

FIAAP GRADUAÇÃO

ENTERPRISE APPLICATION DEVELOPMENT

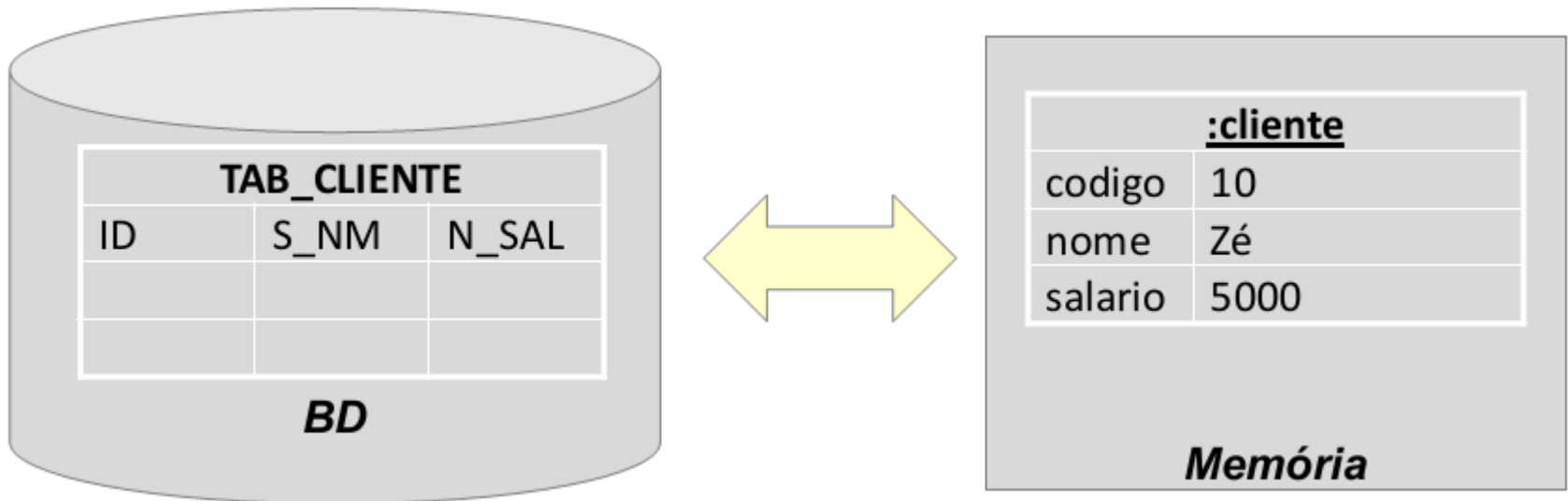
Prof. Me. Thiago T. I. Yamamoto

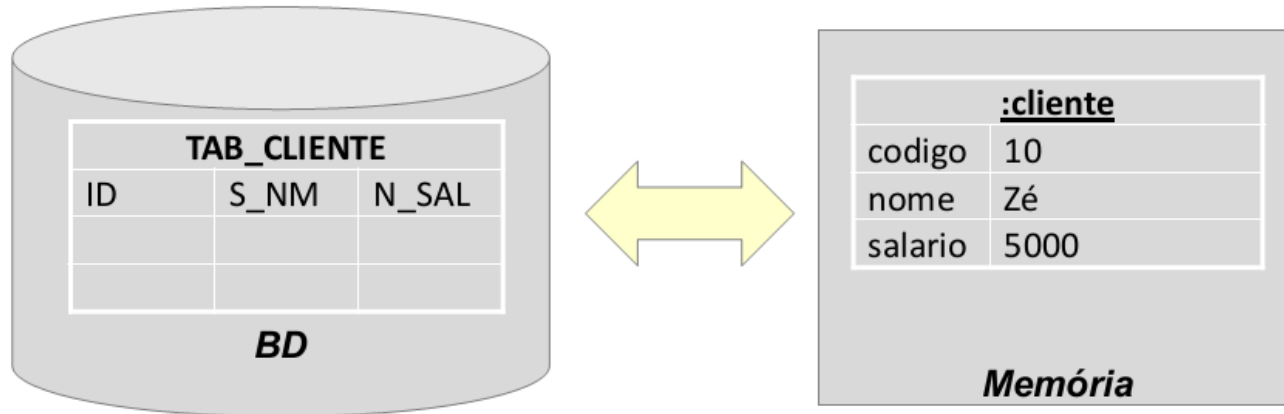
#06 – ENTITY FRAMEWORK



#06 – ENTITY FRAMEWORK

- Entity Framework
- Entity Framework – Formas de utilização
- DbContext
- DbSet
- Configuração do Banco de dados
- Estratégias de Criação de Banco de Dados
- Estados de uma entidade
- Ações básicas:
 - Persistir
 - Buscar
 - Deletar
 - Atualizar
 - Listar
- Data Annotations





Mapeamento:

Cliente -> TAB_CLIENTE

codigo -> ID

nome -> S_NM

salario -> N_SAL

ENTITY FRAMEWORK

- Framework de ORM (Object-Relational Mappers)
- **Entity Framework** permite o mapeamento dos elementos de nossa base de dados para elementos de nossa aplicação orientada a objetos.
