01

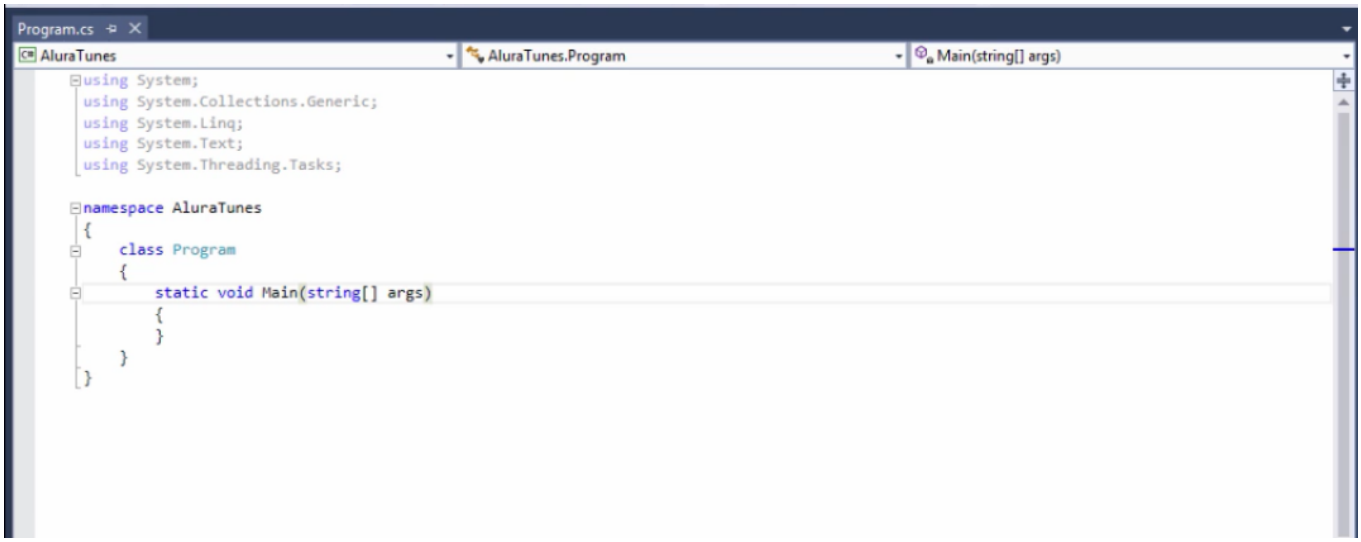
1 - Linq to entities contexto join take log sql

Transcrição

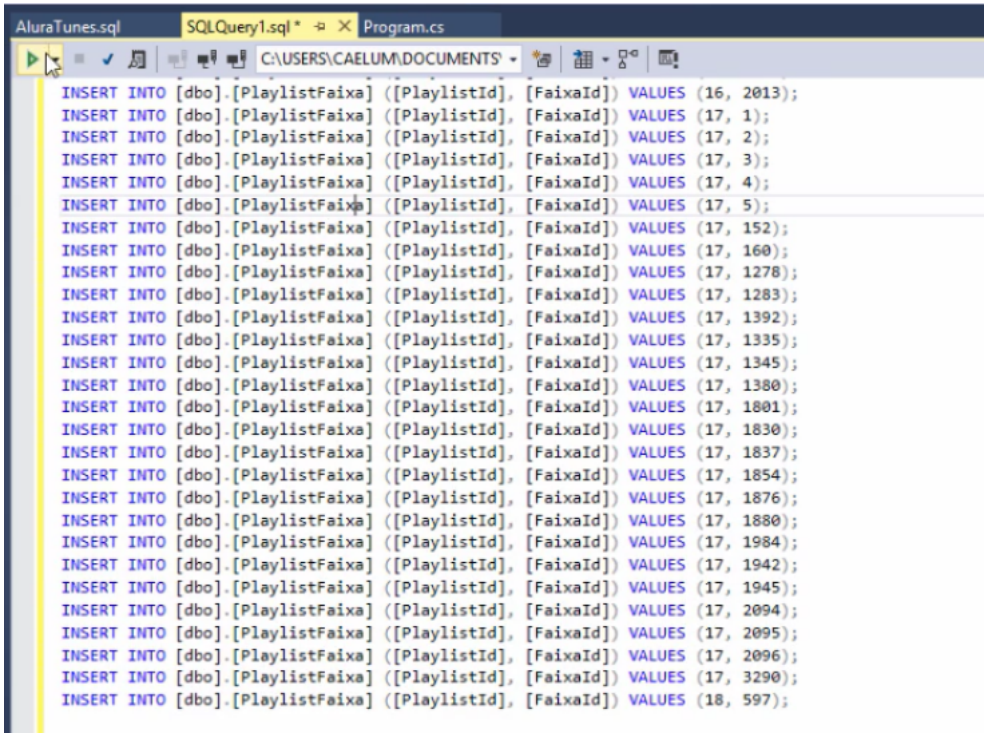
Na última aula, vimos como utilizar o `LINQ` para realizar consultas de objetos em memória e arquivos `xml`.

A tarefa desta aula é criar uma nova requisição, a pedido do cliente, para acessar o banco de dados a fim de realizar consultas. Para começar não vamos utilizar o banco de dados, mas sim a cópia do script SQL do banco de dados!

