02

# Introdução ao Entity Framework

### Transcrição

Você deve estar se perguntando, o que é preciso para fazer esse curso com tranquilidade? O primeiro passo é ter o conhecimento de C# e Orientação à Objetos. A Alura possui três cursos com a linguagem C#.

Cursos de C# da Alura:

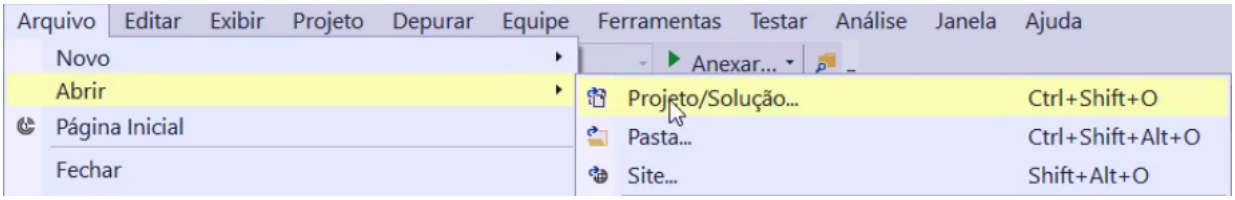
- C# I: Fundamentos da linguagem (https://cursos.alura.com.br/course/fundamentos-de-csharp)
- C# II: Orientação a objetos (https://cursos.alura.com.br/course/csharp-orientacao-a-objetos)
- C# III: Tópicos Avançados (https://cursos.alura.com.br/course/csharp-topicos-avancados)

O segundo passo é baixar e instalar o Visual Studio 2017 (https://www.visualstudio.com/pt-br/). Com a instalação da IDE, também virá o **SQL Server Express**, que é o banco de dados utilizado no projeto do curso.

## Por que Entity Framework?

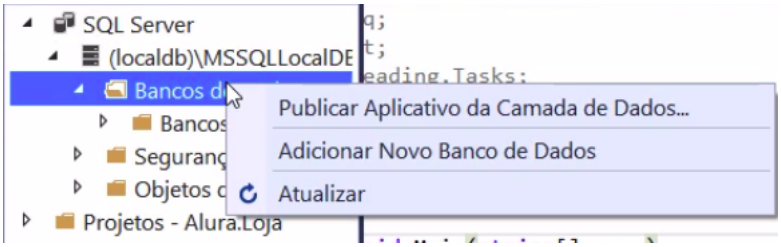
Para responder o motivo de usar o Entity, é necessário entender as dificuldades que existiam antes do Entity. Após baixar o projeto do curso - que será encontrada nos exercícios -, descompactaremos e colocaremos na pasta de projetos do Visual Studio.

Com o Visual Studio aberto, vamos em "Arquivo > Abrir > Projeto/Solução":




Agora dentro da pasta do projeto `Alura.Loja`, selecionaremos o arquivo `Alura.Loja.sln`. Essa Solução é do tipo ***Console Application***, onde veremos o acesso ao banco de dados sem o *Entity Framework*.

Antes é necessário criar o banco de dados para efetuar a conexão. No Visual Studio, clicamos em "Exibir > Pesquisador de Objetos do SQL Server". Uma barra na lateral esquerda será aberta com a opção ***SQL Server***. Ao expandir a opção *SQL Server*, temos a pasta `Banco de Dados`, clicaremos com o botão direito do mouse e selecionamos a opção "Adicionar Novo Banco de Dados".




Na opção **Nome do Banco de Dados**, colocaremos `LojaDB`, que é o domínio da aplicação, em seguida clicamos em "OK". A seguir podemos ver na barra que o banco foi criado, porém ainda não temos nenhuma tabela. Criaremos a tabela de

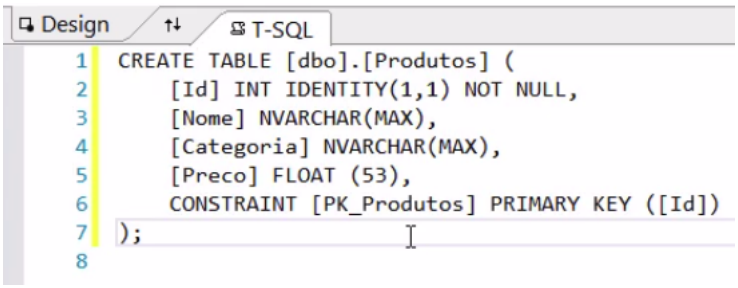
produtos.

Dentro do projeto do curso existe um arquivo chamado `ddl-produtos.txt`, que contém o *script* de criação da tabela.

