

FIA/P GRADUAÇÃO

# ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#06 - ENTITY FRAMEWORK CORE

# TRAJETÓRIA

---



- ✓ Plataforma .NET
- ✓ Linguagem C# e Orientação a Objetos
- ✓ ASP.NET Core – Rotas e Controller
- ✓ ASP.NET Core – Razor e Tag Helpers
- ✓ ASP.NET Core – Layout e Partial Views
- ✓ Entity Framework Core

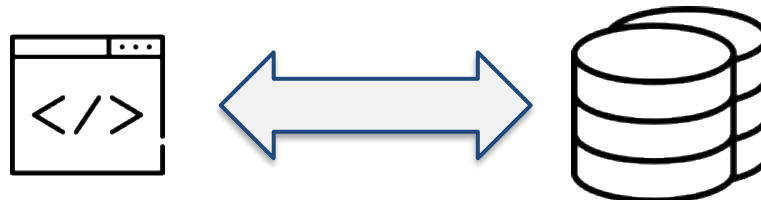
# #06 - AGENDA

---

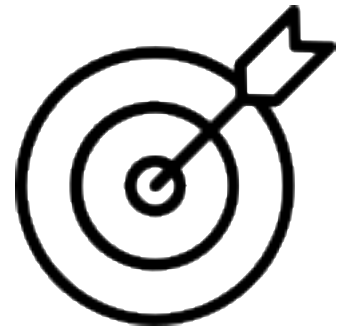


- ORM
- Frameworks de ORM
- Entity Framework Core
- DbContext e DbSet
- SQL Server e String de conexão
- Injeção de dependência
- Migrations
- Manipulando as entidades – CRUD
- Data Annotations

- ORM é uma técnica de **mapeamento objeto relacional** que visa desenvolver uma camada de mapeamento entre o **modelo de objetos** (sistema) e o **modelo relacional** (banco de dados);
- Existem diversos **frameworks ORMs**, que facilitam nas tarefas de **persistência e recuperação de dados**, aumentando a produtividade do desenvolvimento;
- **Frameworks ORMs** estão presentes nas maiorias das plataformas e **linguagens de programação**;



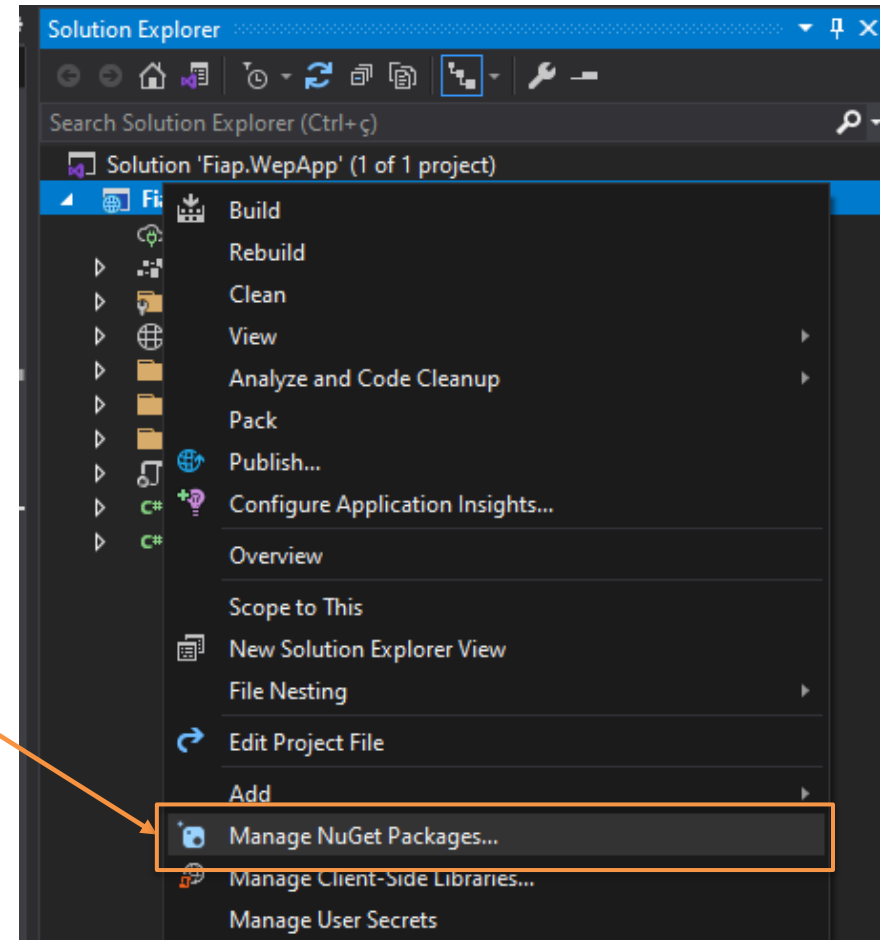
- Na **plataforma .NET** existem diversas soluções para ORM, as mais conhecidas são **Entity Framework** e **NHibernate**;
- Existem também os **micros-ORM** como o **Dapper** que trabalha com **ADO.NET** e disponibiliza **Extensions Methods** que simplificam o desenvolvimento com banco de dados;
- Vamos trabalhar com o **Entity Framework Core**, que é uma versão leve, extensível e multiplataforma;