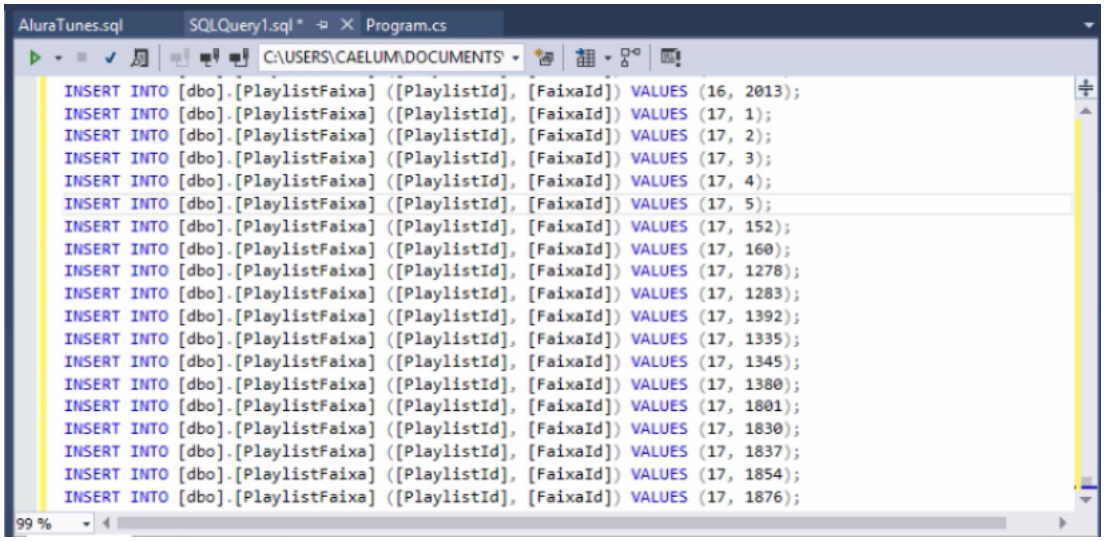
O arquivo `AluraTunes.sql` de tipo `script` possui toda a definição e informações do banco de dados da `AluraTunes`. O que faremos é criar um banco de dados vazio e rodar o `script` para criar uma cópia do banco de dados. Assim, dentro de `Data` vamos em " Add > New Item > Pasta Data > Service - based Database" e nomearemos o novo documento de `AluraTunes.mdf` e clicamos em "Add":



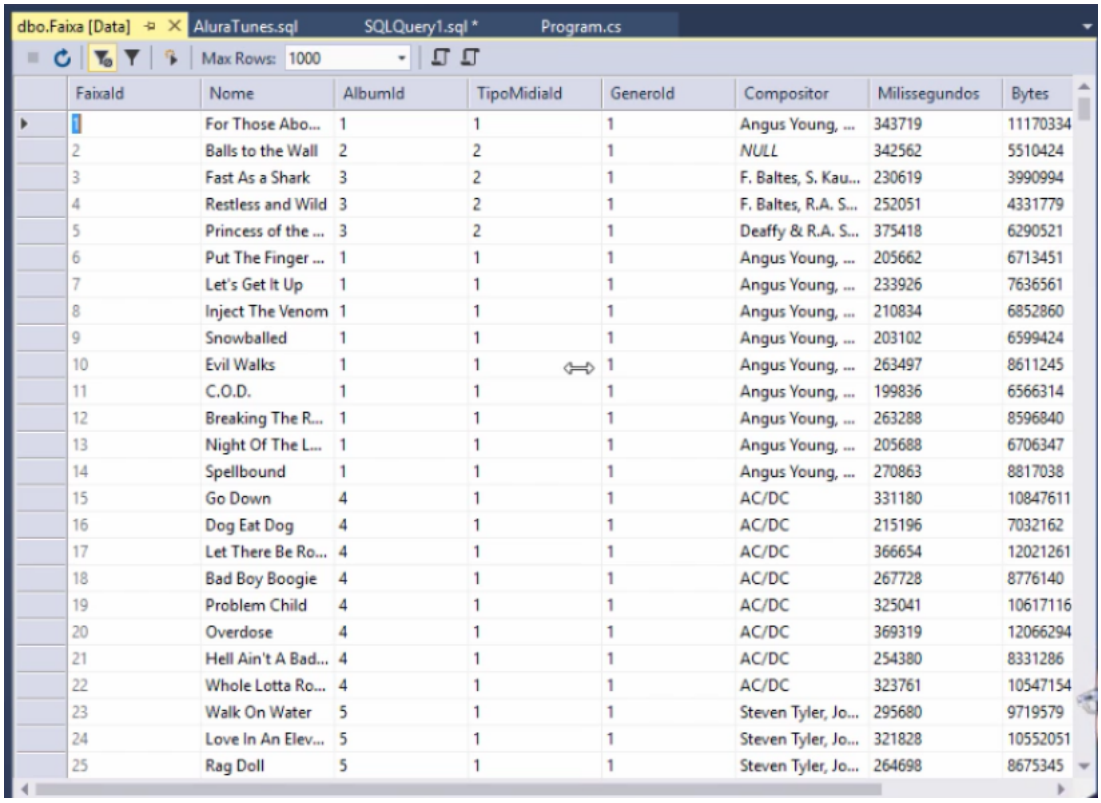
Com isso, o `AluraTunes.mdf` - um banco de dados `SQL` vazio - será criado! Normalmente, não seria possível colocar um banco de dados dentro de um projeto `C#`, mas utilizamos este recurso por conveniência e facilidade. Falta rodar o `script` no banco de dados, para que ele se torne efetivamente uma cópia do que já existe em produção. Assim, vamos abrir o `AluraTunes.mdf` e clicando em cima dele com o botão direito selecionamos "Open" e abrimos a pasta "Tables". Nela verificamos que não existe nenhuma tabela criada! Em cima do arquivo `AluraTunes.mdf`, clicamos com o botão direito e selecionamos "New query" para criar uma nova consulta que de início estará vazia! Vamos abrir o `AluraTunes.sql` que é o script que o cliente gerou e copiá-lo para dentro da `query` que foi criada. Nessa `query` rodaremos o `script`:



Com o `script` executado, todos os objetos necessários serão criados! No fim da execução, entraremos em `AluraTunes.mdf` e vamos dar um "Refresh" para verificar se efetivamente as tabelas foram copiadas:



De inicio, trabalharemos com as tabelas `Faixa` e `Genero`. Primeiro, abriremos a `Faixa` clicando com o botão direito do mouse e selecionando "Show Table Data".

