05

Recuperando filmes de um determinado idioma

Transcrição

Como já mapeamentos e configuramos o dois relacionamentos entre `Filme` e `Idioma` , iremos criar uma aplicação para testar esse relacionamento.

Começaremos evidenciando todos os filmes dublados de um determinado idioma. Criaremos uma variável para representar a lista de idiomas. Também incluiremos o relacionamento representado na propriedade `FilmesFalados` .

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

                var idiomas = contexto.Idiomas
                    .Include(i => FilmesFalados);

                foreach (var idioma in contexto.Idiomas)
                {
                    Console.WriteLine(idioma);
                }
            }
        }
    }
}
```

Faremos um `select` com `join` para o relacionamento entre `FilmesFalados` . Feito isso, pressionamos "F5" para testar a nossa aplicação.

Perceberemos a ocorrência de um erro : `Referência de objeto não definida para uma instância de um objeto.` Ao checarmos o `join` , perceberemos que as relações estão corretas.

```
FROM [Filme] AS [i.FilmesFalados]
INNER JOIN (
    SELECT [i0].[language_id]
    FROM [language] AS [i0]
) AS [t] ON [i.FilmesFalados].[language_id] - [t].[language_id]
ORDER BY [t].[language_id]
```

Portanto, o erro não pode estar no `join` .

Para entendermos a causa do problema, observaremos um exemplo. Na classe `Programa` , iremos criar uma variável de `filmes` que recebe uma propriedade `DbSet` de `Filmes` . Icluíremos o relacionamento de `IdiomaFalado` . Feito isso, listaremos `filmes` dentro da variável `filme` . Imprimiremos o filme e o idioma falado no console.

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

                var filmes = contexto.Filmes
                    .Include(f => f.IdiomaFalado);

                foreach (var filme in filmes)
                {
                    Console.WriteLine(filme);
                    Console.WriteLine(filme.IdiomaFalado);
                }

            }
        }
    }
}
```

Ao pressionarmos "F5" para executarmos a aplicação, perceberemos novamente a ocorrência do erro `Referência de objeto não definida para uma instância de um objeto.` Porém, o programa conseguiu imprimir uma parte dos filmes e os idiomas no console.

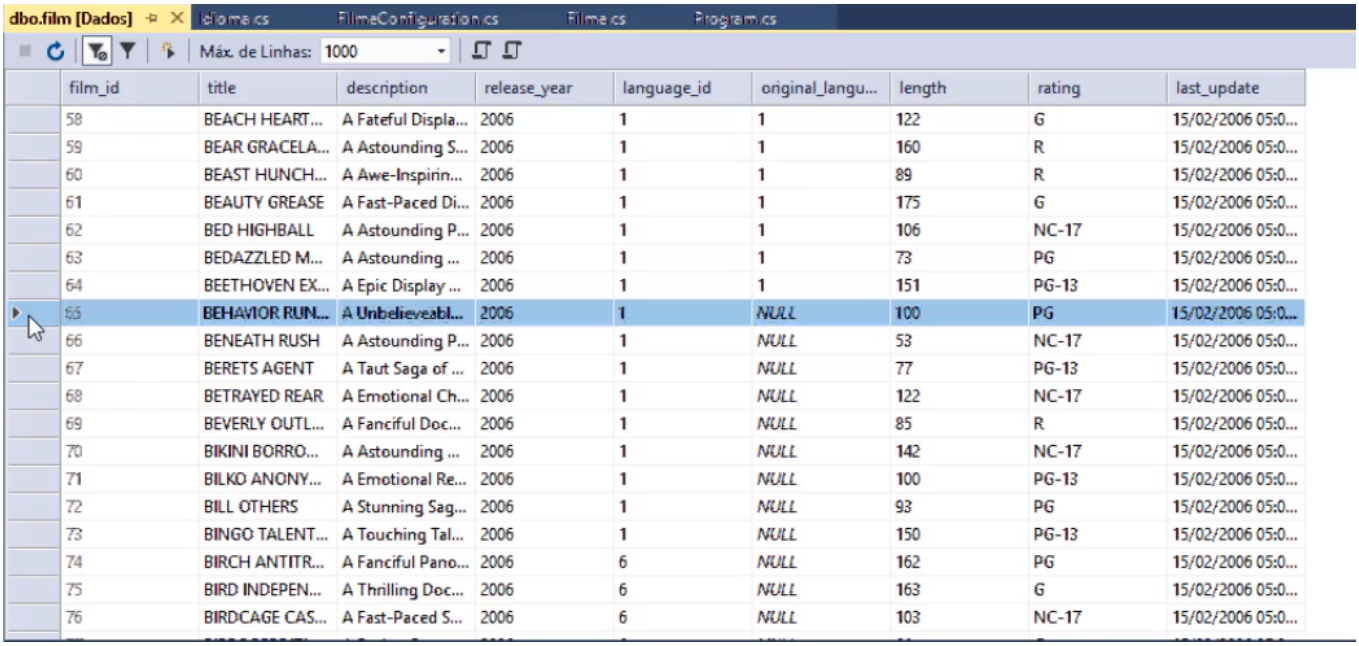
```
Filme (1): ACADEMY DINOSAUR - 2009
idioma (5): French
File (2): ACE GOLDFINGER - 2006
Idioma (5): French
Filme (3): ADAPTATION HOLES - 2006
Idioma (5): French

(...)

Filme (63): BEFAZZLED MARRIED - 2006
idioma (1): English
Filme (64): BEETHOVEN EXORCIST - 2006
Idioma (1): English

A data reader was disposed.
```

