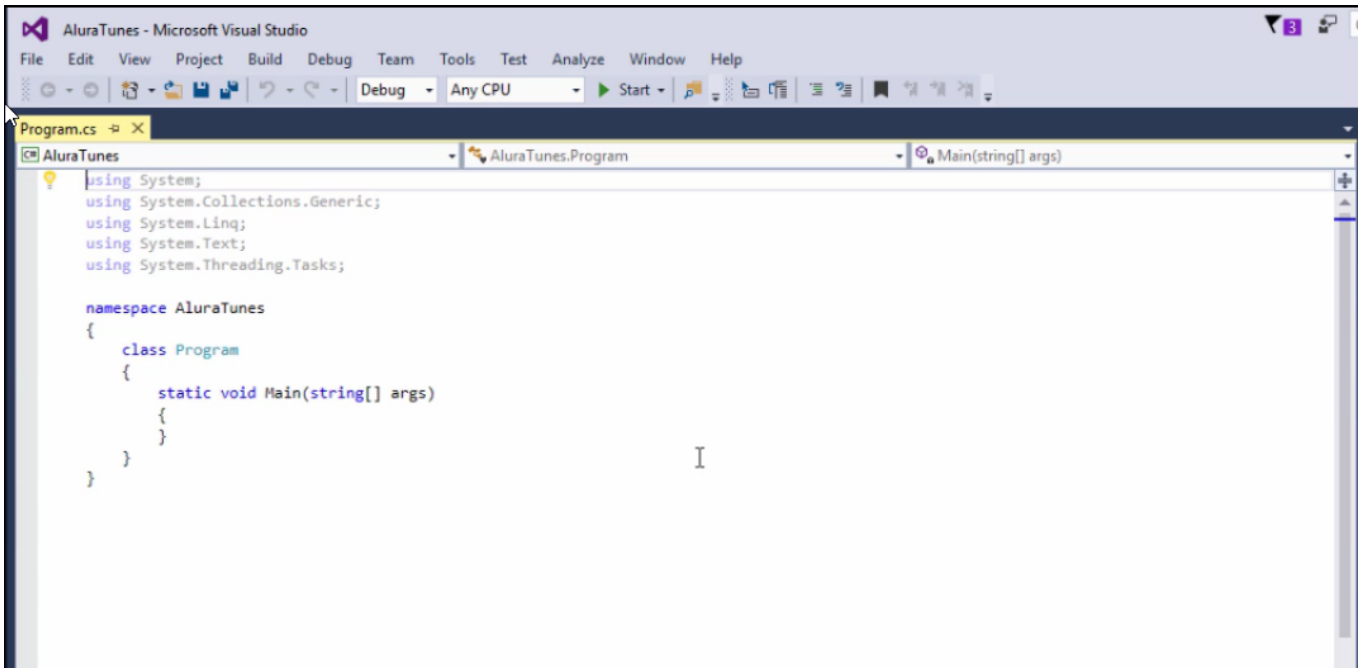
02

1- Introdução ao Linq com select from e where

Transcrição

Para iniciar este curso, primeiro vamos abrir o **Microsoft Visual Studio**. Criaremos nele um novo projeto em "File > New > Project". O tipo do projeto será `ConsoleApplication` e nele colocaremos o nome "AluraTunes" clicando em "Ok". Desta maneira, teremos o projeto criado:



O primeiro pedido do cliente foi: **listar todos os gêneros que possuem a palavra `rock`.**

Para começar a criar essa consulta é preciso de uma classe que represente gênero, por isso, adicionamos o `class Genero` que deve conter `id` e `nome`. Portanto, vinculado a `class Genero` colocamos uma propriedade `Id`, a `public int Id { get; set; }`, e outra propriedade para `Nome`, a `public string Nome { get; set; }`:

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

Vamos acrescentar uma lista contendo diferentes gêneros. Adicionamos a variável `var generos = new List<Genero>` e dentro dela para cada estilo adicionamos um `new Genero`, o `Id` conforme a posição do estilo e `Nome` conforme o estilo. Por exemplo, `new Genero { Id = 1, Nome = "Rock"}`:

```
class Program
{
    static void Main(string[] args)
    {
        //listar os gêneros rock
        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 =  "Punk"},
                new Genero { Id = 5, Nome =  "Clássica"}
            });


        }
    }
```