# ENTITY FRAMEWORK CORE

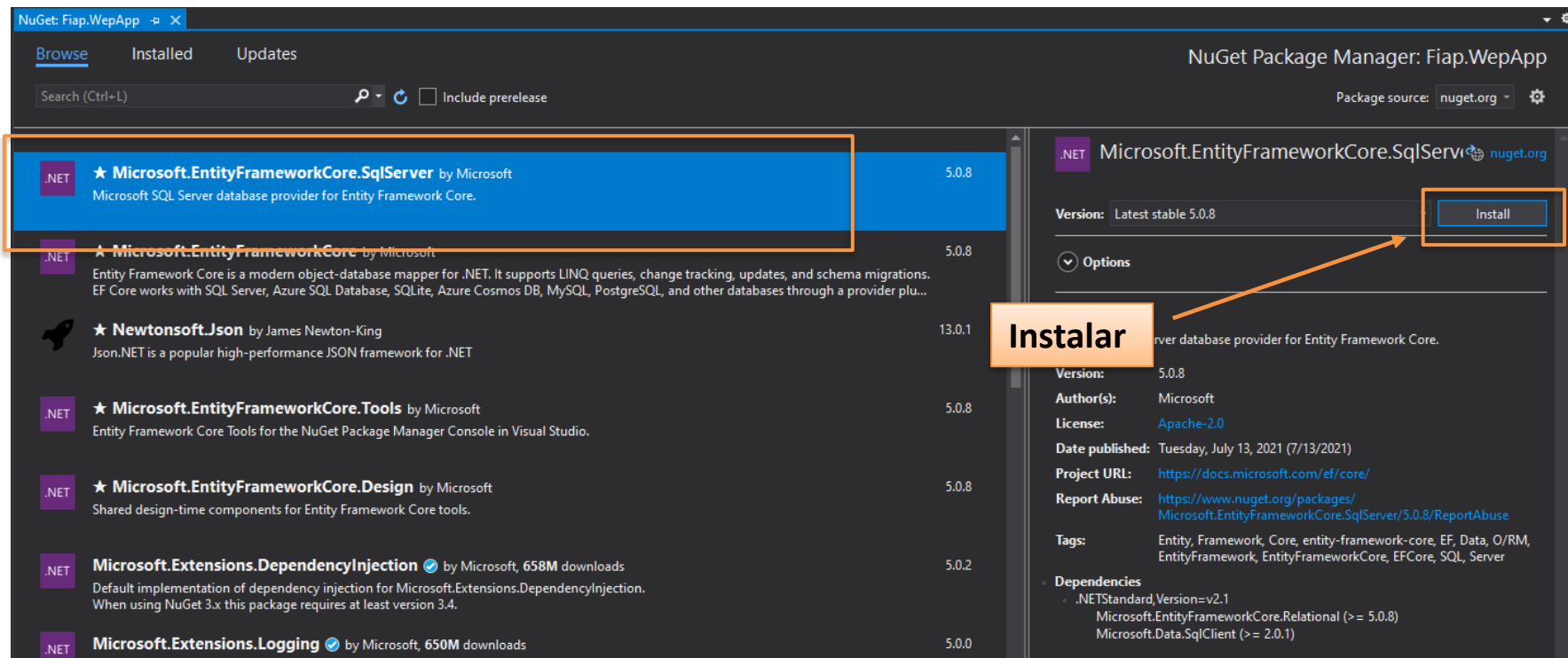
- O Entity Framework Core deve ser instalado no projeto, para isso clique com o botão direito do mouse e escolha “**Manage NuGet Packages..**”