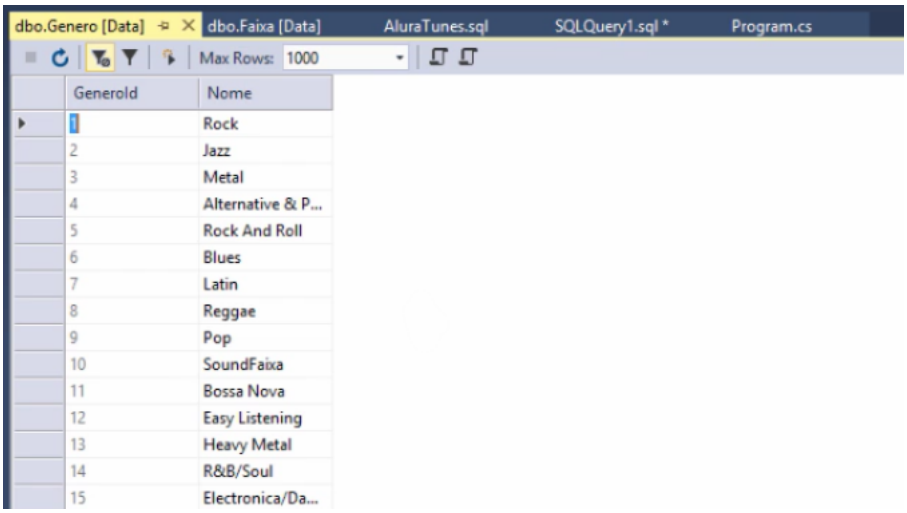


dbo.Faixa [Data] | AluraTunes.sql | SQLQuery1.sql * | Program.cs

Max Rows: 1000

FaixaId	Nome	AlbumId	TipoMidiaId	GeneroId	Compositor	Milissegundos	Bytes
1	For Those Abo...	1	1	1	Angus Young, ...	343719	11170334
2	Balls to the Wall	2	2	1	NULL	342562	5510424
3	Fast As a Shark	3	2	1	F. Baltes, S. Kau...	230619	3990994
4	Restless and Wild	3	2	1	F. Baltes, R.A. S...	252051	4331779
5	Princess of the ...	3	2	1	Deaffy & R.A. S...	375418	6290521
6	Put The Finger ...	1	1	1	Angus Young, ...	205662	6713451
7	Let's Get It Up	1	1	1	Angus Young, ...	233926	7636561
8	Inject The Venom	1	1	1	Angus Young, ...	210834	6852860
9	Snowballed	1	1	1	Angus Young, ...	203102	6599424
10	Evil Walks	1	1	1	Angus Young, ...	263497	8611245
11	C.O.D.	1	1	1	Angus Young, ...	199836	6566314
12	Breaking The R...	1	1	1	Angus Young, ...	263288	8596840
13	Night Of The L...	1	1	1	Angus Young, ...	205688	6706347
14	Spellbound	1	1	1	Angus Young, ...	270863	8817038
15	Go Down	4	1	1	AC/DC	331180	10847611
16	Dog Eat Dog	4	1	1	AC/DC	215196	7032162
17	Let There Be Ro...	4	1	1	AC/DC	366654	12021261
18	Bad Boy Boogie	4	1	1	AC/DC	267728	8776140
19	Problem Child	4	1	1	AC/DC	325041	10617116
20	Overdose	4	1	1	AC/DC	369319	12066294
21	Hell Ain't A Bad...	4	1	1	AC/DC	254380	8331286
22	Whole Lotta Ro...	4	1	1	AC/DC	323761	10547154
23	Walk On Water	5	1	1	Steven Tyler, Jo...	295680	9719579
24	Love In An Elev...	5	1	1	Steven Tyler, Jo...	321828	10552051
25	Rag Doll	5	1	1	Steven Tyler, Jo...	264698	8675345

Vamos abrir `Generos` também:

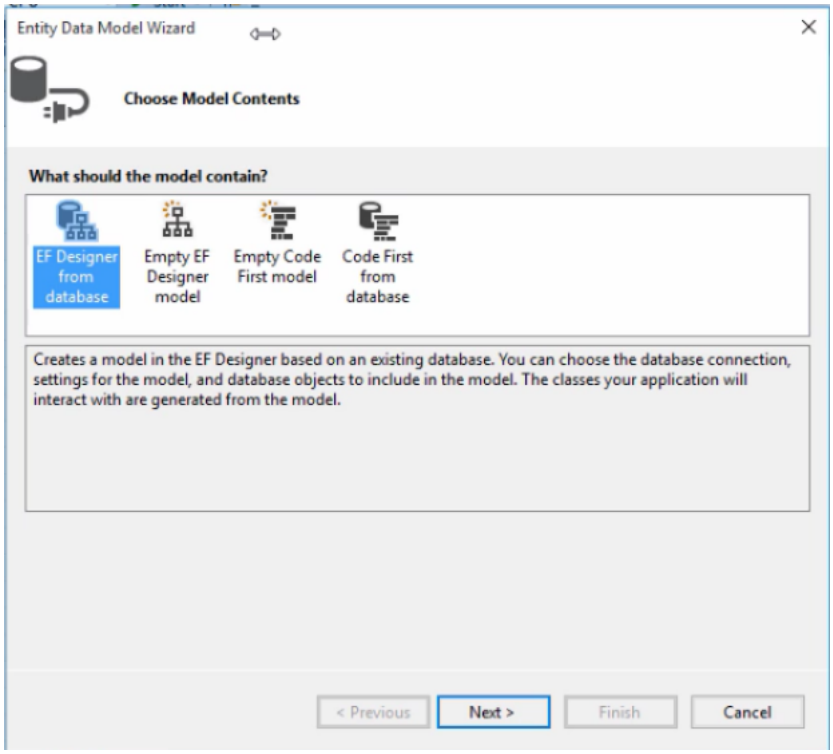


dbo.Genero [Data] | dbo.Faixa [Data] | AluraTunes.sql | SQLQuery1.sql * | Program.cs