- Aumenta a produtividade na persistência e recuperação de dados;
 - Três linhas principais de utilização:
 - » Database First
 - » Model First
 - » Code First



- **Database First:**
 - Primeira modalidade lançada com o EF;
 - Cria as entidades a partir do banco de dados;
- **First Model:**
 - Modela de forma visual o domínio (model) e gera a base de dados a partir dele.
- **Code First:**
 - As classes das entidades são criados antes e a partir delas a base de dados é gerada.

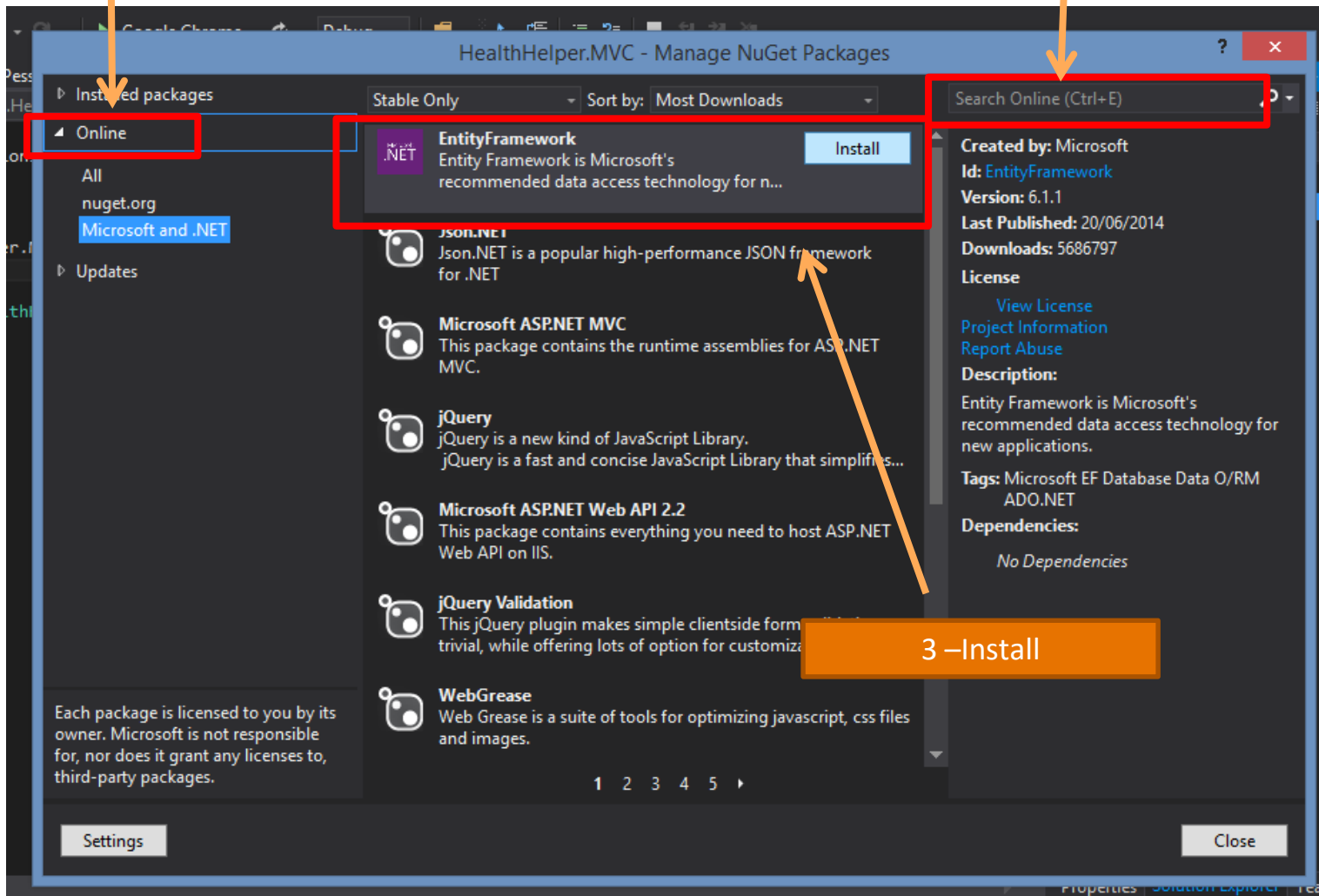
- Vamos utilizar o code first para mapear as nossas entidades em tabelas do banco de dados.
- Ele utiliza pluralize, ou seja, as entidades C# são mapeadas para tabelas no banco de dados com o nome da entidade no plural.