Gerenciar as dependências do projeto



- Instale o pacote “Microsoft.EntityFrameworkCore.SqlServer” e “Microsoft.EntityFrameworkCore.Tools”



## ▪ “Microsoft.EntityFrameworkCore.Tools”

The screenshot shows the NuGet Package Manager interface for the project 'Fiap.WepApp'. The 'Browse' tab is active, and the search filter is set to 'tools'. The package list on the left includes:

- System.Diagnostics.Tools** (4.3.0) by Microsoft, 349M downloads
- Microsoft.EntityFrameworkCore.Tools** (5.0.8) by Microsoft, 100M downloads (highlighted with an orange box)
- runtime.any.System.Diagnostics.Tools** (4.3.0) by Microsoft, 51,1M downloads
- Grpc.Tools** (2.38.1) by The gRPC Authors, 16,4M downloads
- Microsoft.VisualStudio.Azure.Containers.Tools.Targets** (1.10.8) by Microsoft, 43,7M downloads

The right pane shows the details for **Microsoft.EntityFrameworkCore.Tools**. The version is set to 'Latest stable 5.0.8', and the 'Install' button is highlighted with an orange box. An orange arrow points from a label 'Instalar' to this button. Below the 'Options' section, a list of enabled commands is shown:

- Add-Migration
- Drop-Database
- Get-DbContext
- Get-Migration
- Remove-Migration
- Scaffold-DbContext
- Script-Migration
- Update-Database

- É a **classe** que **gerencia** as entidades C# em relação ao banco de dados;
- É através dela que executamos as ações ligadas ao **banco**;
- Para utiliza-la, precisamos criar uma classe que deriva de **Microsoft.EntityFrameworkCore.DbContext**.
- Implemente um **construtor** para instanciar o contexto com algumas opções, como a **string de conexão** com o banco;

```
public class LojaContext : DbContext
{
    public LojaContext(DbContextOptions options)
        : base(options) { }
}
```

- **DbSet<Produto> Produtos** é a propriedade que será utilizada para pesquisar, editar, deletar e salvar (CRUD) a entidade Produto no banco de dados;

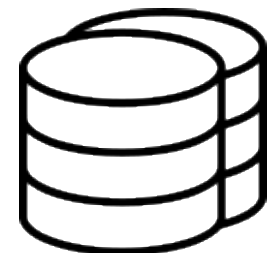
```
public class LojaContext : DbContext
{

    public DbSet<Produto>? Produtos { get; set; }

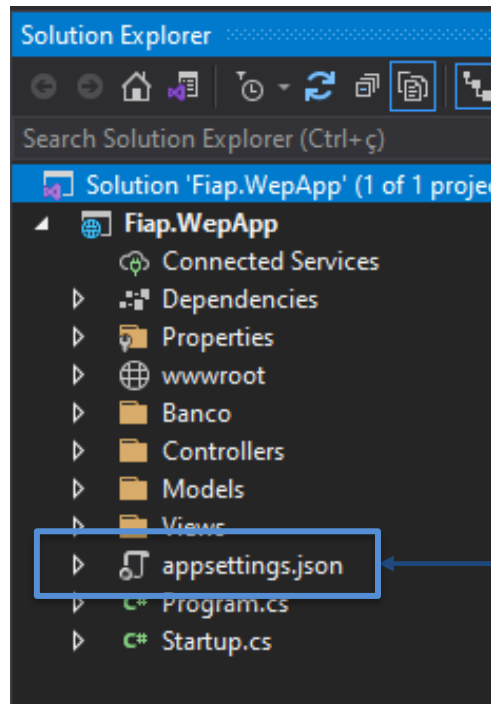
    public LojaContext(DbContextOptions options)
        : base(options) { }

}
```

- O **Entity Framework Core** trabalha com diversos bancos de dados, como **SQL Server**, **MySQL**, **PostgreSQL** e etc.;
- O **SQL Server** é um SGBD da **Microsoft** que é um dos principais do mercado atualmente, com versões pagas e gratuitas;
- O **SQL Server LocalDB** é uma versão leve do **SQL Server Express** voltado para **desenvolvedores**, sem a necessidade de nenhuma configuração complexa, por padrão o banco de dados LocalDB cria arquivos “.mdf”;



- A configuração do banco de dados do projeto ASP.NET Core fica no arquivo **appsettings.json**;



Arquivo de configuração do sistema.

