PROGRAMAÇÃO BACK END II

Maurício de Oliveira Saraiva

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Construção de uma aplicação local em C# com SGBD

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Construir uma aplicação em C# para acessar o banco de dados.
- Manipular o banco de dados por meio de uma aplicação C#.
- Fornecer aplicação e banco de dados no GitHuB.

Introdução

A linguagem C# é amplamente utilizada no meio acadêmico e comercial. Com esta linguagem é possível implementar aplicativos que atendem às mais diversas funcionalidades, como sistemas educativos e comerciais, ferramentas de *software* de uso geral e jogos.

O MySQL é um dos sistemas gerenciadores de bancos de dados mais utilizados no mercado. Sua facilidade de uso colabora para isso. Com as ferramentas do MySQL, pode-se modelar e criar bancos de dados para as mais diversas necessidades, desde sistemas pequenos até grandes bases de dados.

Neste capítulo, você irá aprender a construir de uma aplicação local em C# que manipula um banco de dados MySQL. Além disso, você poderá baixar o projeto do GitHub.

Construindo uma aplicação em C#

C# é uma linguagem de programação da plataforma .NET *Framework*, desenvolvida pela Microsoft. Sua sintaxe lembra as linguagens C, C++, Java e Javascript, e os programas escritos em C# combinam um ou mais arquivos de origem para produzir um *Assembly*, que pode ser um arquivo executável (.exe) ou uma biblioteca dinâmica (.dll) (ESTRUTURA..., 2016).

Entre suas principais características, destacam-se as seguintes (UM TOUR..., 2019).

- **Orientada a componentes:** os programas são compostos por módulos que reutilizam componentes disponibilizados pelo fabricante ou por outros desenvolvedores.
- **Orientada a objetos:** implementa diversos recursos do paradigma da orientação a objetos, como classe, polimorfismo, herança, interface, modificadores de acesso e outros.
- **Fortemente tipada:** as variáveis possuem tipos de dados bem definidos e, sempre que necessário, as conversões devem ser realizadas, implícita ou explicitamente. Além disso, possui recursos para o uso de ponteiros, como C e C++.
- **Ambiente gerenciado:** o gerenciamento de memória dos programas escritos em C# é realizado pelo *runtime* via Garbage Collector (GC), garantindo um modo seguro de execução.

É possível implementar programas de todos os tipos com a linguagem C#, inclusive jogos para dispositivos móveis, *desktop* e Web, não se limitando apenas ao sistema operacional Windows, uma vez que a plataforma *.NET Core* também é compatível com macOS e Linux.

Existem vários ambientes de desenvolvimento integrado (IDE) para C#, sendo o Visual Studio (VS) o mais popular. O VS é um conjunto de ferramentas que auxilia na escrita do código-fonte, sugerindo complementos e melhores práticas enquanto o programador digita as instruções no código. Além disso, realiza a compilação, o *debug* e a execução, entre outras funcionalidades.

Neste capítulo, vamos implementar um aplicativo local em C# para *desktop* com o auxílio do Visual Studio 2019, que foi disponibilizado recentemente à comunidade de desenvolvedores de *software*.

Link

No *link* a seguir, você pode fazer o *download* do *Visual Studio 2019 – Community* (versão gratuita), conjunto completo de ferramentas para desenvolvimento C# e várias outras linguagens:

https://qrgo.page.link/iMi9o

Assim que o *Visual Studio 2019* estiver instalado, acesse a opção do menu *Arquivo – Novo – Projeto*. Entre as diversas opções que aparecerão, selecione *Aplicativo do Windows Forms (.NET Framework)* em C#, conforme ilustra a Figura 1.