ENTITY FRAMEWORK CONFIGURAÇÃO

ADICIONANDO ENTITY FRAMEWORK NO PROJETO

1 - Seleccione Online

2 - Busque por EntityFramework




3 - Install

- 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 **System.Data.Entity.DbContext**.

```
public class EcommerceContext : DbContext
{
    public DbSet<Cliente> Clientes { get; set; }
}
```

- **DbSet<Cliente> Clientes** é a propriedade que será utilizada para editar, deletar e salvar a entidade Cliente no banco de dados;

```
public class EcommerceContext : DbContext
{
    public DbSet<Cliente> Clientes { get; set; }
}
```

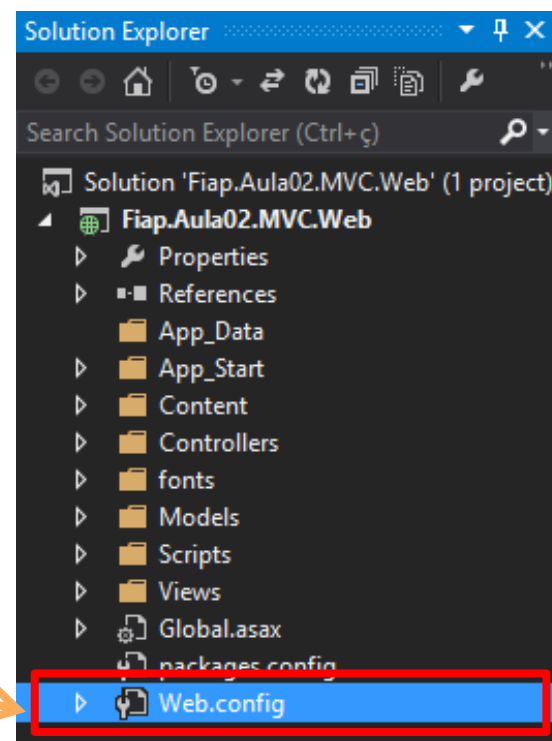


Propriedade que será utilizada para
interagir com o banco de dados

- **SQL Server:** sistema de gerenciamento de banco de dados da Microsoft.
- **SQL Server Local Db:** versão simplificada do SQL Server Express, voltada para os desenvolvedores.

Para configurar o banco de dados, vamos adicionar a String de conexão no arquivo de configuração do projeto.

Abra o arquivo Web.config que está na raiz do projeto



- Adicione a string de conexão dentro da tag <configuration>.