Percebemos que o erro ocorre após o filme de `id = 65` , pois o programa só conseguiu imprimir a lista até o filme de `id = 64` . Iremos até o banco de dados de `film` para checar o item de `id = 65` .



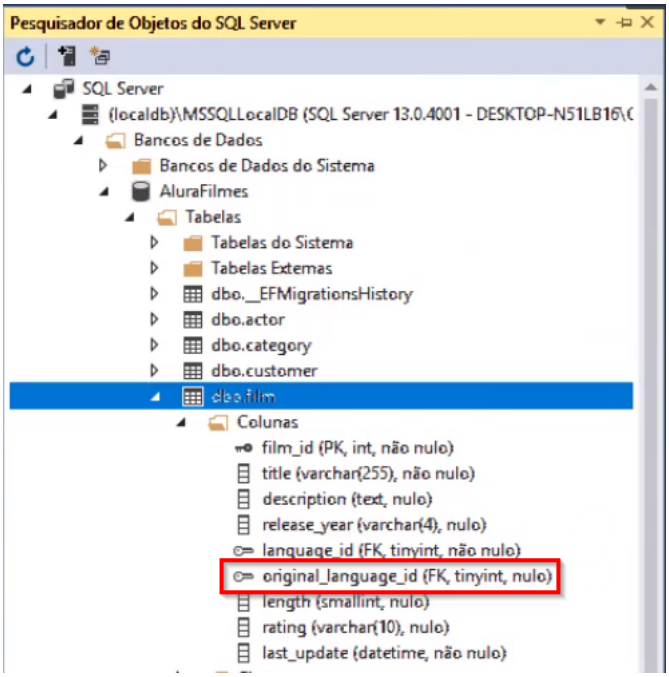
dbo.film [Dados] | Idioma.cs | FilmeConfiguration.cs | Filme.cs | Program.cs

Máx. de Linhas: 1000

film_id	title	description	release_year	language_id	original_langu...	length	rating	last_update
58	BEACH HEART...	A Fateful Displa...	2006	1	1	122	G	15/02/2006 05:0...
59	BEAR GRACELA...	A Astounding S...	2006	1	1	160	R	15/02/2006 05:0...
60	BEAST HUNCH...	A Awe-Inspirin...	2006	1	1	89	R	15/02/2006 05:0...
61	BEAUTY GREASE	A Fast-Paced Di...	2006	1	1	175	G	15/02/2006 05:0...
62	BED HIGHBALL	A Astounding P...	2006	1	1	106	NC-17	15/02/2006 05:0...
63	BEDAZZLED M...	A Astounding ...	2006	1	1	73	PG	15/02/2006 05:0...
64	BEETHOVEN EX...	A Epic Display ...	2006	1	1	151	PG-13	15/02/2006 05:0...
65	BEHAVIOR RUN...	A Unbelieveabl...	2006	1	NULL	100	PG	15/02/2006 05:0...
66	BENEATH RUSH	A Astounding P...	2006	1	NULL	53	NC-17	15/02/2006 05:0...
67	BERETS AGENT	A Taut Saga of ...	2006	1	NULL	77	PG-13	15/02/2006 05:0...
68	BETRAYED REAR	A Emotional Ch...	2006	1	NULL	122	NC-17	15/02/2006 05:0...
69	BEVERLY OUTL...	A Fanciful Doc...	2006	1	NULL	85	R	15/02/2006 05:0...
70	BIKINI BORRO...	A Astounding ...	2006	1	NULL	142	NC-17	15/02/2006 05:0...
71	BILKO ANONY...	A Emotional Re...	2006	1	NULL	100	PG-13	15/02/2006 05:0...
72	BILL OTHERS	A Stunning Sag...	2006	1	NULL	93	PG	15/02/2006 05:0...
73	BINGO TALENT...	A Touching Tal...	2006	1	NULL	150	PG-13	15/02/2006 05:0...
74	BIRCH ANTITR...	A Fanciful Pano...	2006	6	NULL	162	PG	15/02/2006 05:0...
75	BIRD INDEPEN...	A Thrilling Doc...	2006	6	NULL	163	G	15/02/2006 05:0...
76	BIRDCAGE CAS...	A Fast-Paced S...	2006	6	NULL	103	NC-17	15/02/2006 05:0...