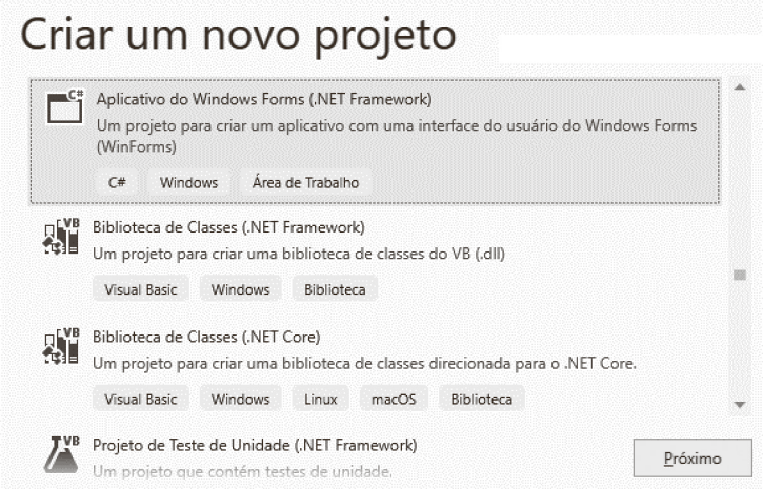

Figura 1. Criação de projeto em C# no Visual Studio 2019.

Na tela seguinte, entre com o Nome do projeto, defina o Local em que o projeto será armazenado, a versão do *.NET Framework* e clique em *Criar*. Um formulário vazio pode ser visualizado, indicando que o projeto está criado.

A aplicação que será implementada representa o Cadastro de empregado, contendo os campos Matrícula, CPF, Nome e Endereço, bem como os botões que irão manipular esse cadastro, como Pesquisar, Salvar, Excluir e Limpar, conforme ilustrado na Figura 2.

Estes elementos serão inseridos no formulário com o auxílio da Caixa de Ferramentas, localizada verticalmente à esquerda do formulário. Clique nos elementos *Label*, *TextBox*, *Button* e *DataGridView* e arreste-os para o *layout* do formulário.

Ajuste o nome do formulário para *FormLocal*, assim como a descrição e o nome de cada um desses elementos, localizando os itens *Text* e *Name* nas suas propriedades. Uma das formas de localizar as propriedades de um elemento, caso elas não estejam visíveis no campo inferior direito da IDE, é efetuar um clique com o botão direito do mouse sobre o elemento desejado e selecionar a opção Propriedades.

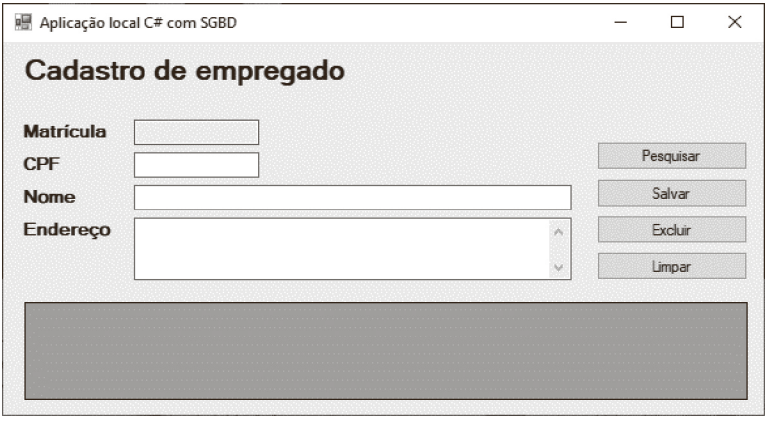

Figura 2. Protótipo da aplicação local.

Os elementos do formulário precisam ter suas propriedades iguais às descritas no Quadro 1, pois o programa irá utilizar esses dados para executar os eventos e manipular o banco de dados.

Quadro 1. Elementos do formulário Cadastro de empregado

Propriedades dos elementos do formulário			
Tipo de elemento	**Text**	**Name**	**ReadOnly**
Label	Cadastro de empregado	`label1`	
Label	Matrícula	`label2`	
Label	CPF	`label3`	
Label	Nome	`label4`	
Label	Endereço	`label5`	
TextBox		`textMatricula`	`True`
TextBox		`textCpf`	`False`
TextBox		`textNome`	`False`
TextBox		`textEndereco`	`False`
Button	Pesquisar	`buttonPesquisar`	
Button	Salvar	`buttonSalvar`	
Button	Excluir	`buttonExcluir`	
Button	Limpar	`buttonLimpar`	
DataGridView		`DataGrid-ViewEmpregado`	

Manipulando o banco de dados

Agora que o protótipo de nossa aplicação já está concluído, vamos associar os eventos de clique aos elementos da tela (botões e *DataGridView*) por meio de instruções escritas na linguagem C# no código-fonte. Antes disso, precisamos incluir no projeto uma biblioteca para acesso ao banco de dados MySQL.