Mesmo nome da classe
DbContext

SQL Server Local Db

```
<connectionStrings>
  <add name="EcommerceContext"
    connectionString="Data Source=(LocalDB)\MSSQLLocalDB;
    AttachDbFilename=|DataDirectory|\Banco.mdf;
    Integrated Security=True"
    providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Mesmo nome da classe
DbContext

SQL Server Express

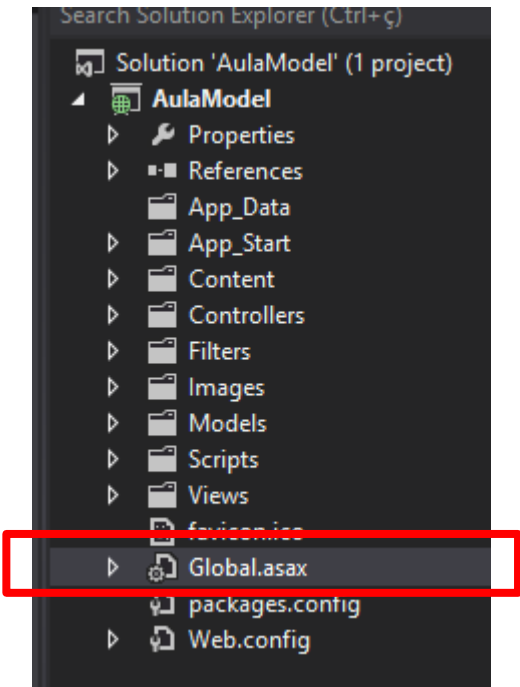
```
<connectionStrings>
  <add name="EcommerceContext"
    connectionString="Data Source=THIAGO\SQLEXPRESS;
    Initial Catalog=FIAP;
    User Id=usuario;
    Password=senha;"
    providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Data Source -> Server Name\Instance Name;
Initial Catalog -> Nome do Banco de Dados;
User Id -> usuário do banco de dados;
Password -> senha do banco de dados;

- Há quatro categorias para criação de base de dados:
 - **CreateDatabaseIfNotExists:** identifica se não existe uma base de dados e então cria uma nova. Essa é a estratégia padrão.
 - **DropCreateDatabaseWhenModelChanges:** cria uma nova base de dados se houver alterações.
 - **DropCreateDatabaseAlways:** sempre cria uma nova base de dados.
 - **Custom DB Initializer:** podemos desenvolver a nossa própria estratégia, caso as outras não sejam suficientes.

CONFIGURAÇÃO EXEMPLO

- O arquivo Global.asax contém código de inicialização que é executado quando a aplicação é inicializada.
- Dentro do método Application_Start, podemos adicionar o código de inicialização do banco de dados:

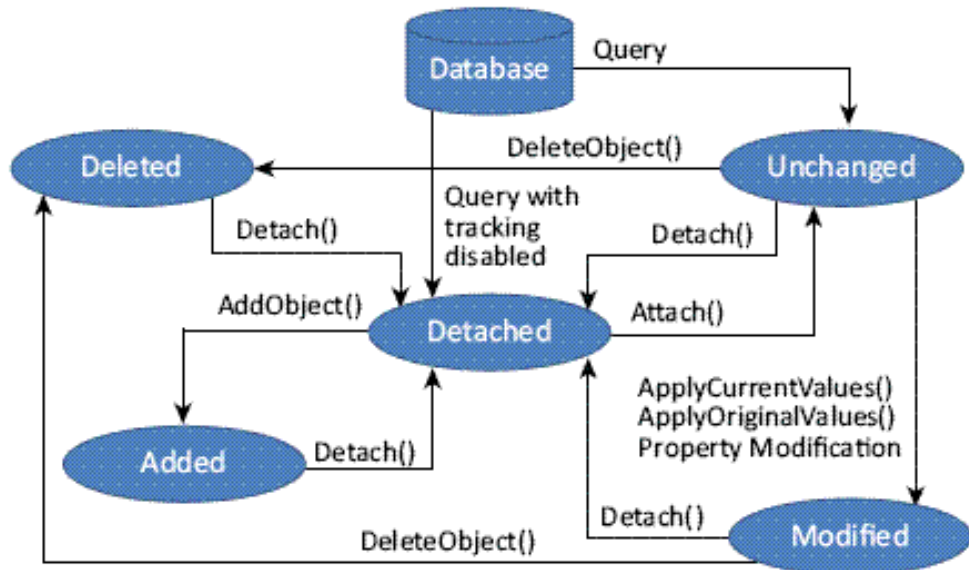


```
protected void Application_Start()
{
    Database.SetInitializer(new DropCreateDatabaseIfModelChanges<EcommerceContext>());
    //Código...
}
```