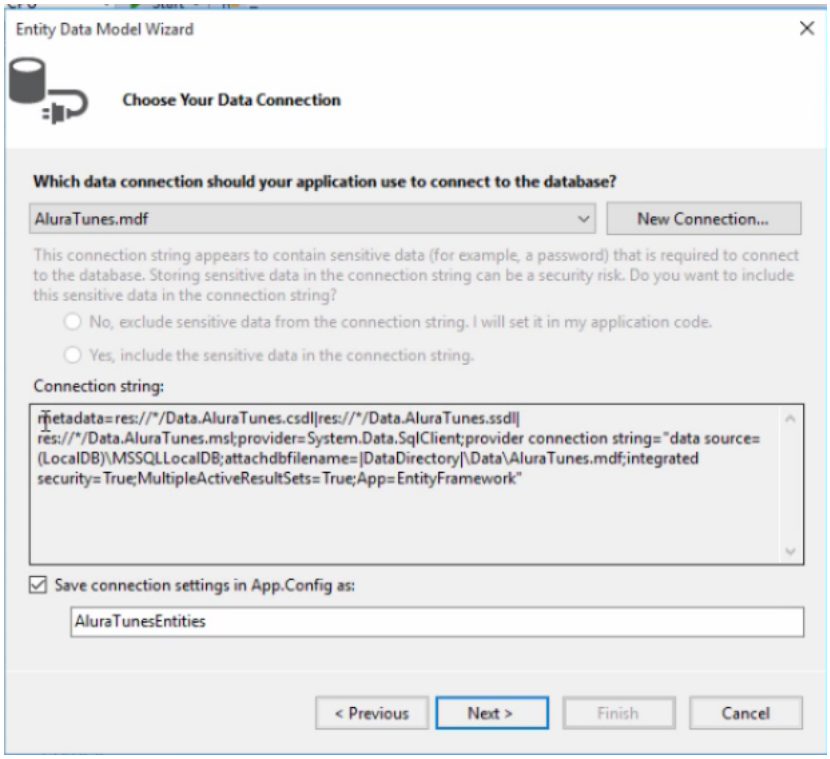
Max Rows: 1000

GeneroId	Nome
1	Rock
2	Jazz
3	Metal
4	Alternative & P...
5	Rock And Roll
6	Blues
7	Latin
8	Reggae
9	Pop
10	SoundFaixa
11	Bossa Nova
12	Easy Listening
13	Heavy Metal
14	R&B/Soul
15	Electronica/Da...

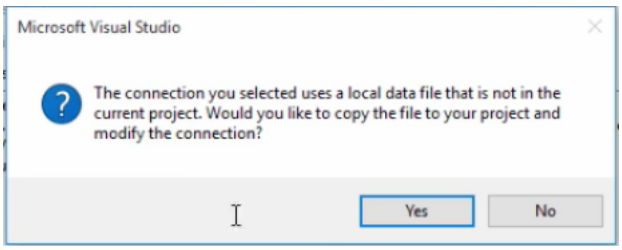
O que faremos é criar o modelo `EntityFrame` a partir do banco de dados `SQL` . Para fazer isto, vamos em `Data` e "Add > New Item... ", selecionaremos o `ADD.NET Entity Data Model` e nomearemos o modelo como `AluraTunes` . Depois, vamos clicar em "Add". Assim, abrirá a seguinte janela:



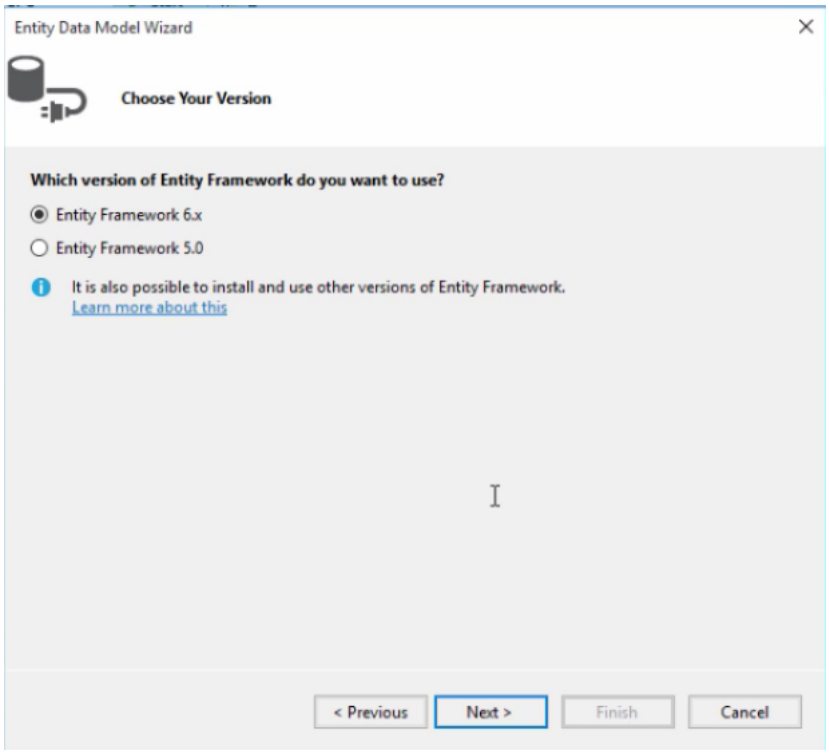
Deixaremos selecionada a primeira opção! Clicando em `Next`, será aberta uma nova janela:



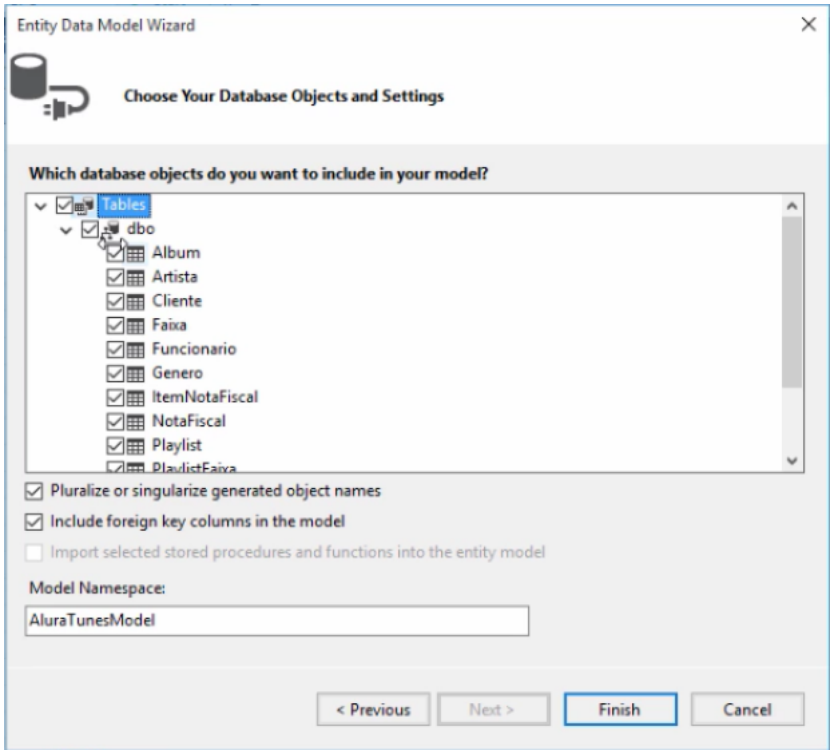
Diante disto, selecionaremos novamente a opção "Next" e veremos um alerta avisando que estamos utilizando um banco de dados local. Clicaremos em "Yes":



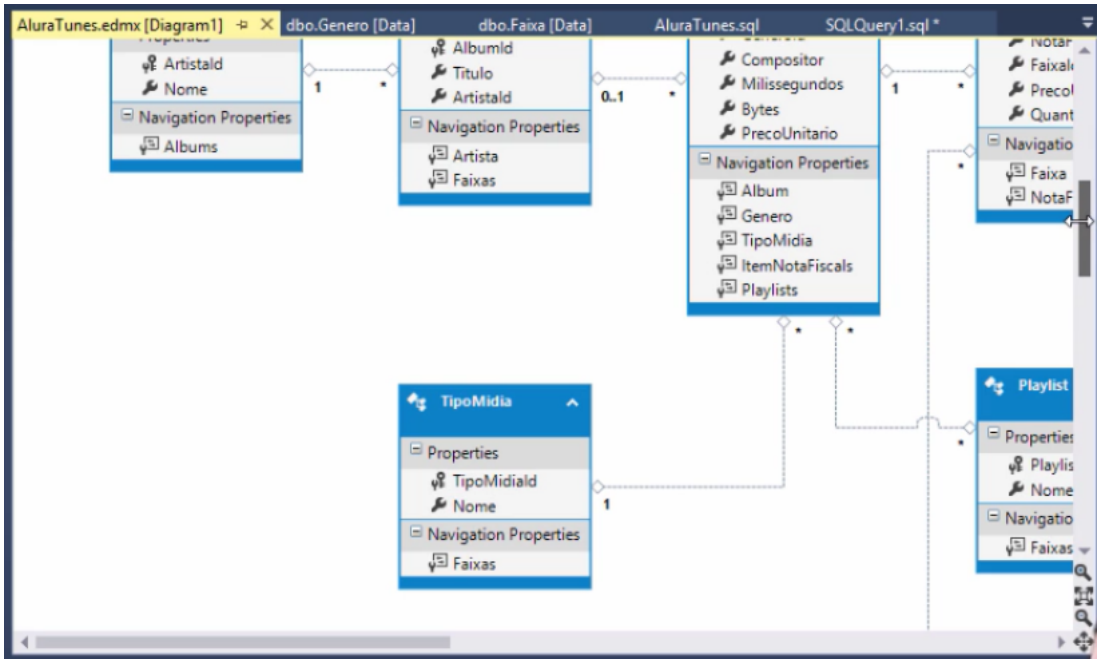
Uma outra janela será aberta perguntando qual a versão do `Entity Framework` que vamos utilizar, no caso, a `Entity Framework 6.x` :



Dando um "Next", teremos outra janela aberta e nela aparecerá uma pergunta sobre quais objetos do `database` desejamos incluir no modelo e com isso incluímos um `check` no `Tables` e, assim, teremos todas as tabelas selecionadas:

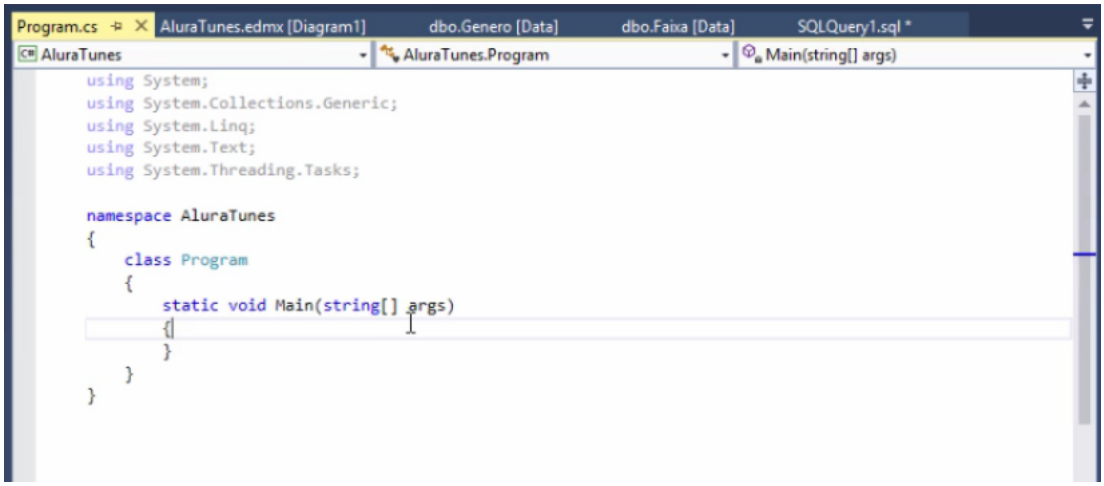


Observe que existe a opção de pluralizar ou singularizar os nomes. Quando deixamos a opção `pluralizar` marcada significa que teremos mais facilidade distinguir o nome de um objeto do nome de uma coleção de objetos. Com isso, podemos clicar em "Finish" e teremos o modelo de entidades do `Entity Framework` com o nome de `AluraTunes.edmx` :



Agora, vamos começar a brincar com o `LINQ` do **Entity**!

Como o cliente pediu para realizar uma consulta em cima da base de dados, vamos abrir o `Programa.cs` :



Primeiro, é preciso criar um contexto que será utilizado para acessar todas as tabelas e propriedades do modelo **Entity Framework**. Assim, declaramos `using (var contexto = new )` e colocamos junto do `new` o nome do modelo, o `AluraTunesEntities` . Dentro disso, ìremos inserir uma definição de consulta e, em seguida, imprimimos os dados no console. Para definir a consulta criamos uma variável `query` que será igual a `from` e dessa maneira acessamos tanto o contexto quanto uma tabela, portanto, escrevemos `from g in contexto.Generos` . E como falta trazer os dados do `Genero` acrescentamos o `select g` .

Teremos:

```
using (var contexto = new AluraTunesEntities())
{
    var query = from g in contexto.Generos
                select g;

}
```

O `LINQ` é capaz de acessar o `Gênero` , pois, o `DbSet` que também é implementado por ele é igualmente acessível para leitura. O `DbSet` , por sua vez, implementa o `IEnumerable` e todos os objetos que implementam o `IEnumerable` incluindo as entidades do **Entity Framework** podem ser acessados pelo `LINQ` . Por isso a tabela `Genero` pode ser acessada via `LINQ` .

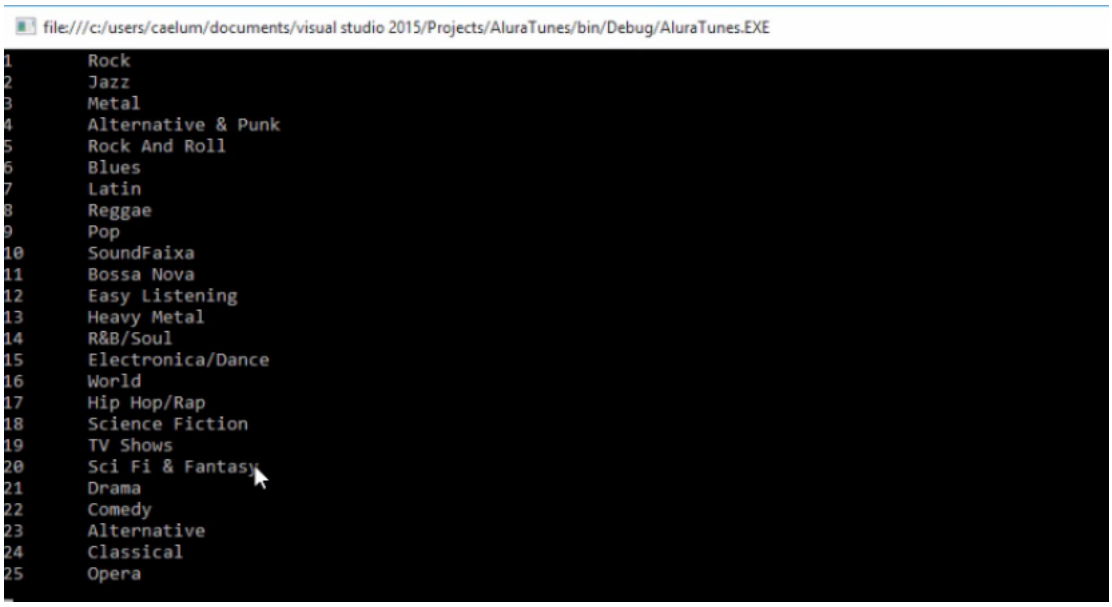
Agora, vamos imprimir no `Console` , as informações referentes a consulta. Assim, escreveremos `foreach (var genero in query)` e, dentro disso, acrescentaremos o `Console.WriteLine()` . Passaremos para o `Console` os dados sobre gênero. No caso, as propriedades `Id` e `Nome` , `genero.GeneroId` e `genero.nome` respectivamente. Falta ainda, acrescentar a `string` de formatação, `"{0}\t{1}"` . O que código ficará assim:

```
using (var contexto = new AluraTunesEntities())
{
    var query = from g in contexto.Generos
                select g;`

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

Rodando isso os dados serão exibidos e logo, em seguida, a aba se fechará. Isto ocorre, pois falta acrescentar o `Console.ReadKey()` para fazer com que ele aguarde uma tecla do usuário e feche! Ficará assim:

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

    Console.ReadKey();
```

Rodando isso temos o seguinte:



Dessa maneira temos todos os gêneros musicais existentes no banco de dados do cliente!