- Arquivo do tipo JSON;
- Podemos definir várias strings de conexão com o banco de dados através da chave “**ConnectionStrings**”;
- A **chave** da **conexão** será utilizada na aplicação para referenciar a string de conexão;

```
{  
  "ConnectionStrings": {  
    "conexao":  
    "Server=(localdb)\\MSSQLLocalDB;Database=LojaDB;Trusted_Connection=True;MultipleActiveResultSets=true"  
  }  
}
```

Nome do **Banco de Dados**

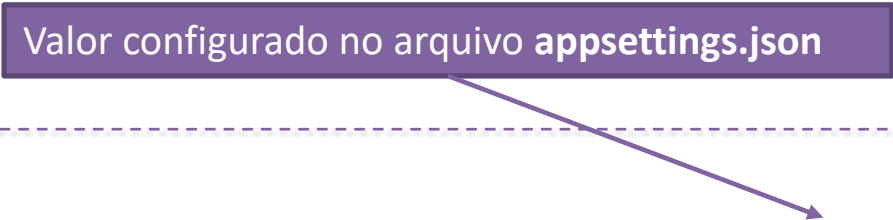
- O **ASP.NET Core** possui por padrão um framework de **Injeção de Dependência**;
- **Injeção de dependência** é um padrão de projetos que visa diminuir o acoplamento entre as classes;
- No **ASP.NET Core** as dependências entre as classes são “injetadas” através do **construtor**, de acordo com a configuração realizada na classe **startup.cs**;





- Dentro da classe **Program.cs**
  - Crie uma variável para recuperar a **string de conexão** configurado no appsettings.json
  - Adicione o serviço de **contexto**, informando a variável com a string de conexão;

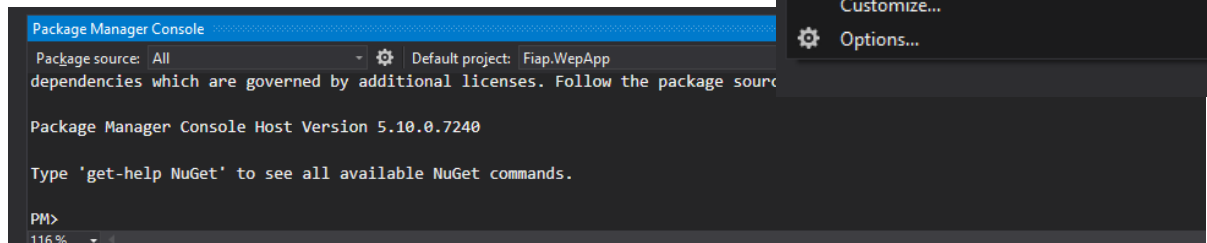
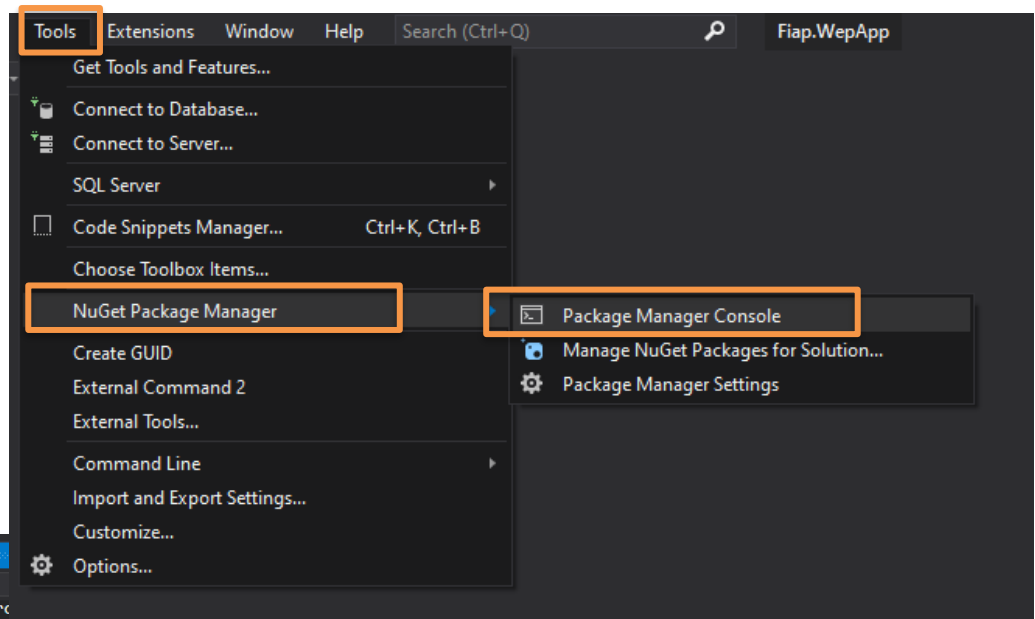
Valor configurado no arquivo appsettings.json