Apaga e cria o banco de dados caso o model tenha sido alterado

MANIPULANDO AS ENTIDADES

- **Added** – A entidade é marcada para ser adicionada;
- **Deleted** – A entidade é marcada para ser deletada;
- **Modified** – A entidade foi modificada;
- **Unchanged** – A entidade não foi modificada;
- **Detached** – A entidade não está sendo gerenciada pelo contexto;



- Vamos utilizar o método `DbSet.Add()` para adicionar o objeto ao contexto com o estado **Added**.

```
var context = new EcommerceContext();  
Cliente cli = new Cliente();  
cli.Nome = "Thiago Yamamoto";  
context.Clientes.Add(cli);  
context.SaveChanges();
```

Salva as alterações no
Banco de dados

Adiciona o cliente ao
contexto no estado Added

- Podemos recuperar um objeto do banco de dados utilizando o método **DbSet.Find()**, o parâmetro é o identificador da classe.

```
Cliente cliente = context.Clientes.Find(1);
```



Identificador (id) da entidade

- Para remover uma entidade podemos utilizar o método **DbSet.Remove()**, esse método irá colocar o objeto no estado Deleted.

```
Cliente cliente = context.Clientes.Find(1);  
context.Clientes.Remove(cliente);  
context.SaveChanges();
```

Salva as alterações no banco de dados

Coloca o cliente no estado Deleted

- Quando as propriedades de uma entidade do contexto são alteradas, o objeto ganha o estado Modified.

```
Cliente cliente = context.Clientes.Find(1);  
cliente.Nome = "Thiago Yama";  
cliente.Email = "thiagoyama@gmail.com";  
context.SaveChanges();
```

Salva as alterações no
Banco de dados

As propriedades foram
alteradas.

- No ASP.NET, quando recebemos da view a entidade com as alterações, podemos utilizar o método Entry para acessar a entidade no contexto e marca-la como modificada:

```
context.Entry(cliente).State = EntityState.Modified;  
context.SaveChanges();
```

Salva as alterações no Banco de dados

Marca a entidade como modificada.

- Para listar todos os registros de uma tabela, podemos utilizar `DbSet.ToList()`

```
List<Cliente> lista = context.Clientes.ToList();
```

- **LINQ – Language Integrated Query** é um componente do .NET que permite efetuar consultas de propósito geral, com uma sintaxe parecida com SQL.
- Vamos trabalhar com **Extension Method e Expressões Lambdas**.

```
var busca =  
context.Clientes.Where(c => c.Name == nome).ToList();  
  
var usuario =  
context.Clientes.Where(c => c.Name == nome &&  
c.Email == email).FirstOrDefault();
```

ACESSANDO O BANCO DE DADOS

The image shows the Microsoft Visual Studio interface. The 'TOOLS' menu is highlighted with a red box, and the 'Connect to Database...' option is also highlighted with a red box. An orange box labeled 'Tools -> Connect to Database' points to the 'Connect to Database...' option. The 'Add Connection' dialog box is open, showing the 'Data source' as 'Microsoft SQL Server (SqlClient)', the 'Server name' as '(localdb)\v11.0', and the 'Log on to the server' section with 'Use Windows Authentication' selected. Two purple arrows point from labels to the dialog: one from 'Data Source' to the 'Data source' field, and another from 'Autenticação' to the 'Log on to the server' section.

Start Page - Microsoft Visual Studio

FILE EDIT VIEW DEBUG TEAM TOOLS TEST

Tools -> Connect to Database

Connect to Database...

Connect to Server...

Add SharePoint Connection...

SQL Server

Code Snippets Manager...

Choose Toolbox Items...

Add-in Manager...

Library Package Manager

Extensions and Updates...

Create GUID

PreEmptive Dotfuscator and Analytics

Ultimate 2013

Start

New Project...

Open Project...

