03

Definindo chaves primárias compostas

Transcrição

Para mapear uma chave primária composta, precisaremos criar uma classe de configuração. Na pasta "Dados", criaremos uma classe chamada `FilmeAtorConfiguration` .

Essa classe implementa a entidade `IEntityTypeConfiguration<FilmeAtor>` , que possui apenas um método chamado `Configure()` , que por sua vez, passa um argumento de entrada `builder` do tipo `EntityTypeBuilder` para a classe `FilmeAtor` .

Iremos registrar que a configuração da classe `FilmeAtor` será mapeada para a tabela `film_actor` . Também explicitaremos que o argumento possui uma chave primária composta por duas colunas e iremos representá-las (`film_id` , `actor_id`).

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

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

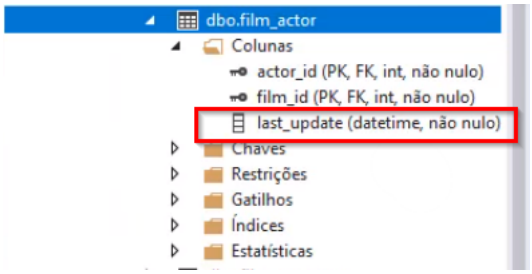
Reparem que as duas colunas só existem no mundo relacional e não no mundo orientado do objeto. Colunas que existem somente no mundo relacional são chamadas shadow properties. Portanto, antes de dizermos que a chave é composta pelas colunas `film_id` e `actor_id` , precisaremos definir a shadow property.

Deixaremos explicitado no código que temos uma propriedade do tipo `int` chamada `film_id` e não nula. Faremos o mesmo procedimento com a propriedade `actor_id` . Não é necessário utilizaremos o método `IsRequired()` neste caso, pois os tipos `int` e `DateTime` não aceitam valores nulos

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

            builder.Property<int>("film_id");

            builder.Property<int>("actor_id");

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

```
    }
}
```

Iremos configurar a ultima coluna `last_update` - também uma shadow property -. Criaremos uma propriedade `DateTime` e importaremos o system. Além disso, explicitaremos o tipo da coluna via `HasColumnType` , bem como o valor : `HasDefaultValueSql` .



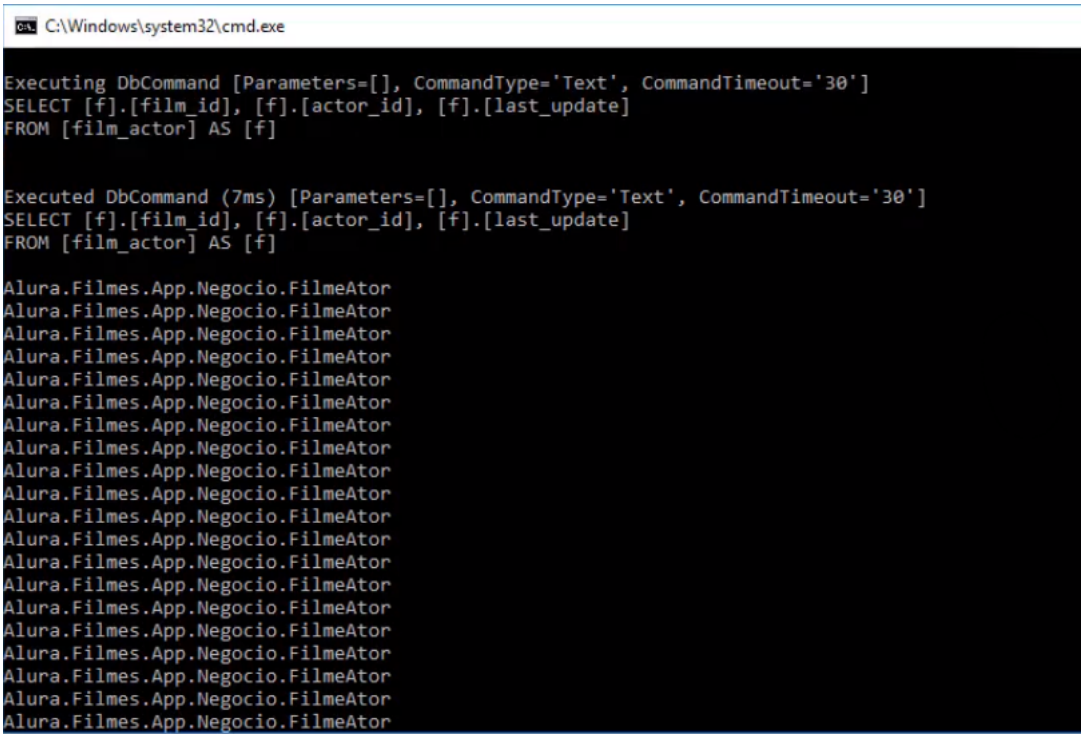

```
namespace Alura.Filmes.App.Dados
{
    public class FilmeAtorConfiguration : IEntityTypeConfiguration<FimeAtor>
    {
        public void Configure(EntityTypeBuilder<FilmeAtor> builder)
        {
            builder.ToTable("film_actor");

            builder.Property<int>("film_id");

            builder.Property<int>("actor_id");

            builder.Property<DateTime>("last_update")
            .HasColumnType("datetime")
            .HasDefaulValueSql("getdate()");

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

Aparentemente terminamos de configurar as propriedades, mas ao tentarmos executarmos o programa veremos a ocorrência de um erro : `The entity type 'FilmeAtor' requires a primary key to be defined` , ou seja, o mesmo erro anterior. O problema ocorreu, pois apesar de criarmos a classe de configuração nós não a utilizamos. Temos de ir no método `OnModelCreating()` na classe de contexto e aplicar a configuração.

```
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());
            modelBuilder.ApplyConfiguration(new FilmeAtorConfiguration());

        }
    }
}
```

Feito isso, a aplicação é executada corretamente. Porém, temos apenas o nome completo da classe sendo exibida. Precisaremos recuperar os valores das shadow properties.



Na classe `Program`, acionaremos o método `Entry()` e evocaremos o valor de cada shadow property, que são `film_id`, `actor_id` e `last_update`. Feito isso, iremos imprimir os valores.

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

                foreach (var item in contexto.Elenco)
                {
                    var entidade = contexto.Entry(item);
                    ver 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}");
```

```
                }
            }
        }
    }
}
```

Ao executarmos o programa veremos que os valores foram convertidos corretamente. Fizemos uma revisão sobre as shadow properties e conseguimos mapear a classe `FilmeAtor`, que é a classe `join` entre as classes `Filme` e `Ator`. Porém, a classe `FilmeAtor` só possui a função de relacionar outras duas classes, portanto, os valores dessa tabela não são importantes. O que queremos é saber para um determinado filme quais são os atores que atuaram, ou então para um determinado ator qual é filmografia dele. Começaremos a discutir essa relação entre informações nas próxiams aulas.