Percebemos que a coluna `language_id` possui valor 1, que corresponde ao idioma inglês. Porém, a coluna `original_language_id` está com valor nulo.

É algo que devemos ficar atentos no momento do mapeamento: a coluna `original_language_id` aceita valores nulos.



É exatamente pela nulidade da coluna que a aplicação não ocorreu como deveria. Estamos considerando na nossa configuração uma shadow property `not null`, sabemos disso pela convenção de nulidade do Entity. O tipo `byte` não permite valores nulos.

Para que a aplicação ocorra de forma correta, precisaremos ajustar o mapeamento da coluna `original_language_id`, sabendo que ela permite nulidade.

Para definirmos a coluna usaremos o mesmo método de definição para tipos primitivos, ou seja, além dos valores padrões do tipo, queremos que seja considerado o valor `null`. Para isso, inserimos o caractere `?`, dessa forma transformamos o tipo em um *nulable type*. Vejam o exemplo:

```
byte? idade = null;
```

na classe `FilmeConfiguration` faremos a alteração da shadow property com relação à nulidade inserindo o caractere `?`.

```
namespace Alura.Filmes.App.Dados
{
    public class FilmeConfiguration : IEntityTypeConfiguration
    {
        public void Configure(EntityTypeBuilder<Filme>builder)
        {

            modelBuilder.Entity<Filme>()
                .ToTable("film")

            modelBuilder.Entity<Filme>()
                .Property(f => f.id)
                .HasColumnName("film_id");

            modelBuilder.Entity<Filme>()
                .Property(f => f.Titulo)
                .HasColumnName("title")
                .HasColumnType("varchar(255)")
                .IsRequired();

            modelBuilder.Entity<Filme>()
                .Property(f => f.Descricao)
                .HasColumnName("description")
                .HasColumnType("text");

            modelBuilder.Entity<Filme>()
                .Property(f => f.AnoLancamento)
                .HasColoumnName("release_year")
                .HasColumnTypoe("varchar(4)");

            modelBuilder.Entity<Filme>()
                .Property(f => f.Duracao)
                .HasColumnName("length")

            modelBuilder.Entity<Filme>()
                .Property<DateTime>("last_update")
                .HasColumnType("datetime")
                .IsRequired();

            builder.Property<byte>("language_id");
            builder.Property<byte?>("original_language_id");

            builder
                .HasOne(f => f.IdiomaFalado);
                .WithMany(i => i.FilmesFalados)
                .HasForeignKey("language_id");

            builder
                .HasOne(f => f.IdiomaOriginal)
                .WithMany(i => i.FilmesOriginais)
                .HasForeignKey("original_language_id");
```

```
        }
    }
}
```

Ao executarmos novamente a aplicação, perceberemos que não haverá ocorrência de erro. É importante prestarmos muita atenção na nulidade das tabelas no momento do mapeamento. Caso trabalhemos com uma coluna que esteja vazia o erro ocorrerá posteriormente a medida em que os dados vão sendo incluídos.

Iremos voltar à primeira organização do código da classe `Program`.

