03

Logando o SQL no console

Transcrição

```
insira seu código aqui
```

Na aula anterior, vimos que quando colocamos o Entity para cuidar das entidades, ele passa a monitorar o estado desses objetos. Quando buscamos dados no banco, o estado veio como ***Unchanged***, ao alterarmos alguma informação do dado, ele passou a ser ***Modified***.

O que faremos durante essa parte, não é necessário que você repita. O intuito é mostrar o SQL que o Entity está gerando e qual a relação desse SQL com o estado atual dos objetos.

Primeiro vamos configurar alguns detalhes na class `Program`. Dentro do método `Main()` vamos adicionar o provedor de serviços `GetInfrasctructure<IServiceProvider>()`, lembrando de adicionar o *namespaces*.

```
class Program
{
    static void Main(string[] args)
    {
        using (var contexto = new LojaContext())
        {
            var serviceProvider = contexto.GetInfrasctructure<IServiceProvider>();

            // ...
        }
    }
}
```

Pediremos para o `serviceProvider` fornecer um serviço específico que cria ***Loggers***.

```
class Program
{
    static void Main(string[] args)
    {
        using (var contexto = new LojaContext())
        {
            var serviceProvider = contexto.GetInfrastructure<IServiceProvider>();
            var loggerFactory = serviceProvider.GetService<ILoggerFactory>();

            // ...
        }
    }
}
```

Com o `loggerFactory` passaremos para o método `AddProvider()` um *Logger* específico que chamaremos de `SqlLoggerProvider`. O `SqlLoggerProvider` chama um método chamado `Create()`.

```
class Program
{
    static void Main(string[] args)
    {
        using (var contexto = new LojaContext())
        {
            var serviceProvider = contexto.GetInfrasctructure<IServiceProvider>();

            var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
            loggerFactory.AddProvider(SqlLoggerProvider.Create());

            // ...
        }
    }
}
```

Porém a classe `SqlLoggerProvider` não existe, vamos criá-la.

> Lembrando que todos esses passos não são necessários que sejam feitos, é apenas para mostrar como o SQL que o Entity está gerando.

Criaremos a classe código `SqlLoggerProvider` e deixaremos da seguinte maneira:

```
using Microsoft.EntityFrameworkCore.Storage;
using Microsoft.Extensions.Logging;
using System;

namespace Alura.Loja.Testes.ConsoleApp
{
    public class SqlLoggerProvider : ILoggerProvider
    {
        public static ILoggerProvider Create()
        {
            return new SqlLoggerProvider();
        }

        public ILogger CreateLogger(string categoryName)
        {
            if (categoryName == typeof(IRelationalCommandBuilderFactory).FullName)
            {
                return new SqlLogger();
            }
            return new NullLogger();
        }

        public void Dispose()
        {

        }
    }

    internal class NullLogger : ILogger
    {
        public IDisposable BeginScope<TState>(TState state)
        {
```

```
                return null;
            }

            public bool IsEnabled(LogLevel logLevel)
            {
                return true;
            }

            public void Log<TState>(LogLevel logLevel, EventId eventId, TState state, Exception exceptic
            {
                //não faz nada
            }
        }

        public class SqlLogger : ILogger
        {
            public IDisposable BeginScope<TState>(TState state)
            {
                return null;
            }

            public bool IsEnabled(LogLevel logLevel)
            {
                return true;
            }

            public void Log<TState>(LogLevel logLevel, EventId eventId, TState state, Exception exceptic
            {
                Console.WriteLine("");
                Console.WriteLine(formatter(state, exception));
                Console.WriteLine("");
            }
        }
    }
```

Executando a aplicação veremos o SQl:

```
SELECT [p].[Id], [p].[Categoria], [p].[Nome], [p].[Preco]
FROM [Produtos] AS [p]
```

Mostrando também o estado de cada dado. Como o último item da lista estamos modificando o nome e não salvamos, o seu estado é *Modified*. O que acontecerá quando salvarmos essas alterações?

Tiraremos os comentários do comando `contexto.SaveChanges()` , que está **depois** da alteração que fizemos no nome do último elemento da lista.

```
static void Main(string[] args)
{
    using(var contexto = LojaContext())
    {

        var serviceProvider = contexto.GetInfrasctructure<IServiceProvider>();
        var loggerFactory = serviceProvider.GetService<ILoggerFactory>();
        loggerFactory.AddProvider(SqlLoggerProvider.Create());
```

```
            var produtos = contexto.Produtos.ToList();
            foreach (var p in produtos)
            {
                Console.WriteLine(p);
            }

            Console.WriteLine("=================");
            foreach (var e in contexto.ChangeTracker.Entries())
            {
                Console.WriteLine(e);
            }

            var p1 = produtos.Last();
            p1.Nome = "007 - O Espiao Que Me Amava";


            Console.WriteLine("=================");
            foreach (var e in contexto.ChangeTracker.Entries())
            {
                Console.WriteLine(e);
            }

            contexto.SaveChages();

            //Console.WriteLine("=================");
            //produtos = contexto.Produtos.ToList();
            //foreach (var p in produtos)
            //{
            //    Console.WriteLine(p);
            //}
        }
    }
```

Executando a aplicação novamente vemos que após apresentar o a mensagem dos estados, foi gerada um SQL para atualizar no banco de dados.

```
UPDATE [Produtos] SET [Nome] = @p0
WHERE [Id] = @p1;
```

Então, quando chamamos o `contexto.SaveChages()`, ele olha para cada objeto e verifica o seu estado. Caso seja um estado que é preciso sincronizar com o banco, ele executará o SQL. Para o estado "Modified", ele executa o SQL Update.

No próximo vídeo veremos quais são os outros estados.