Esta consulta é bastante simples, mas ela pode se tornar complicada à medida que avançamos no curso. O `LINQ` e o `Entity Framework` possuem uma relação perfeita. Não é possível avançar muito no `Entity Framework` se não tivermos uma boa base

de `LINQ` !

O próximo pedido do cliente é imprimir músicas! As músicas utilizadas no novo modelo serão chamadas de faixas. Para fazer essa consulta, criaremos uma nova `query` , a `var faixaEgenero` , que conterá os dados da tabela. Utilizaremos o `join` para construir o `join f Contexto.Faixas` e como é preciso fazer a ligação entre a `Faixa` e o `Genero` , escreveremos `on g.GeneroId` . Faremos que isto equivalha, utilizando o `equals` , a `f.GeneroId` . Ainda, vamos fazer um `select` para que os dados da faixa e gênero sejam trazidos e combinados:

```
select new { f, g }
```

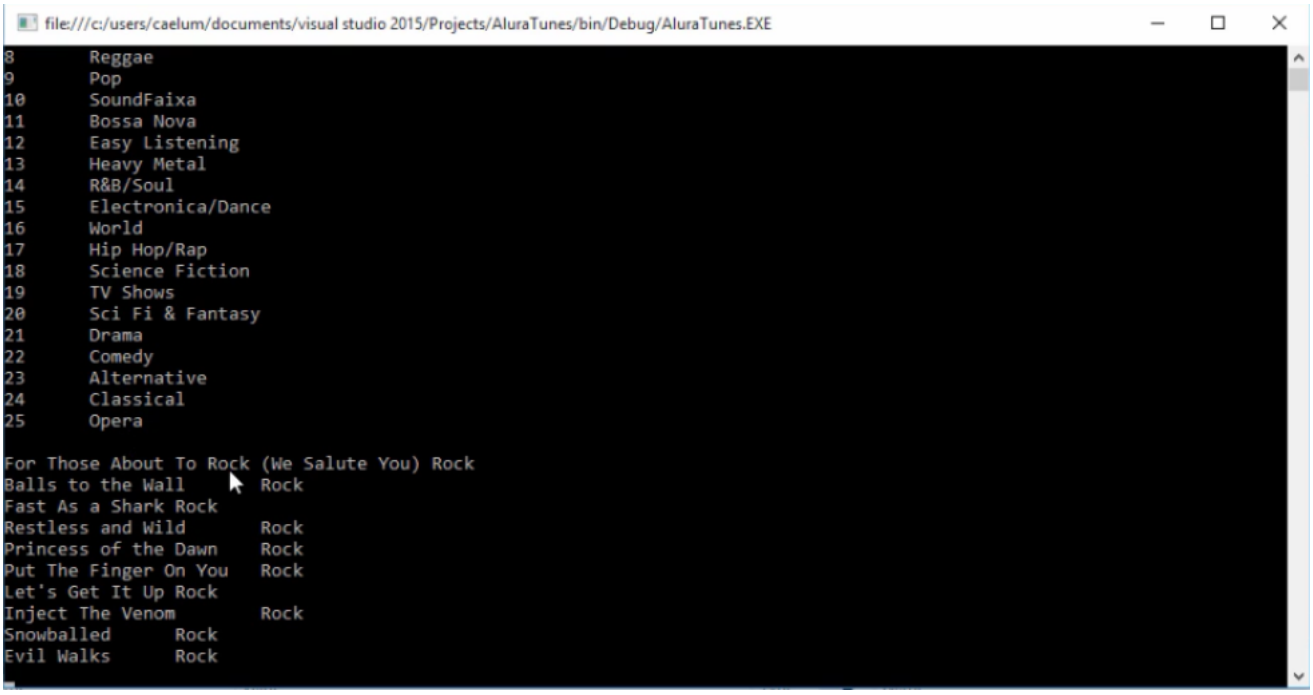
Teremos:

```
var faixaEgenero = from g in contexto.Generos
                   join f in contexto.Faixas
                   on g.GeneroId equals f.GeneroId
                   select new { f, g };
```

Agora, para imprimir a `faixaEgenero` , vamos utilizar o `foreach()` e passaremos para ele o `var item In faixaEgenero` . Dentro dele, acrescentaremos o `Console.WriteLine()` e vamos imprimir nele o `item.f.Nome, item.g.Nome` e a formatação da `string` , o `"{0}\t{1}"` . Teremos:

```
var faixaEgenero = from g in contexto.Generos
                   join f in contexto.Faixas
                   on g.GeneroId equals f.GeneroId
                   select new { f, g };

foreach (var item In faixaEgenero)

{
    Console.WriteLine("{0}\t{1}", item.f.Nome, item.g.Nome )  ;
}
```

Rodando isso teremos o seguinte:



Observe que diversos itens foram trazidos! Mas isso não é tão positivo, pois não queremos que na consulta venham todos os dados. Para resolver esse problema é preciso modificar a `query` do `faixaEgenero` para que ela traga um número limitado de resultados. Portanto, vamos escrever:

```
faixaEgenero = faixaEgenero.Take()
```

Depois, passaremos a quantidade de elementos que desejamos visualizar, no caso, o valor `10`:

```
var faixaEgenero = from g in contexto.Generos
                   join f in contexto.Faixas
                   on g.GeneroId equals f.GeneroId
                   select new { f, g };

faixaEgenero = faixaEgenero.Take(10);

foreach (var item In faixaEgenero);
```

Ainda, vamos acrescentar uma linha que irá separar ambas as listas. Adicionaremos o `Console.WriteLine()`:

```
var faixaEgenero = from g in contexto.Generos
                   join f in contexto.Faixas
                   on g.GeneroId equals f.GeneroId
                   select new { f, g };

faixaEgenero = faixaEgenero.Take(10);


Console.WriteLine();
foreach (var item In faixaEgenero);
```

E rodando isso teremos as 10 primeiras faixas, o nome da faixa e o nome do gênero!

