05

Recuperando os atores de um filme

Transcrição

A única função que a classe `FilmeAtor` possui é relacionar outras duas classes `Filme` e `Ator`. Essa relação possibilita realizarmos buscas integradas, como por exemplo, pesquisar atores e atrizes que atuaram em um determinado filme. Iremos elaborar um código no programa que busca atender essa relação entre classes.

Na classe `Program`, adicionaremos um filme qualquer dentro de uma variável denominada `filme`. Acessaremos o banco através da variável `contexto`, acionaremos o primeiro filme que está disponível no banco de dados (`First()`), ou seja, um *select top 1*. Queremos que o filme selecionado seja mostrado no console, bem como seu elenco.

```
namespace Alura.Files.App
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var contexto = new AluraFilmesContexto())
            {
                contexto.LogSQLToConsole();

                var filme = contexto.Filmes.First();

                Console.WriteLine(filme);
                Console.WriteLine("Elenco");

                foreach (var item in contexto.Elenco)
                {
                    var entidade = contexo.Entry(item);
                    var filmId = entidade.Property("film_id").CurrentValue;
                    var actorId = entidade.Property("actor_id").CurrentValue;
                    var lastUpd = entidade.Property("last_update").CurrentValue;
                    Console.WriteLine($Filme {filmId}, Ator {actorId}, LastUpdate: {lastUpd}");
                }
            }
        }
    }
}
```

Feito isso, eliminaremos a parte do código com as variáveis `entidade`, `fimlId`, `actorId` e `lastUpd` que não serão úteis no momento. Queremos percorrer uma lista com todos os atores do filme. Portanto, o ideal é termos na instância de `filme` uma lista de `Atores`, e mostrar esses atores no console. Adicionaremos essas orientações ao código.

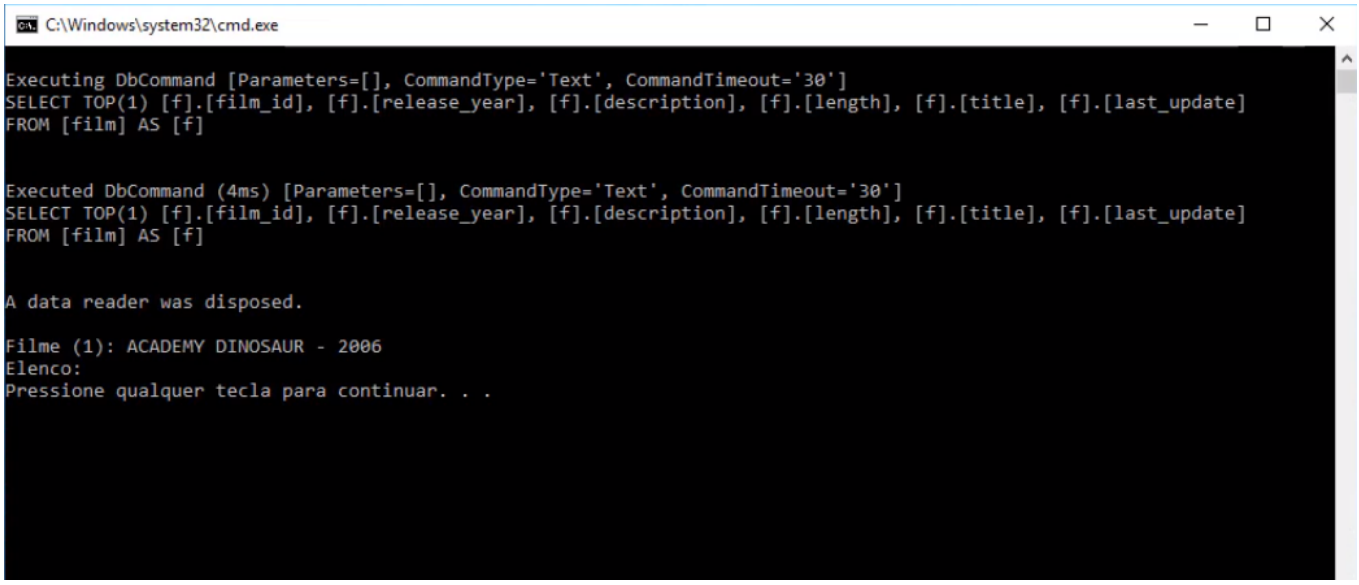
```
namespace Alura.Files.App
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var contexto = new AluraFilmesContexto())
```

```
            {
                contexto.LogSQLToConsole();

                var filme = contexto.Filmes.First();

                Console.WriteLine(filme);
                Console.WriteLine("Elenco");

                foreach (var ator in filme.Atores)
                {
                    Console.WriteLine(ator);
                }
            }
        }
    }
}
```