Open from Source Control...

Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient) Change...

Server name: (localdb)\v11.0 Refresh

Log on to the server

☒ Use Windows Authentication

☐ Use SQL Server Authentication

User name: Password: Save my password

Connect to a database

☒ Select or enter a database name:

☐ Attach a database file: Browse...

Logical name:

Advanced...

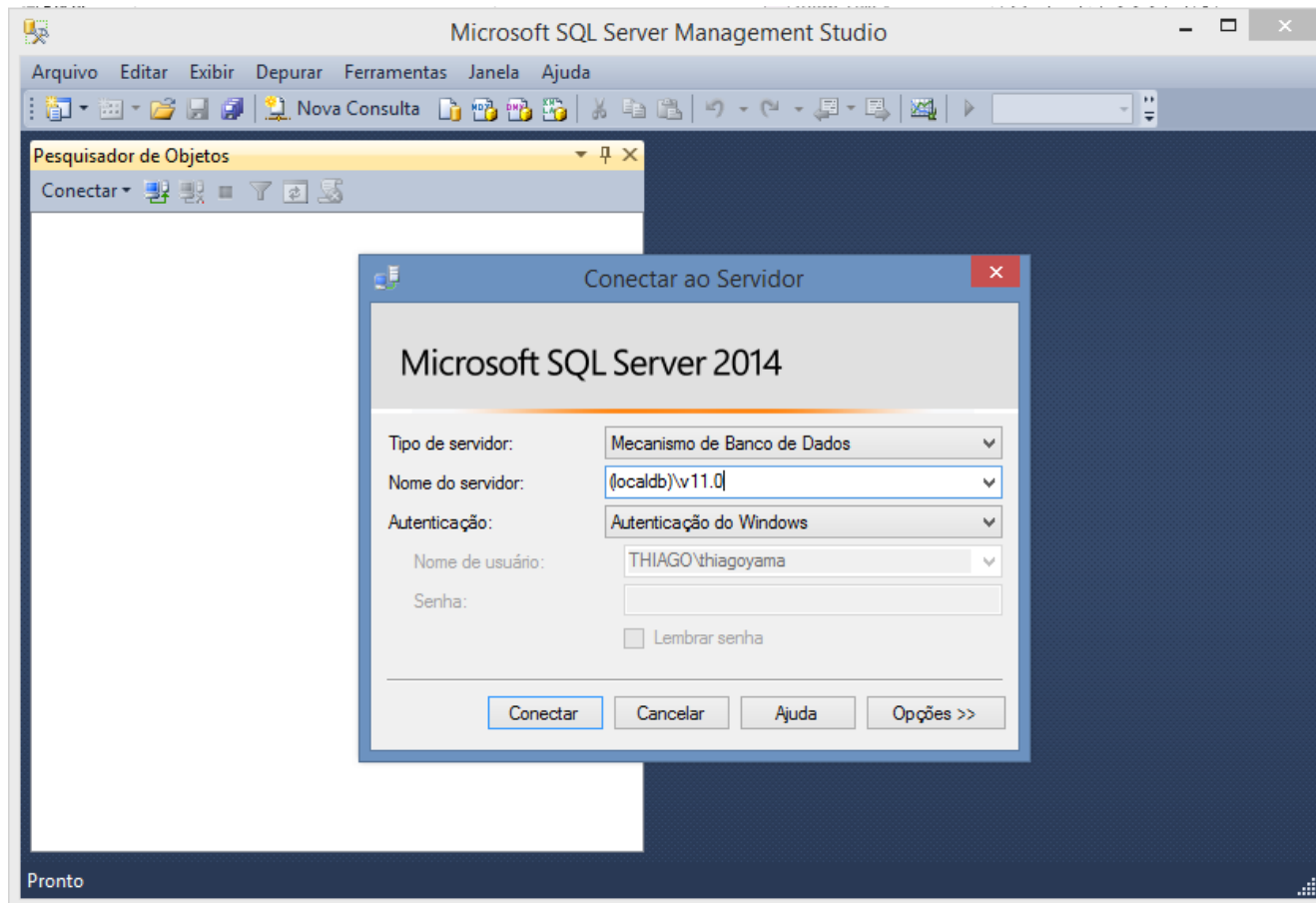
Test Connection OK Cancel

Data Source

Autenticação

ACESSANDO O BANCO DE DADOS

- Podemos utilizar outra ferramenta, como o **Microsoft SQL Server Management Studio**.