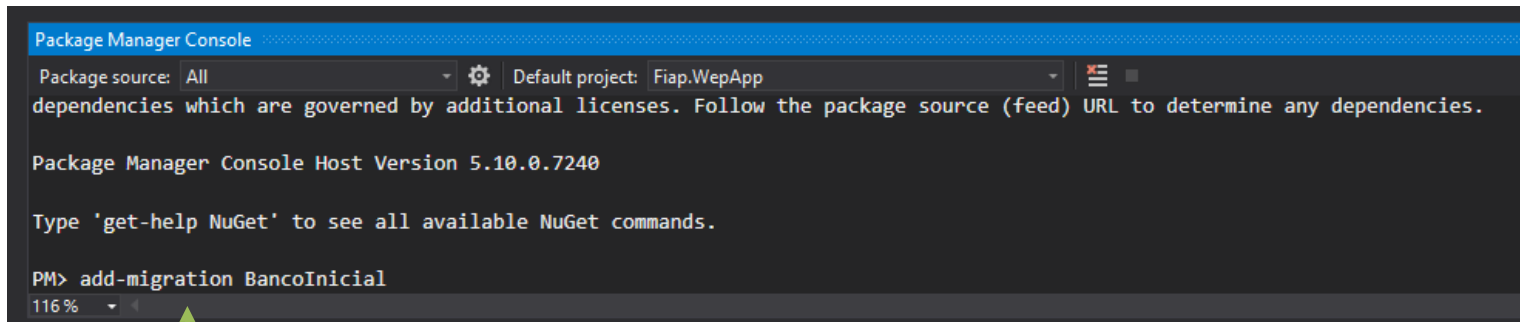
```
var connection =  
    builder.Configuration.GetConnectionString("conexao");  
  
builder.Services.AddDbContext<LojaContext>(options =>  
    options.UseSqlServer(connection));
```

- Vamos utilizar algumas ferramentas do **Entity Framework Core** para **criar o banco de dados** ou realizar modificações;
- Acesse o Package Manager Console: **Tools > NuGet Package Manager> Package Manager Console**;

**Package Manager Console**



- Depois de criar ou modificar as **Entidades**, crie uma **migração** para criar ou aplicar as modificações no **banco de dados**;
- Para isso, utilize o comando **Add-Migration** definindo um nome para a migração;



```
Package Manager Console
Package source: All [v] Default project: Fiap.WepApp [v]
dependencies which are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 5.10.0.7240

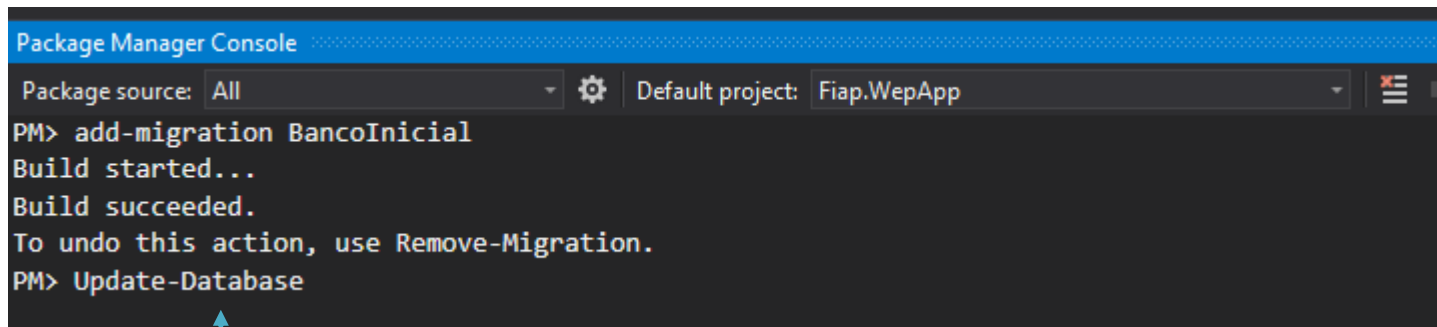
Type 'get-help NuGet' to see all available NuGet commands.

PM> add-migration BancoInicial
116 %
```