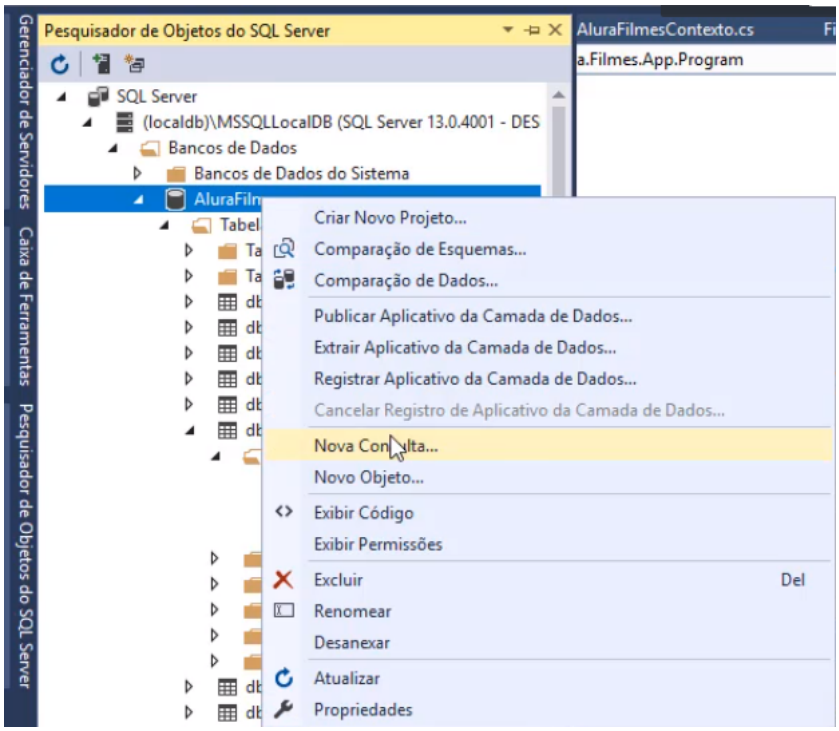
Na classe `Filme`, criaremos a lista de atores através do `Ilist` e importaremos o namespace. Depois, criaremos uma lista vazia de atores através do comando `List<T>`.

```
namespace Alura.FilmesApp.Negocio
{
    public class Filme
    {
        public int Id { get; set; }
        public string Titulo { get; set; }
        public string Descricao { get; set; }
        public string AnoLancamento { get; set; }
        public short Duracao { get; set; }
        public IList<Ator> Atores { get; set; }

        public Filme()
        {
            Atores = new List<Ator>();
        }

        public override string ToString()
        {
            return $"Filme ({Id}): {Titulo} - {AnoLancamento}";
        }
    }
}
```

Iremos executar o programa para realizar um pequeno teste. Veremos que o select top 1 foi realizado, pois temos a exibição do primeiro filme de `id = 1`. A lista de `Elenco` foi exibida, mas ainda está vazia. Veremos se no banco de dados há atores para o filme de de `Id = 1`.



Na área do "Gerenciador de Objetos SQL Server", selecionamos o banco de dados `AluraFilmes` e pressionaremos o botão direito. Feito isso, escolheremos a opção "Nova Consulta".



Realizaremos a pesquisa através do `select`. Da tabela `actor`, queremos acionar apenas os atores do filme `id = 1`. Como não existe uma coluna `film_id = 1` na tabela `Ator`, precisaremos fazer um `inner join` com uma tabela que fez esse relacionamento. Como estamos trabalhando com uma relação de **N:N**, vincularemos a tabela de `inter join` com a tabela `actor_id` através da coluna `actor_id`.

```
select a.*
from actor a
    inner join film_actor fa on fa.actor_id = actor_id
where fa.film_id = 1
```

Ao executarmos o programa perceberemos que existem quatro atores que estrelaram no filme de `Id = 1`.

