07

# Configurando chaves estrangeiras

### Transcrição

Aprendemos que a convenção do Entity para definir chaves estrangeiras é utilizar `<nome do tipo> Id`, o que não atende aos nossos interesses no projeto, pois não corresponde ao nome da chave estrangeira do banco legado.

Teremos de configurar a chave estrangeira para `filme`. A chave está presente na tabela `film_actor`, portanto, trabalharemos na classe `FilmeAtor`.

Para configurarmos uma chave estrangeira precisamos determinar os dois lados de um relacionamento. Neste caso, estamos lidando com um relacionamento de 1:N, pois estamos relacionando a classe `join` com a classe `FilmeAtor` e `Film`. `FilmeAtor` aponta apenas para um `Film`, e `Film` aponta para uma lista de instâncias de `FilmeAtor`.

Iremos declarar que a configuração `FilmeAtor` possui um filme ( `HasOne()` ), e passaremos uma expressão lambda para informar qual é a propriedade que representa a instância do filme.

```
namespace Alura.Filme.App.Dados
{
    public class FilmeAtorConfiguration : IEntityTypeConfiguration<FilmeAtor>
    {
        public void Configure(EntityTypeBuilder<FilmeAtor> builder)
        {
            builder.ToTable("film_actor");

            builder.Property<int>("film_id")
            builder.Property<int>("actor_id")
            builder.Property<DateTime>("last_update")
                .HasColumnType("datetime")
                .HasDefaultValueSql("getdate()");

            builder.HasKey("film_id", "actor_id");

            builder
                .HasOne(fa => fa.Filme)
        }
    }
}
```

Não temos ainda a propriedade representativa da instância do filme, portanto, iremos cria-la na classe `FilmeAtor`.

```
namespace Alura.Filmes.App.Negocio
{
    public class FilmeAtor
    {
        public Filme Filme { get; set; }
    }
}
```

Feito isso, continuaremos a configurar a chave estrangeira. Iremos declarar o outro lado da relação, ou seja, o lado do "muitos"(N). O método que usaremos para isso será o `WithMany()` e passaremos uma expressão lambda para informar qual é a propriedade desse relacionamento especificamente. Faremos para o tipo `Filme`.

Para declarar o nome da chave estrangeira ( `film_id` ) usaremos o método `HasForeignKey()`.

```
namespace Alura.Filme.App.Dados
{
    public class FilmeAtorConfiguration : IEntityTypeConfiguration<FilmeAtor>
    {
        public void Configure(EntityTypeBuilder<FilmeAtor> builder)
        {
            builder.ToTable("film_actor");

            builder.Property<int>("film_id")
            builder.Property<int>("actor_id")
            builder.Property<DateTime>("last_update")
                .HasColumnType("datetime")
                .HasDefaultValueSql("getdate()");

            builder.HasKey("film_id", "actor_id");

            builder
                .HasOne(fa => fa.Filme)
                .WithMany(f => Atores)
                .HasForeignKey("film_id");
        }
    }
}
```

Feito isso, configuramos a chave estrangeira na classe `FilmeAtor` que existe na tabela `film_actor`. Ao executarmos o programa, perceberemos que não existe ocorrência de erro.

No console, existem quatro itens da lista que correspondem aos quatro atores.

```
Filme (1): ACADEMY DINOSAUR - 2006
Elenco:
Alura.Filmes.App.Negocio.FilmeaAtor
Alura.Filmes.App.Negocio.FilmeaAtor
Alura.Filmes.App.Negocio.FilmeaAtor
Alura.Filmes.App.Negocio.FilmeaAtor
```

Observaremos `select` realizado pelo Entity:

```
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].[Film_id] - [t].[film_id]
ORDER BY [t].[film.id]
```

Percebam que o problema da chave estrangeira foi corrigido.

Precisamos que sejam exibidos os nomes dos atores, não apenas instâncias da classe `FilmeAtor`, ou seja, queremos saber os valores de cada registro. Para isso, iremos realizar mais um `join` no nosso `select`.

O `select` que fizemos estará nessa configuração:

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

Faremos adaptações para que o nosso `select` fique mais parecido com o que foi gerado pelo Entity e aplicaremos para `inner join`.

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

Com isso, teremos os registros da tabela `Atores`. Tivemos de colocar mais um `join`, ou seja, avançar mais um nível no estabelecimento de relações. Por isso, na classe `Program`, teremos de adicionar um novo método chamado `ThenInclude()`. Iremos na instância da classe `FilmeAtor` que chamaremos de `fa`. Passeremos uma expressão lambda para dizer qual é a propriedade com que queremos fazer o 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)
                    .ThenInclude(fa => fa.Ator)
                    .First();

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

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

O objetivo é atingir o nível dos `Atores`. A questão é que ainda não existe na classe `FilmeAtor` uma propriedade que representa a instância `Ator`, portanto, criaremos essa instância.

```
namespace Alura.Filmes.App.Negocio
{
    public class FilmeAtor
    {
        public Filme Filme { get; set; }
        public Ator Ator { get; set; }
    }
}
```

Ao tentarmos executar o programa, teremos uma mensagem de erro: `invalid column name 'AtorId'.` O Entity está acionando a convenção de utilizar como chave estrangeira `<nome do tipo>Id`. Teremos de configurar a chave estrangeira de `film_actor` para `actor`.