**Add-Migration**, utilize o Tab para completar o comando;

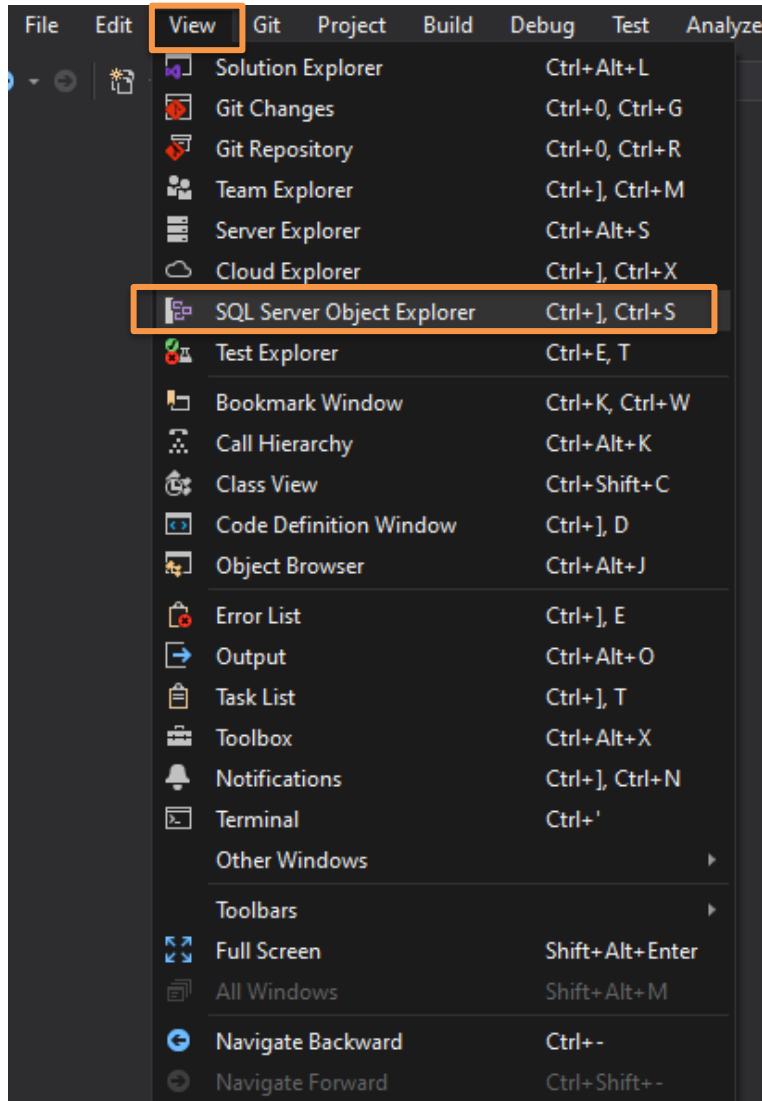
- Após o comando, uma pasta **Migrations** com algumas **classes** serão criadas no projeto;

- Para aplicar a migração no banco de dados utilize o comando Update-Database;