Acesse o menu Projeto — Adicionar referência. Pesquise pelo nome MySql, selecione o item MySql.Data e clique no botão OK, conforme ilustra a Figura 3.

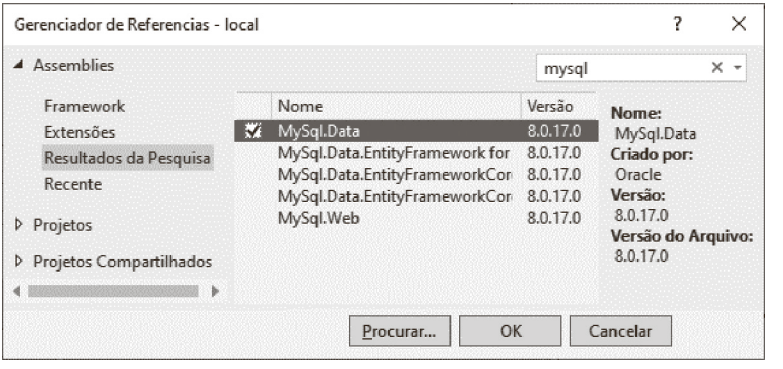

Figura 3. Principais comandos SQL agrupados por tipo.

A estrutura do banco de dados MySQL, criada para esta aplicação, está descrita na Figura 4. Essa estrutura contém o campo de matrícula como chave primária *identity* (AI), que incrementa seus valores automaticamente a cada registro incluído. O campo `cpf` possui uma restrição única (UQ), que impede valores duplicados. Os campos `nome` e `endereco` não possuem outras restrições, exceto a obrigatoriedade de conter valores não nulos (NN).

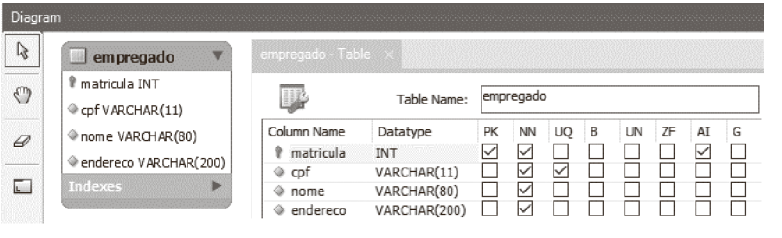

Figura 4. Estrutura da tabela *empregado*.

Os próximos passos são criar uma classe chamada `Empregado.cs`, que conterá os atributos da tabela `empregado` e os métodos de acesso ao banco de dados para realizar as operações que manipulam a tabela `empregado`. Além disso, deve-se associar os botões do formulário aos métodos da classe `Empregado`.

Foge ao foco deste capítulo disponibilizar por completo o código-fonte do programa, uma vez que o projeto inteiro contém muitas linhas e será inteiramente disponibilizado para *download* no GitHub. Dessa forma, apresentaremos alguns trechos que consideramos mais relevantes para o entendimento da aplicação.

Classe `Empregado.cs`

Para criar a classe `Empregado` no projeto, acesse o menu Projeto — Adicionar Classe. Marque o item Classe (itens do Visual C#), altere seu nome para `Empregado` por meio do campo localizado no canto inferior esquerdo e clique no botão Adicionar.

A classe `Empregado`, que contém 126 linhas, apresenta a *string* de conexão com o banco de dados, os atributos e seus respectivos métodos de acesso, além dos métodos `Pesquisar`, `Incluir`, `Alterar` e `Excluir`, que manipulam a tabela `Empregado` do banco de dados.

```
13 | private const string stringConexao=
   | "SERVER=localhost;PORT=3308;DATABASE=db_empregado;UID=root;PWD=admin";
14 | public int Matricula { get; set; }
15 | public string Cpf { get; set; }
16 | public string Nome { get; set; }
17 | public string Endereco { get; set; }
```

A linha 13 é a declaração da *string* de conexão com o banco de dados MySQL, na qual contém o nome do servidor (*server*), a porta (*port*) em que o MySQL responde às requisições, o nome do *schema* (*database*) do banco de dados, o usuário (UID) e a senha (PWD). A configuração da porta é opcional e deve ser informada somente se o MySQL estiver respondendo em uma porta diferente da porta padrão (3306).