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

                var idiomas = contexto.Idiomas
                    .Include(i => FilmesFalados);

                foreach (var idioma in idiomas)
                {
                    Console.WriteLine(idioma);
                }
            }
        }
    }
}
```

O próximo objetivo é mostrar os filmes dublados em um determinado idioma. Faremos um novo `foreach` para cada filme dentro da instância `idioma` na coleção `FilmesFalados` e mostraremos o respectivo filme no console.

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

                var idiomas = contexto.Idiomas
                    .Include(i => FilmesFalados);

                foreach (var idioma in idiomas)
                {
                    Console.WriteLine(idioma);

                    foreach (var filme in idioma.FilmesFalados)
```

```
                    {
                        Console.WriteLine(filme);
                    }
                    Console.WriteLine("\n");
                }

            }
        }
    }
}
```

Ao executarmos o programa veremos, primeiramente, todos os filmes que possuem inglês como idioma falado, depois italiano(não há nenhum filme falado em italiano no nosso banco de dados), japonês e assim sucessivamente.

```
Idioma (1): Engish
Filme (115): CAMUPUS REMEMBER - 2005
Filme (116): CANDIDATE PERDITION - 2006
Filme (117): CANDELS GRAPES - 2006

(...)

idioma(2): Italian

Idioma(3): Japanese
filme(21): AMERICAN CIRCUS - 2006
filme(22): AMISTAD MIDSUMMER - 2006
filme(23): ANACONDA CONFESSIONS - 2006
```

Conseguimos mapear corretamente a relação entre idiomas e filmes. Finalizaremos gerando a migração do relacionamento entre `Idioma` e `Filmes`.

No mapeamento de `Idioma` havíamos criado apenas a tabela `language`, mas queremos que nessa mesma migração sejam incluídas as chaves estrangeiras na tabela de `Filmes`.

```
public partial class Idioma: Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "language",
            columns> table => new
            {


<!-- ... -->
```

Iremos acessar o console NuGet para remover a migração através do comando `Remove-Migration`. Feito isso, adicionaremos a migração de `Idioma` através do comando `Add-Migration-Idioma`. Essa migração atualizada contará com a tabela `language` e as chaves estrangeiras na tabela de `Filme`.

```
public partial class Idioma : Migration
{
```

```
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<byte>(
                name: "language_id",
                table: "film",
                type: "tinyint",
                nullable: false
                defaultValue: (byte)0);

            migrationBuilder.AddColumn<byte>(
                name: "original_language_id",
                table: "film",
                type: "tinyint",
                nulable: true);

            migrationBuilder.CreateTable(
                name: "language",
                columns: table => new
                {
                    language_id = table.Column<byte>(type: "tinyint", nullable: false),
                    name = table.Column<string>(type: "char(20)", nullable: false),
                    last_update = table.Column<DateTime>(type: "datetime", nullable: false, default
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_language", x => x.language_id);

                });

            migrationBuilder.CreateIndex(
                name: "IX_film_original_langauge_id",
                table: "film",
                column: "original_language_id");

            migrationBuilder.CreateIndex(
                name: "IX_film_original_language_id",
                table: "film",
                column: "original_language_id");

            migrationBuilder.AddForeignKey(
                name: "FK_film_language_language_id",
                table: "film",
                column: "language_id",
                principalTable: "language",
                principalColumn: "language_id",
                onDelete: ReferentialAction.Cascade);

            migrationBuilder.AddForeignKey(
                name: "FK_film_language_original_language_id",
                table: "film",
                column: "original_language_id",
                principalTable: "language",
                principalColumn: "language_id",
                onDelete: ReferentialAction.Restrict);


    <!-- ... -->
```

Iremos aplicar a migração ao banco de dados `AluraFilmesTST`. Para isso, apontamos para este banco de dados na classe `AluraFilmesContexto`.

```
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 }
        public DbSet<Idioma> Idiomas { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(Server=(localdb)mssqllocaldb;Database=AluraFilmesTST;Trustec
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new AtorConfiguration());
            modelBuilder.ApplyConfiguration(new FilmeConfiguration());
            modelBuilder.ApplyConfiguration(new FilmeAtorConfiguration());
            modelBuilder.ApplyConfiguration(new IdiomaConfiguration());
        }
    }
}
```

Feito isso, acionaremos o comando `Update-Database` no gerenciador NuGet. Temos todas as informações do banco de dados legado `AluraFilmes` no novo banco de dados `AluraFilmesTST`.