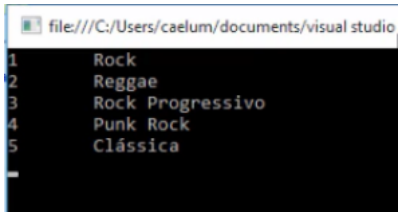
Agora, vamos imprimir os gêneros no `Console` e para isso utilizamos um laço. Portanto, logo abaixo do último elemento da lista de gêneros, nós adicionaremos o `foreach (var genero In genero)` e `Console.WriteLine(genero.Id, genero.Nome)`. No `Console.WriteLine`, o primeiro parâmetro deve ser uma `String` composta por `{0}`, `{1}` e `\t` que serve para tabulação entre um dado e outro. Teremos:

```
foreach (var genero in generos)
{
Console.WriteLine("{0}\t{1}", genero.Id, genero.Nome);
}
```

Ao rodarmos o código, a lista aparecerá e sumirá logo em seguida! Isso ocorre pois falta uma instrução de que o `Console` deve aguardar uma interação do usuário. Portanto, adicionaremos o `Console.ReadKey()` abaixo do `WriteLine`:

```
foreach (var genero in generos)
{
Console.WriteLine("{0}\t{1}", genero.Id, genero.Nome);
}
Console.ReadKey();
```

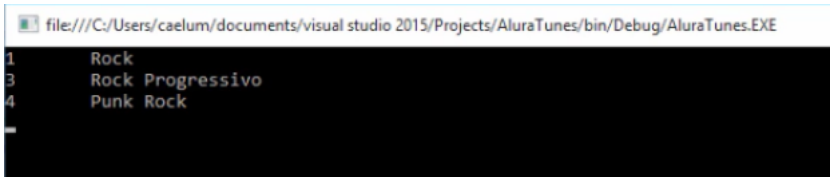
Ao acrescentar isso a lista mostrará no `Console` todos os gêneros:



O que faremos é criar um filtro para trazer apenas o gênero `rock`. Portanto, adicionaremos a condição usando o `if()` acompanhado de `(genero.Nome.Contains("Rock"))`:

```
foreach (var genero in generos)
{
    if (genero.Nome.Contains("Rock"))
    {
    Console.WriteLine("{0}\t{1}", genero.Id, genero.Nome);
    }
}
```

Logo, ao rodar esse código temos:



Será que poderíamos fazer a mesma consulta, mas de uma maneira diferente? Vamos testar uma outra forma que será mais parecida ao que fazemos no `SQL` !

Felizmente a **Microsoft** criou um tipo de componente que nos auxilia a fazer esse tipo de consulta, o `select * from` . Junto disso, adicionaremos o `C#` que também possui uma sintaxe própria, portanto, a consulta deve iniciar por `from` . Depois, vamos inserir a variável `g` . Teremos `from g in generos` e abaixo disso `select g` :

```
from generos in generos
select g;
```

Falta colocar uma variável que armazene a definição de consulta! Assim, vamos escrever `var query = from g in generos` :

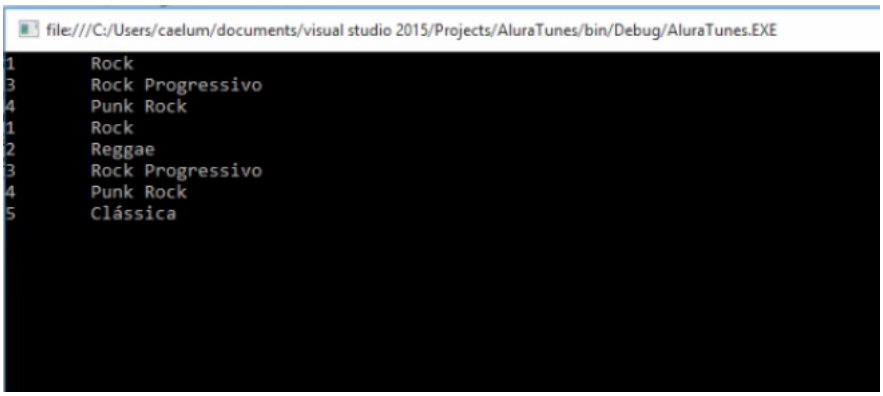
```
var query = from g in generos
select g;
```

