public int Id { get; set; }
    public string Nome { get; set; }
}

class Musica

```

Entretanto, aparece um alerta de que o `ReadKey()` não faz sentido! Assim, vamos deixar o `Console.ReadKey()` comentado:

```
//           Console.ReadKey()
```

Agora, ao rodarmos a aplicação, teremos o seguinte resultado:



Genero	MusicaId
Rock	1
Rock Progressivo	3
Punk Rock	5
Sweet Child O'Mine	1
I Shot The Sheriff	2
Danúbio Azul	3

O resultado é exatamente igual ao que foi obtido rodando no **Visual Studio**!

Vamos melhorar o resultado mostrando-o no formato de uma grade, em vez de um texto solto e livre!

O `foreach()` utilizado no código é o responsável pelo resultado visual de um texto. Vamos deixar essa parte comentada e, no lugar, utilizaremos um método chamado `Dump()` que serve para visualizar o conteúdo na forma de tabela. Escreveremos `query.Dump()` acima do `List<Musica> musicas = new List <Musica> :`

![]mostrando como fica o resultado utilizando o dumb](https://s3.amazonaws.com/caelum-online-public/3_LINQ/1_51+mostrando+como+fica+o+resultado+utilizando+o+Dumb.png (https://s3.amazonaws.com/caelum-online-public/3_LINQ/1_51+mostrando+como+fica+o+resultado+utilizando+o+Dumb.png))

O resultado mostra uma tabela colapsável. Mas, parte do código ainda é mostrado em forma de texto! Para que ambos os resultados sejam mostrados no formato de tabela é preciso comentar a linha: `foreach (var musicaEgenero in queryMusicas)`. Para fazer isso, selecionaremos parte do código e depois, vamos utilizar o "Ctrl + K + C". Abaixo disso nós adicionamos o `queryMusicas.Dump()`. O resultado será:

No começo do curso, vimos que é possível fazer consultas LINQs usando duas sintaxes distintas: a primeira sintaxe é de consulta e a segunda de método, nesse caso fizemos uma consulta com sintaxe de consulta. Como poderemos transformar a consulta para trazer a sintaxe de método?

O LINQPad transforma a sintaxe de método em uma sintaxe de consulta automaticamente, basta clicar no botão da expressão lambda:

▼ Results **λ** SQL JL Tree

Mas, ao clicarmos no símbolo, o resultado será uma janela vazia. Isto ocorrerá, pois ele exige uma preparação da consulta para que ele possa trazer o resultado usando uma sintaxe de método. Assim, é preciso modificar a origem da dados e colocar um método chamado `AsQueryable()`. Dessa forma, vamos escrever junto do `from m in musicas` `AsQueryable()`:

```
var queryMusicas
= from m in musicas.AsQueryable()
join g in generos on m.GeneroId equals g.Id
select new {
    MusicaId = m.id,
    Musica = m.Nome,
    Genero = g.Nome
};
```

Agora, ao clicar no sinal de `lambda`, teremos o seguinte resultado:

▼ Results SQL JL Tree

```
System.Collections.Generic.List`1[UserQuery+Program+Musica]
  .Join (
    System.Collections.Generic.List`1[UserQuery+Program+Genero],
    m => m.GeneroId,
    g => g.Id,
    (m, g) =>
    new
    {
      MusicaId = m.Id,
      Musica = m.Nome,
      Genero = g.Nome
    }
  )
```

Query successful (0.008 seconds)

Por último, no `select` , teremos as propriedades que estão inseridas na consulta.