Na classe `FilmeAtorConfiguration`, faremos a configuração para `Atores`. No método `HasOne()` passaremos uma expressão lambda declarando que `fa` possui relação com a propriedade `Ator`. Feito isso, acionaremos a propriedade `WithMany()`, indicando os múltiplos relacionamentos. Não temos ainda a lista de `FilmeAtor`, portanto criaremos uma lista denominada `Filmografia`.

```
namespace Alura.Filme.App.Dados
{
    public class FilmeAtorConfiguration : IEntityTypeConfiguration<FilmeAtor>
    {
        public void Configure(EntityTypeBuilder<FilmeAtor> builder)
        {
            builder.ToTable("film_actor");

            builder.Property<int>("film_id")
            builder.Property<int>("actor_id")
            builder.Property<DateTime>("last_update")
                .HasColumnType("datetime")
                .HasDefaultValueSql("getdate()");

            builder.HasKey("film_id", "actor_id");

            builder
                .HasOne(fa => fa.Filme)
                .WithMany(f => Atores)
                .HasForeignKey("film_id");

            builder
                .HasOne(fa => fa.Ator)
                .WithMany(a => a.Filmografia)
        }
    }
}
```

Na classe `Ator`, criaremos a lista `Filmografia`, sem esquecer de importarmos o `IList`.

```
namespace Alura.Filmes.App.Negocio
{
    public class Ator
    {
        public int Id { get; set; }
        public string PrimeiroNome { get; set; }
        public string UltimoNome { get; set; }
        public IList<FilmeAtor> Filmografia { get; set; }

        public override string ToString()
        {
            return $"Ator ({Id}): {PrimeiroNome} {UltimoNome}";
        }
    }
}
```

Iremos também acionar um construtor para criar uma lista de `FilmeAtor` vazia.

```
namespace Alura.Filmes.App.Negocio
{
    public class Ator
    {
        public int Id { get; set; }
        public string PrimeiroNome { get; set; }
        public string UltimoNome { get; set; }
        public IList<FilmeAtor> Filmografia { get; set; }

        public Ator()
        {
            Filmografia = new List<FilmeAtor>();
        }

        public override string ToString()
        {
            return $"Ator ({Id}): {PrimeiroNome} {UltimoNome}";
        }
    }
}
```

Feito isso, podemos voltar para a configuração da chave estrangeira na classe `FilmeAtorConfiguration`. Adicionaremos o método `HasForeignKey()` e o nome da chave estrangeira será `actor_id`.

```
namespace Alura.Filme.App.Dados
{
    public class FilmeAtorConfiguration : IEntityTypeConfiguration<FilmeAtor>
    {
        public void Configure(EntityTypeBuilder<FilmeAtor> builder)
        {
            builder.ToTable("film_actor");

            builder.Property<int>("film_id")
            builder.Property<int>("actor_id")
            builder.Property<DateTime>("last_update")
                .HasColumnType("datetime")
```

```
                .HasDefaultValueSql("getdate()");

            builder.HasKey("film_id", "actor_id");

            builder
                .HasOne(fa => fa.Filme)
                .WithMany(f => Atores)
                .HasForeignKey("film_id");

            builder
                .HasOne(fa => fa.Ator)
                .WithMany(a => a.Filmografia)
                .HasForeignKey("actor_id);
        }
    }
}
```

Ao executarmos o programa, perceberemos que não há ocorrência de erro. O Entity conseguiu realizar os dois `inner join` s, o primeiro de `film_actor` com `actor` e o segundo de `film_actor` com `film` .

```
FROM [film_actor] AS [f.Atores]
INNER JOIN [actor] AS [f.Ator] ON [f.Atores].[actor_id] = [f.Ator].[actor_id]
INNER JOIN (
    SELECT TOP (1) [f0].[film_id]
    FROM [film] AS [f0]
    ORDER BY [f0].[film_id]
) AS [t] ON [f.Atores].[film_id] = [t].[film_id]
ORDER BY [t].[film_id]
```

Ainda não estão sendo exibidos os nomes dos atores, e sim os registros.

```
Filme (1): ACADEMY DINOSAUR - 2006
Elenco:
Alura.Filmes.App.Negocio.FilmeaAtor
Alura.Filmes.App.Negocio.FilmeaAtor
Alura.Filmes.App.Negocio.FilmeaAtor
Alura.Filmes.App.Negocio.FilmeaAtor
```

Para solucionar essa questão faremos uma pequena modificação na classe `Program` : adicionaremos a propriedade `Ator` em `WriteLine()`

```
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)
```

```
                .ThenInclude(fa => fa.Ator)
                .First();

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

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

Ao executarmos o programa, veremos que os nomes dos atores do filme. Os relacionamentos estão configurados.

```
Filme (1): ACADEMY DINOSAUR - 2006
Elenco
Ator (1): PENELOPE STALLONE
Ator (10): CHRISTIAN GABLE
Ator (20): LUCILLE TRACY
Ator (30): SANDRA PECK
```