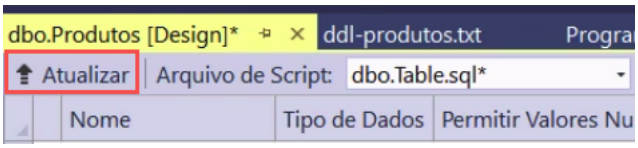
```
CREATE TABLE [dbo].[Produtos] (
    [Id] INT IDENTITY(1,1) NOT NULL,
    [Nome] NVARCHAR(MAX),
    [Categoria] NVARCHAR(MAX),
    [Preco] FLOAT (53),
    CONSTRAINT [PK_Produtos] PRIMARY KEY ([Id])
);
```

Nesse momento vemos a primeira diferença, antes do Entity, é necessário manter os *scripts* de criação e atualização da estrutura de dados.

Após copiar o *script* com "Ctrl + C", clicaremos com o botão direito do mouse em "Tabelas", e selecionaremos a opção "Adicionar Nova Tabela". Colaremos o *script* na aba `T-SQL`, o Visual Studio irá processar automaticamente e mostrar a tabela.




Pressionamos em "Atualizar", localizado em cima da coluna `Nome` do banco de dados.




Em seguida, na nova janela aberta clicaremos em "Atualizar Banco de Dados". Pronto! Podemos ver no campo esquerdo, dentro de "Tabelas" a tabela "dbo.Produtos" criada. Fecharemos a aba de criação de tabelas.

Como a nossa aplicação está fazendo acesso aos dados sem usar o Entity? No projeto, isolamos toda a responsabilidade de acesso aos dados com o banco em uma classe específica. Essa prática, é um padrão de arquitetura chamada de ***Data Access Object***. Como estamos acessando os dados relacionados a classe `Produto`, essa responsabilidade ficou na classe `ProdutoDAO`. Vamos acessar essa classe.

Com a classe `Program.cs` aberta:

```
namespace Alura.Loja.Testes.ConsoleApp
{

    class Program
    {
        static void Main(String [] args)
        {
            GravarUsandoAdoNet();
        }
```

```
        private static void GravarUsandoAdoNet()
        {
            Produto p = new Produto();
            p.Nome = "Harry Potter e a Ordem da Fênix";
            p.Categoria = "Livros";
            p.Preco = 19.89;

            using (var repo = new ProdutoDAO())
            {
                repo.Adicionar(p);
            }
        }
    }
}
```

Podemos colocar o cursos sobre a chamada `repo.Adicionar(p);` e teclar "F12". Dessa forma, seremos direcionados diretamente para o método `Adicionar()` na classe `ProdutoDAO`. Repare que no método `Adicionar()`, a comunicação com o banco de dados é feita por meio da linguagem SQL. Para esse curso, não é necessário ter conhecimentos avançados de SQL, mas caso se interesse pelo assunto, a Alura tem a carreira Iniciando com SQL e MySQL (https://cursos.alura.com.br/career/iniciando-com-sql-e-mysql).

Sem o Entity, nós somos os responsáveis por montar o SQL na mão. Colocamos uma string com o comando que insere no banco de dados, passando algumas marcações que serão substituídas por parâmetros. No final executamos essa SQL.

```
internal void Adicionar(Produto p)
{
    try
    {
        IDbCommand insertCmd = conexao.CreateCommand();
        insertCmd.CommandText = "INSERT INTO Produtos (Nome, Categoria, Preco) VALUES (@nome, @categ

        IDbDataParameter paramNome = new SqlParameter("nome", p.Nome);
        insertCmd.Parameters.Add(paramNome);

        IDbDataParameter paramCategoria = new SqlParameter("categoria", p.Categoria);
        insertCmd.Parameters.Add(paramCategoria);

        IDbDataParameter paramPreco = new SqlParameter("preco", p.Preco);
        insertCmd.Parameters.Add(paramPreco);

        insertCmd.ExecuteNonQuery();
    } catch (SqlException e)
    {
        throw new SystemException(e.Message, e);
    }
}
```

Além do método `Adicionar()`, temos também na classe `ProdutoDAO` métodos para atualizar, remover e recuperar produtos da base de dados. Esses comandos SQL são representados através de interfaces, que estão definidas no *namespace* `System.Data`, para representar o componente ADO.NET da Microsoft.

Uma das interfaces que faz parte desse componente é a `IDbConnection`, que usamos para conexão com o banco. Para implementar essa conexão, usamos a classe `SqlConnection` que fica no *namespace* `System.Data.SqlClient`. A classe

`SqlConnection` representa um conexão com o SQL Server, caso deseje usar de algum outro banco de dados é necessários usar outra classe, cada banco possui a sua própria classe de conexão.

Se a aplicação não sofresse mudanças, não seria necessário usarmos o Entity. Se o cliente solicitar que seja removido o campo `categoria` da classe `Produto`, teríamos que tratar todas as SQL's, caso contrário não conseguiríamos compilar o código.

A aplicação ainda é pequena e já tivemos um grande trabalho. Imagine um aplicação enorme com várias classes de negócio, o impacto seria enorme, o que dificultaria a evolução da aplicação.

O ideal seria que alguém ficasse responsável pelo trabalho de criar as SQL's, passar parâmetros e assim por diante. É aí que entra o Entity Framework.

No próximo vídeo veremos como instalar Entity Framework.