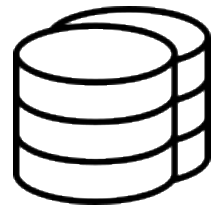


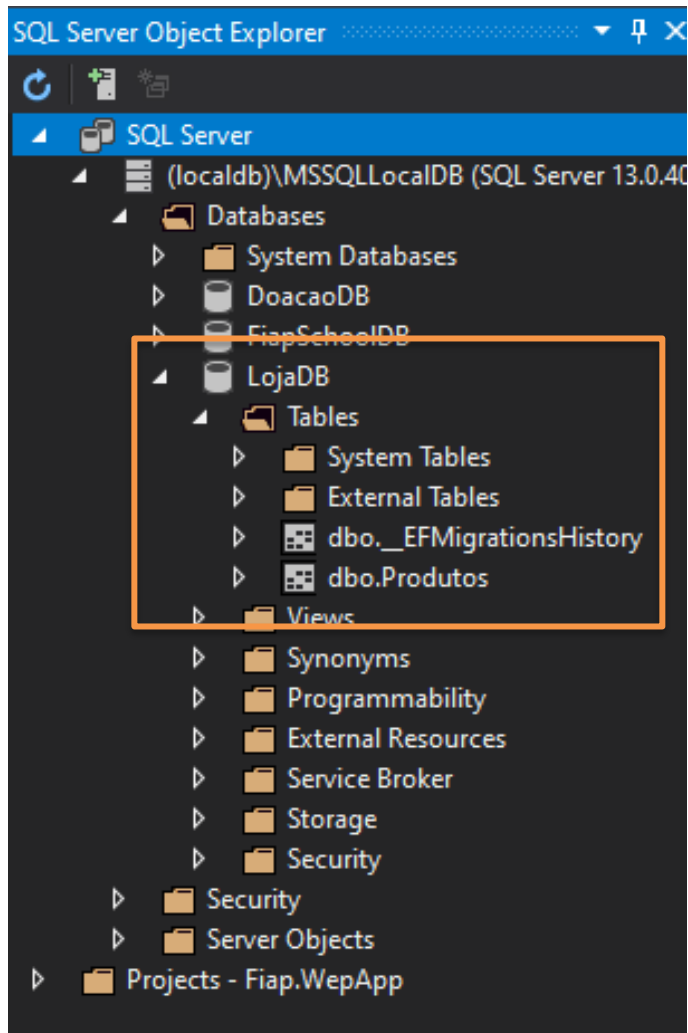
```
Package Manager Console
Package source: All
Default project: Fiap.WepApp
PM> add-migration BancoInicial
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> Update-Database
```

**Update-Database**, utilize o Tab para completar o comando;



- Para acessar o banco de dados, abra a janela SQL Server Object Explorer, localizado em View > SQL Server Object Explorer;





- Navegue até o banco de dados e abra a pasta **Tables** para visualizar as tabelas;
- Para **exibir as informações** gravadas em uma **tabela**, clique com o botão direito do mouse na tabela e escolha: **View Data**;



# MANIPULANDO AS ENTIDADES

- O método `DbSet.Add()` adiciona o objeto ao **contexto** e **cadastra as informações** no banco de dados após executar o método `DbContext.SaveChanges()`;

```
Produto produto = new Produto()  
{  
    Nome = "Livro", Valor = 10  
};  
context.Produtos.Add(produto);  
context.SaveChanges();
```





- Para recuperar um registro do banco de dados pela chave primária da tabela, utilize o método `DbSet.Find()`, onde o parâmetro é o identificador da classe;

```
var produto = context.Produtos.Find(1);
```



- Para **remover** uma entidade podemos utilizar o método `DbSet.Remove()`, onde o parâmetro é o **objeto da entidade** e não a chave primária;
- Após o método `DbContext.SaveChanges()` for chamado, a entidade é **removida** do banco de dados;

```
context.Produtos.Remove(produto);  
context.SaveChanges();
```



- Após **pesquisar** ou **cadastrar** um objeto, caso alguma **propriedade** for **alterada**, o objeto é atualizado no banco de dados após chamar o método **DbContext.SaveChanges()**;

```
var produto = context.Produtos.Find(id);  
produto.Nome = "Novo nome";  
context.SaveChanges();
```



- Em uma aplicação **ASP.NET Core**, o objeto que recebemos da página é um **novo objeto** com os valores do formulário, assim não é o mesmo objeto que foi pesquisada no banco de dados;
- Dessa forma, é preciso utilizar o método **DbSet.Update()** para atualizar o banco de dados após o método **DbContext.SaveChanges();**

```
context.Produtos.Update(produto);  
context.SaveChanges();
```