ANNOTATIONS

- Podemos utilizar annotations para sobrescrever o mapeamento padrão:
 - Table
 - Key
 - DatabaseGenerated
 - Column
 - MaxLength
 - MinLength
 - StringLength
 - NotMapped
 - Required
 - ForeignKey
 - InverseProperty

KEY

- Define a propriedade que será chave primária na tabela:

```
[Key]  
public intCodigo { get; set; }
```

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

Não precisa mapear com [Key]



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

Por padrão, chave primária é auto-increment

- 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.

Será gerado um valor pelo banco



```
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]  
public int codigo { get; set; }
```

Não será gerado valor pelo banco



```
[DatabaseGenerated(DatabaseGeneratedOption.None)]  
public int ClienteId { get; set; }
```

- Podemos definir o **nome** da coluna no banco de dados:

Nome da coluna no Banco de Dados



```
[Column("Nascimento")]  
public DateTime DataNasc { get; set; }
```

REQUIRED

- Define que a coluna é obrigatória.

Coluna obrigatória

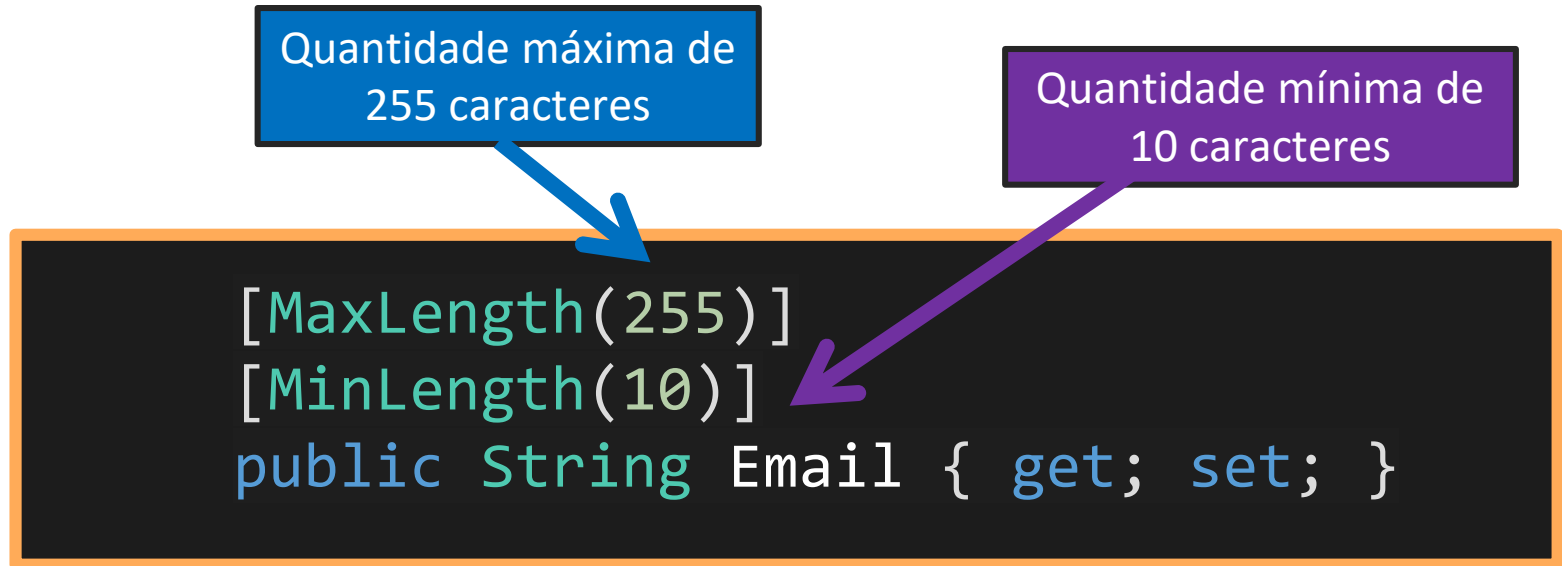


```
[Required]  
public string Nome { get; set; }
```

No ASP.NET MVC algumas annotations para mapeamento de banco de dados são utilizados para validação de tela.