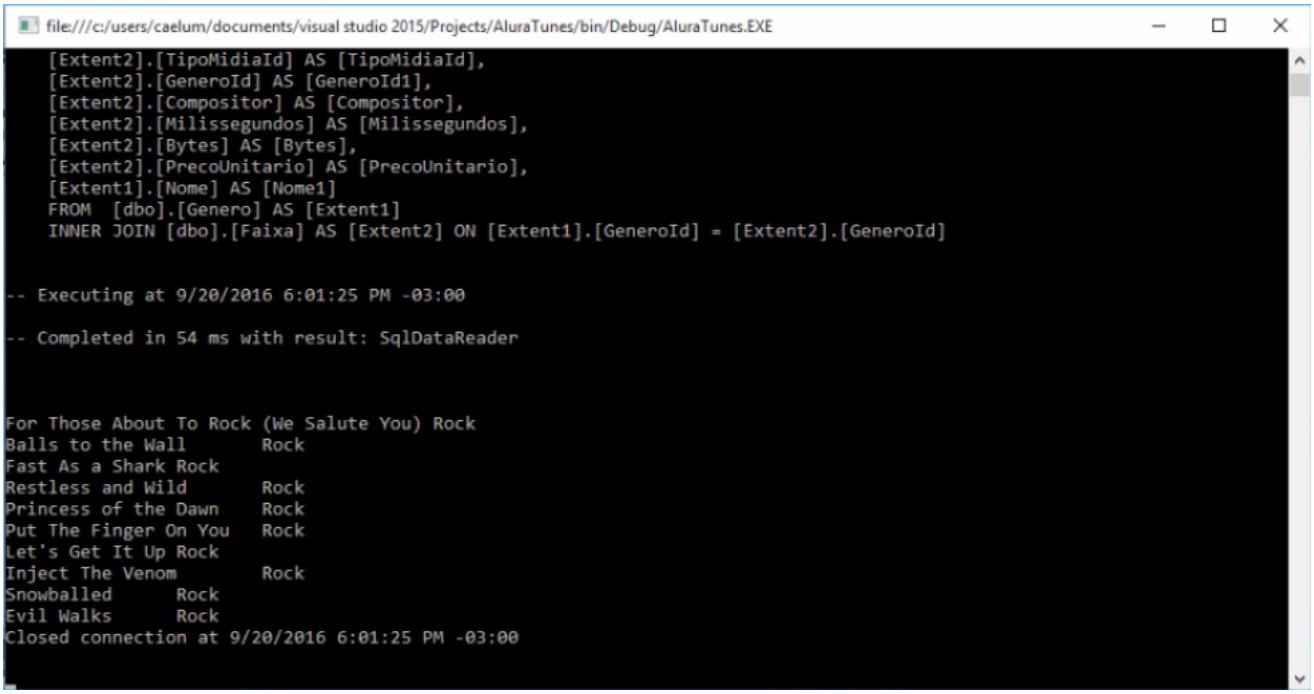
Agora, será que esse tipo de consulta é verdadeiramente eficiente? O que acontece é que não trazemos todos os dados do servidor para o cliente e, depois, aplicaremos um filtro. Na verdade, a consulta `LINQ` - que é traduzida para o `sqlserver` em formato `script` - já possui uma limitação. Ou seja, não estamos trazendo mais do que dez registros, mas sim, apenas 10!

Como saber como é a consulta `LINQ` que está sendo traduzida para o `sql` ? Podemos observar o `script` gerado e para isso basta criar uma configuração para trazer no `Console` , tudo o que é gerado do `LINQ` para o `SQL` . Primeiro, acessaremos o `contexto` escrevendo `contexto.Database.Log` . Em seguida, adicionaremos um método que receberá todos os `logs` do Banco de Dados: `Console.WriteLine()` :

```
contexto.Database.Log = Console.WriteLine;
```


```
contexto.Database.Log = Console.WriteLine;
```


Rodando a consulta teremos o seguinte :



Rodando a aplicação uma consulta `sql` aparecerá! Observe que inclusive aparece a cláusula `INNER JOIN` que traz quais são os campos e colunas que estão sendo associados entre uma tabela e outra. Ademais, aparecem informações sobre o tempo que demorou para carregar as informações e isso nos auxilia para descobrir onde estão os gargalos, problemas de performance, `join` errados, etc. Essa ferramenta é muito importante para depuração da aplicação!

Vimos nesta aula, como fazer consultas simples do `LINQ` utilizando o contexto do `Entity Frame` e também consultas um pouco mais complexas utilizando o `join` . Vimos sobre a importância do relacionamento entre `LINQ` e `Entity Framework` e como acessar qualquer tipo de objeto que implemente a interface `IEnumerable` em uma consulta `LINQ` . Ainda, aprenderemos a pegar um número limitado de objetos e linhas em uma consulta `LINQ` e, por último, vimos como depurar, debugar e visualizar tanto os logs como também consultas `sql` geradas a partir de uma `LINQ` .