- Para recuperar todos os registros de uma tabela utilize o método **DbSet.ToList()**;

```
var lista = context.Produtos.ToList();
```

- **LINQ - Language Integrated Query** é um componente do .NET que permite efetuar **consultas de propósito geral em coleções**, com uma sintaxe parecida com SQL.
- **Principais métodos:**
  - **Where** - aplica um filtro na pesquisa;
  - **Select** - determina os dados que serão retornados da pesquisa;
  - **ToList** - retornar os dados, de acordo com as configurações;
  - **OrderBy** - ordena o resultado;
  - **FirstOrDefault** - retorna o primeiro elemento ou null;
  - **Count** - retorna o número de elementos;
  - **Include** - inclui um relacionamento do resultado da pesquisa;

```
context.Produtos.Where(p => p.Nome.Contains("Livro") && p.Valor > 100).OrderBy(p => p.Nome).ToList();
```

Recupera os produtos que possuem parte do nome a string "Livro" e valor maior do que 100, ordenado por nome

```
context.Produtos.Count();
```

Recupera a quantidade de produtos cadastrados

```
context.Produtos.Where(p => p.ProdutoId == 1)  
.FirstOrDefault();
```

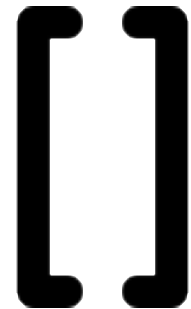
Recupera o primeiro produto (ou null) que possui código igual a 1



# ANNOTATIONS



- É possível utilizar anotações para sobrescrever o mapeamento padrão:
  - Table
  - Key
  - DatabaseGenerated
  - Column
  - MaxLength
  - MinLength
  - StringLength
  - NotMapped
  - Required



- Define a propriedade que será **chave primária** da tabela:

```
[Key]
public int Codigo { get; set; }
```

- Se o nome da propriedade for **Id (ou ID)** ou **nome da classe seguido por Id (ou ID)**, esta propriedade é tratada como chave primária, assim **não é necessário o mapeamento**:

```
public int ProdutId { get; set; }
```

- O valor da propriedade é gerado automaticamente pelo banco de dados:
  - **DatabaseGeneratedOption.Identity:** é gerado um valor para o atributo sempre que a instância for salva pela primeira vez.
  - **DatabaseGeneratedOption.Computed:** é um valor calculado pelo banco de dados. Não é inserido pelo EF. Ex. Uma coluna que é a concatenação do nome e sobrenome.
  - **DatabaseGeneratedOption.None:** não é gerado valor pelo banco de dados.

```
[DatabaseGenerated(DatabaseGeneratedOption.None)]
```

```
public int ProdutoId { get; set; }
```

Não gera valores automáticos para a propriedade ProdutoId

```
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
```

```
public int Numero { get; set; }
```

Gera valores automáticos para a propriedade Numero

- Especifica o **nome da tabela** no banco de dados;

```
[Table("Tbl_Produto")]  
public class Produto  
{  
  
}
```

- Determina o nome da coluna no banco de dados;

```
[Column("Nm_Produto")]  
public string Nome { get; set; }
```

- Define que a coluna é **obrigatória**;

**[Required]**

```
public string Nome { get; set; }
```

- Define o tamanho máximo permitido para a coluna;

```
[MaxLength(50)]  
public string Nome { get; set; }
```



- Define o **tamanho máximo** permitido para a coluna;
- Funciona de forma igual ao **MaxLength**;

```
[StringLength(50)]  
public string Nome { get; set; }
```

- Define que a propriedade **não deve ser mapeada** para o banco de dados;

```
[NotMapped]  
public string Token { get; set; }
```

# VOCÊ APRENDEU..

---



- Sobre ORM;
- Frameworks de ORM e **Entity Framework Core**;
- Implementar o **DbContext** e **DbSets**;
- Definir uma **string de conexão** e configurar a **injeção de dependência** do contexto;
- **Aplicar** a modelagem no banco de dados;
- Realizar as operações básicas (**CRUD**) e **pesquisas** através do Linq;
- Utilizar as **anotações** para configurar o modelo;

# Copyright © 2013 – 2023

## Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

*“Descobri que quanto mais eu estudo, mais sorte eu pareço ter nas provas”*