Agora que a definição de consulta está pronta, vamos listar os dados! Para isto, utilizaremos o laço `foreach()` e especificaremos que para cada gênero dentro da consulta deve ser impressa informações, portanto, basta copiar a linha do `Console.WriteLine` :

```
var query = from g in generos
select g;

foreach (var genero in query);
{
    Console.WriteLine("{0}\t{1}", genero.Id, genero.Nome);
}
```

Ao rodar a consulta temos:



Agora falta inserir um espaço entre uma listagem e outra, assim, podemos adicionar `Console.WriteLine()` :

```
var query = from g in generos
select g;
```

```
Console.WriteLine();

foreach (var genero in query);
```

Rodando isso temos:

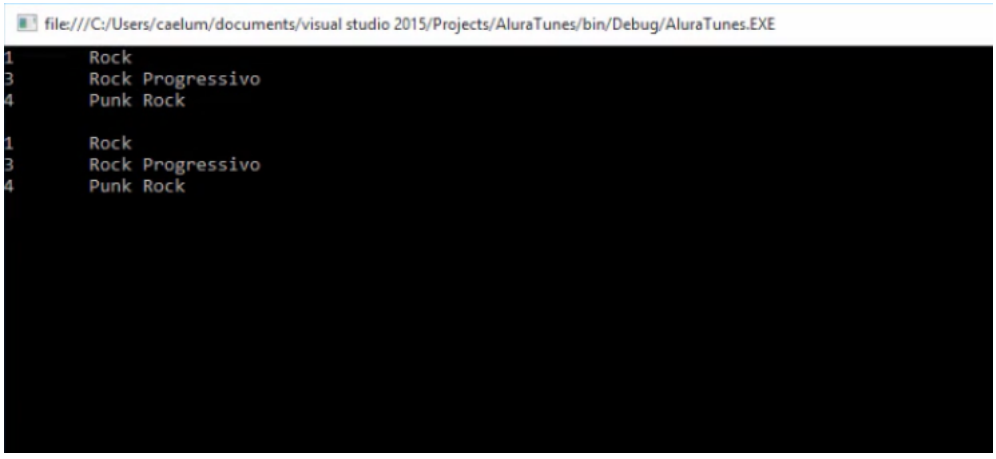
Agora temos uma listagem que utiliza o método antigo e outro, o novo! Mas, ainda falta um filtro para a segunda consulta, portanto, vamos adicioná-lo! A única diferença em relação a uma consulta no `SQL` é o `Contains`, assim, escrevemos `where g.Nome.Contains()` e dentro disso, passamos aquilo que deve ser filtrado: o `Rock`:

```
// select * from generos

var query = from g in generos
            where g.Nome.Contains("Rock")
            select g;

Console.WriteLine();
foreach (var genero in query)
```

Rodando isso temos o seguinte:



Finalmente, ambas as consultas possuem o mesmo resultado!

A consulta na qual utilizamos uma sintaxe similar a `SQL` só é possível devido a um componente que a **Microsoft** criou que é o **LINQ - Language Integrated Query** ou **Consulta Integrada a Linguagem** e essa linguagem é a `.NET`, `C#` ou Visual Basic. O **LINQ** é muito interessante, pois podemos utilizá-lo com base no conhecimento em consultas `SQL` e aplicá-lo em uma linguagem `.NET`.

Imagine se tivéssemos diversas condicionais, não só o `if` que filtra a palavra `Rock`, mas inúmeros `if`s dentro de diversos `foreach`. Nesse caso a consulta se tornaria extremamente complexa! Seria difícil entender o objetivo da consulta! Utilizando o **LINQ** e, baseado na vivência `SQL`, já é possível compreender melhor a intenção do código!