Precisaremos fazer o que o Entity execute o `inner join`. Para isso, iremos até a classe `Program` e no `DbSet` incluiremos o método `Include()`. Iremos passar o método uma expressão lambda indicando qual é a propriedade do relacionamento.

```
namespace Alura.Files.App
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var contexto = new AluraFilmesContexto())
            {
                contexto.LogSQLToConsole();

                var filme = contexto.Filmes.First()
                    .Include(f => f.Atores);
                    .First();

                Console.WriteLine(filme);
                Console.WriteLine("Elenco");

                foreach (var ator in filme.Atores)
                {
                    Console.WriteLine(ator);
                }
            }
        }
    }
}
```

Ao tentarmos executar o programa veremos a mensagem de erro `Invalid column name 'FilmId'.`. Observaremos também no console o `select` feito pelo Entity:

```
FROM [actor] AS [f.Atores]
INNER JOIN(
    SELECT TOP(1) [f0]. [film_id]
    FROM [film] AS [f0]
    ORDER BY [f0].[film_id]
)AS [t] on [f.Atores].[FilmeId] - [t].[film_id]
ORDER BY [t].[film_id]
```

Percebam que o Entity tentou realizar o `inner join` com a tabela `film`. Sabemos que o nosso relacionamento é de "muitos para muitos", portanto, deve haver uma tabela intermediária (`FilmeAtor`) que não está aparecendo no `select`.

O Entity está utilizando mais uma de suas regras implícitas para fazer o relacionamento. Essa regra está sendo realizada pela propriedade `Atores` que incluímos na classe `Film`.

```
namespace Alura.FilmesApp.Negocio
{
    public calss Filme
    {
        public int Id { get; set; }
        public string Titulo { get; set; }
        public string Descricao { get; set; }
        public string AnoLancamento { get; set; }
        public short Duracao { get; set; }
        public Ilist<Ator> Atores { get; set; }

        public Filme()
        {
            Atores = new List<Ator>();
        }

        public override string ToString()
        {
            return $"Filme ({Id}): {Titulo} - {AnoLancamento}";
        }
    }
}
```

Estamos diante de mais uma das convenções do Entity. Existe um relacionamento entre a classe a ser escaneada e a lista de atores, precisamos analisar a cardinalidade desse relacionamento. Em um filme, nós temos uma lista de atores, então o Entity irá entender que a cardinalidade é **muitos**.

Porém, ao mapear a classe `Ator`, o Entity não percebe nenhum relacionamento com a classe `Film` e supõe que é um relacionamento para **1**, portanto, o Entity criará uma shadow property para **1** filme, criando um relacionamento **1:N**.

Queremos que a tabela `FilmeAtor` faça o `inner join`. Para isso, faremos a seguinte modificação na classe `Filme`: ao invés de termos uma lista de `Atores` teremos uma lista de instâncias da classe `FilmeAtor`.

```
namespace Alura.FilmesApp.Negocio
{
    public calss Filme
    {
        public int Id { get; set; }
        public string Titulo { get; set; }
        public string Descricao { get; set; }
        public string AnoLancamento { get; set; }
        public short Duracao { get; set; }
        public Ilist<Ator> Atores { get; set; }

        public Filme()
        {
            Atores = new List<Ator>();
        }

        public override string ToString()
        {
```

```
                return $"Filme ({Id}): {Titulo} - {AnoLancamento}";
            }
        }
    }
```

Ao executarmos o programa, veremos novamente uma ocorrência de erro. Porém, o `select` apresenta diferenças.

```
FROM [film_actor] AS [f.Atores]
INNER JOIN (
    SELECT TOP(1) [f0]. [film_id]
    FROM [film] AS [f0]
    ORDER BY [f0].[film_id]
)AS [t] on [f.Atores].[FilmeId] - [t].[film_id]
ORDER BY [t].[film_id]
```

Percebemos que há um relacionamento que existe no banco legado (`FROM [film_actor] AS [f.Atores]` e `FROM [film]`). O erro está no fato de que a coluna `FilmId` não existe dentro da tabela `film_actor` .

O Entity utiliza uma convenção para descobrir as chaves estrangeiras muito parecida com a regra das chaves primárias, ou seja, se não houver nenhuma chave estrangeira definida na respectiva classe, será utilizada um `<nome do tipo>Id` .