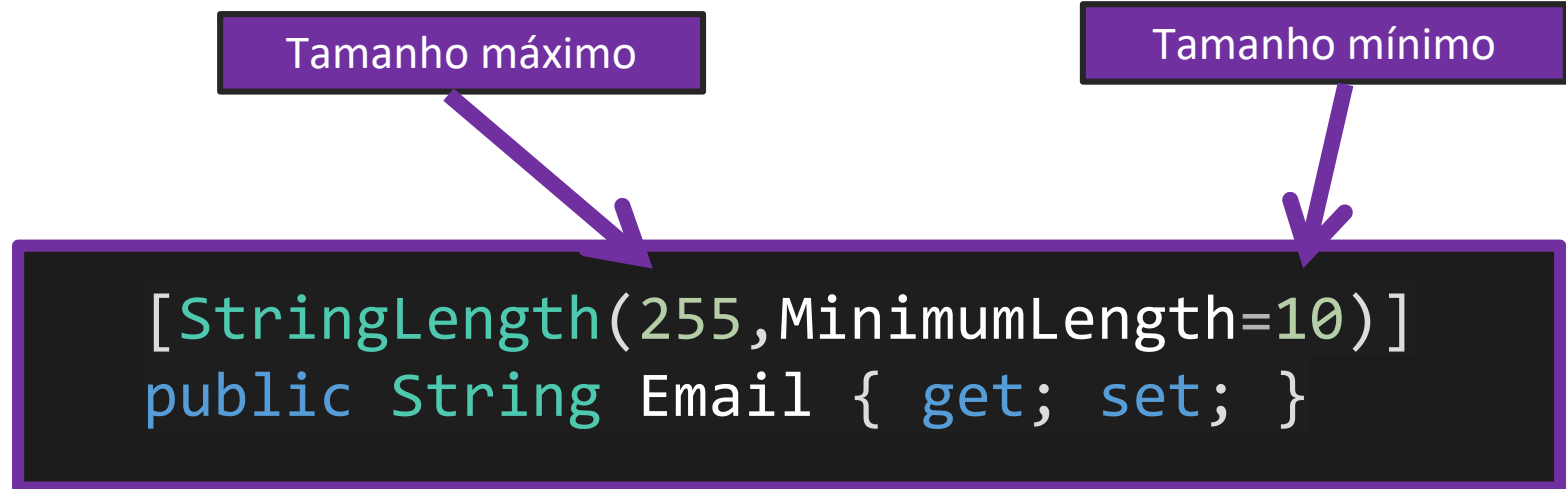
MAXLENGTH E MINLENGTH

- Define o tamanho máximo e mínimo, respectivamente, permitido para um array ou string.




O Exemplo mostra as duas annotations sendo aplicadas juntas, mas podemos utiliza-las de forma separadas.

- Define os tamanhos máximo e mínimo para um campo string.



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

A propriedade idade não será mapeada para o banco de dados



```
[NotMapped]  
public int Idade { get; set; }
```

LINQ

- **LINQ – Language Integrated Query** é um componente do .NET que permite efetuar consultas de propósito geral, com uma sintaxe parecida com SQL.
- Vamos trabalhar com **Extension Method e Expressões Lambdas**.

```
var busca =  
context.Clientes.Where(c => c.Name == nome).ToList();  
  
var usuario =  
context.Clientes.Where(c => c.Name == nome &&  
c.Email == email).FirstOrDefault();
```

- **FirstOrDefault():** retorna o primeiro elemento ou null caso não encontre.
- **Any():** retorna se encontrou ou não o elemento;
- **OrderBy():** ordena a resposta;
- **Count():** retorna o número de elementos;
- **Take():** configura a quantidade de elementos para o retorno;

<http://linq101.nilzorblog.com/linq101-lambda.php>

<http://msdn.microsoft.com/en-us/magazine/cc337893.aspx>

Copyright © 2013 - 2017 - 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”