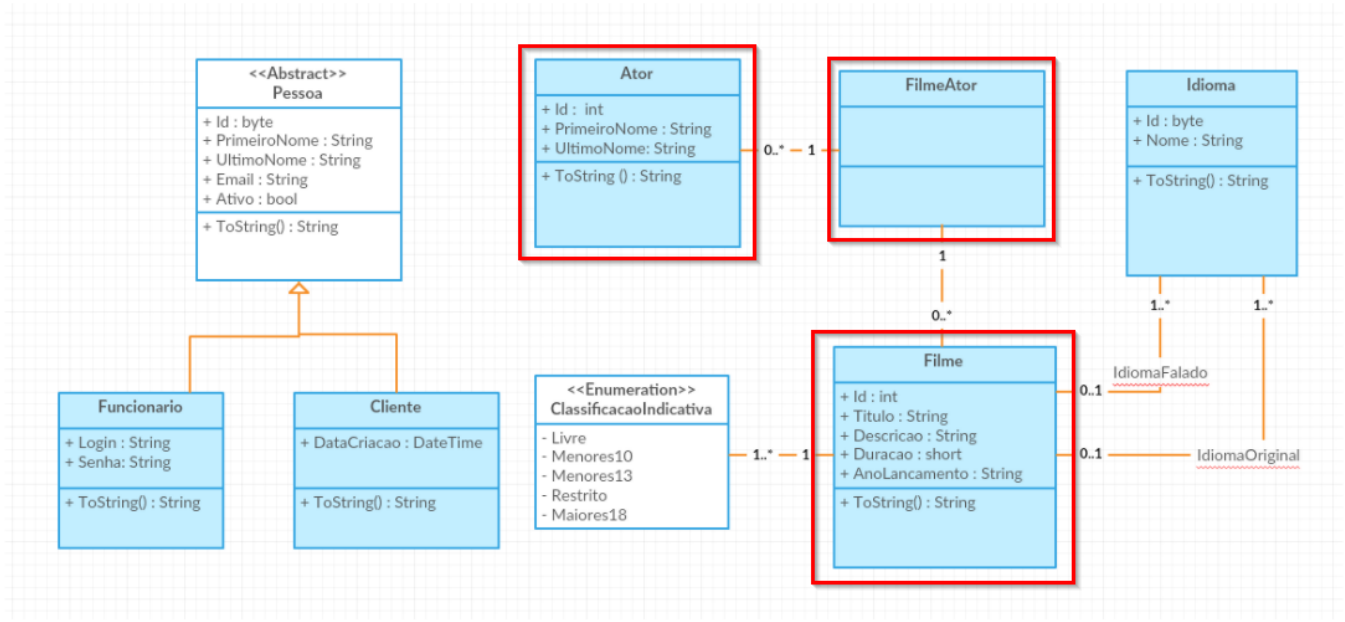
01

Relacionando filmes a atores

Transcrição

Nesta aula, pensaremos sobre o relacionamento da classe `Ator` com a classe `Filme`, portanto as duas classes já devem estar mapeadas.



Percebemos que as classes em questão se relacionam através de uma terceira classe denominada `FilmeAtor`. Um ator pode atuar em vários filmes, e um filme pode ser estrelado por vários atores, portanto, temos um relacionamento **N:N** (muitos para muitos). Vimos no curso anterior que o Entity Framework Core não cria automaticamente uma classe de `join`, o desenvolvedor deve explicitar essa classe.

Criaremos a classe `FilmeAtor` na camada de negócios. Como vimos no diagrama, ela não contem nenhuma informação.

```
namespace Açura.Filmes.App.Negocio
{
    public cass FilmeAtor
    {
    }
}
```

Com a classe já criada, iremos testar o seu mapeamento para o banco de dados legado. Tentaremos fazer um *select* na classe `FilmeAtor`. Na classe de contexto, adicionaremos uma propriedade `DbSet` denominada `Elenco` para `FilmeAtor`. É desnecessário criar uma propriedade para uma classe `join`, mas iremos faze-lo apenas para verificar o mapeamento do Entity em relação ao banco legado.

```
namespace Alura.Filmes.App.Dados
{
    public class AluraFilmesContexto : DbContext
    {
        public DbSet<Ator> Atores { get; set; }
```

```
        public DbSet<Filme> Filmes { get; set; }
        public DbSet<FilmeAtor> Elenco { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server=(localdb)mssqllocaldb;Database=AluraFilmes;Trusted_(
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new AtorConfiguration());
            modelBuilder.ApplyConfiguration(new FilmeConfiguration());

        }
    }
}
```

Observemos a classe `Program`:

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

                foreach (var filme in contexto.Filmes)
                {
                    Console.WriteLine(filme);
                }
            }
        }
    }
}
```

Iremos percorrer a propriedade `DbSet` chamada `Elenco`, e substituiremos `filme` por `item`.

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

                foreach (var item in contexto.Elenco)
                {
                    Console.WriteLine(item);
                }
```

```
                }
            }
        }
    }
```

Ao executarmos a aplicação, perceberemos a ocorrência de um erro : `The entity 'FilmeAtor' requires a primary key to defined` . Ou seja, a entidade `FilmeAtor` necessita de uma chave primária definida. Devemos nos perguntar por que este mesmo erro não ocorreu quando mapeamos a classe `Ator` e `Filme` . O erro que surgiu para as duas classes que mapeamos anteriormente foi apenas `invalid object name` .

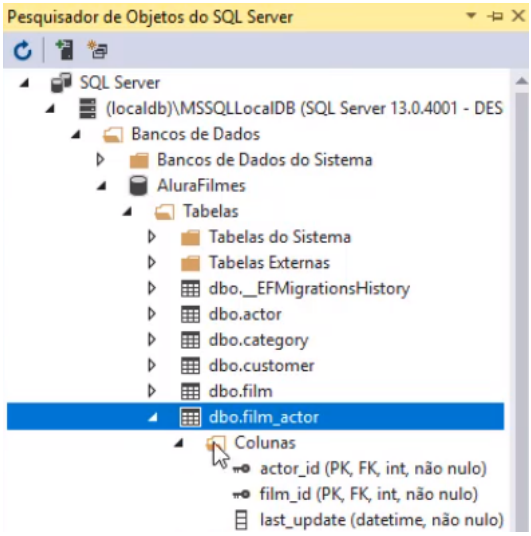
Analisaremos a classe `Ator` para identificar o que o *sofware* reconhece como uma chave primária.

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

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

Estamos nos deparando com mais uma convenção. Na documentação (https://docs.microsoft.com/pt-br/ef/core/modeling/keys#data-annotations) do Entity Framework Core veremos que uma propriedade `Id` ou `<type name>Id` será configurada como chave primária da entidade. Logo, tanto a classe `Ator` quanto `Filme` possuem uma propriedade `Id` que foi mapeada como chave primária. No caso da classe `FilmeAtor` não há nada definido em seu conteúdo, portanto, o Entity não consegue mapear nenhum item.

Observaremos qual é a chave primária do banco de dados legado `AluraFilmes` . Na área "Pesquisador de Objetos", selecionaremos a tabela "dbo.film_actor". Veremos que existem três colunas, e duas delas são consideradas chaves primárias.



Iremos estudar nas próximas aulas como definir uma chave primária com mais de uma coluna, ou seja, como mapear uma **chave primária composta**.