As linhas 14 a 17 declaram os atributos `Matrícula`, `CPF`, `Nome` e `Endereço` da tabela `Empregado` e seus respectivos métodos de acesso `get` e `set` em um formato simplificado.

Link

Saiba mais sobre métodos de acesso em C# no *link* a seguir, que trata sobre os métodos `get` e `set`.

https://qrgo.page.link/d2r5t

```
19 public DataTable Pesquisar()
23     StringBuilder stringSql = new StringBuilder();
24     stringSql.Append("select Matricula, CPF, Nome, Endereco ");
25     stringSql.Append("from empregado ");
26     stringSql.Append("where 1");
28     if (Matricula != 0)
29       stringSql.Append(" and matricula=@matricula");
30     if (!Cpf.Equals(""))
31       stringSql.Append(" and cpf=@cpf");
32     if (!Nome.Equals(""))
33       stringSql.Append(" and nome like @nome");
35     MySqlConnection conexao=new MySqlConnection(stringConexao);
36     MySqlCommand comando=new MySqlCommand(stringSql.ToString(), conexao);
38     comando.Parameters.AddWithValue("@matricula", Matricula);
39     comando.Parameters.AddWithValue("@cpf", Cpf);
40     comando.Parameters.AddWithValue("@nome", "%"+Nome+"%");
42     MySqlDataAdapter adaptador = new MySqlDataAdapter();
43     adaptador.SelectCommand = comando;
45     DataTable dados = new DataTable();
46     adaptador.Fill(dados);
47     return dados;
53 }
```

Para reduzir espaço, as linhas em branco e o tratamento de exceções foram retirados no código-fonte escrito entre as linhas 19 e 53, com o objetivo de apresentar o que realmente nos interessa neste método `Pesquisar`, isto é, o modo como ele acessa o MySQL e retorna os registros da tabela `Empregado` para serem exibidos no *DataGridView* do formulário.

A linha 19 informa que o método retorna um objeto do tipo `DataTable`, que significa uma relação de registros de um banco de dados.

As linhas 23 a 33 e 38 a 40 definem uma *string* que representa a instrução SQL `select` que será executada, com a opção de realizar filtros por *matrícula*, *CPF* e *nome*. Utilizamos a classe `StringBuilder` para montar a *string* de maneira otimizada e o método `AddWithValue` para substituir os parâmetros da pesquisa de forma segura, impedindo ataques por *SQL Injection*.

Link

Saiba mais sobre ataques por *SQL Injection* no *link* a seguir:

https://qrgo.page.link/NJH1u

As linhas 35 e 36, respectivamente, abrem a conexão com o banco de dados e definem uma variável chamada comando, que será responsável pela preparação da *string* SQL que será executada.

A linha 42 declara um adaptador que, de fato, será o responsável pela execução do comando, por meio da instrução `SelectCommand` apresentada na linha 43.

Por fim, a linha 45 declara a variável de dados do tipo `DataTable` para receber o conjunto de registros, resultado da execução da instrução SQL, por meio do método `Fill` apresentado na linha 46. O retorno dos registros é efetuado pelo método `return` na linha 47.

Arquivo FormLocal.cs

O arquivo FormLocal.cs é o arquivo que contém o código-fonte do formulário do projeto (FormLocal.cs [Design]), que apresenta os campos e os botões inseridos anteriormente. Efetue um duplo clique do mouse sobre o botão Pesquisar e o código-fonte abrirá imediatamente em um evento de clique deste botão (método `ButtonPesquisar_Click`).

Este arquivo contém 124 linhas e os seguintes métodos: `Limpar`, que inicializa os valores dos campos do formulário; `Pesquisar`, que preenche o *DataGridView* com os registros do banco de dados; `Salvar`, que inclui ou atualiza um registro; e `Excluir`, que exclui um registro da tabela do banco de dados.

Um evento de clique vai ao *DataGridView* para que ele carregue nos campos do formulário os dados do registro selecionado. Isto é fundamental para que um registro possa ser editado, pois assim o botão Salvar identifica que se trata de uma atualização e executa o comando `update` no banco de dados, ao invés do comando `insert`.

```
28  private void Pesquisar()
29  {
32    int matricula = 0;
33    bool n = Int32.TryParse(textMatricula.Text, out matricula);
34
35    Empregado empregado = new Empregado();
36    empregado.Matricula = matricula;
37    empregado.Cpf = textCpf.Text;
38    empregado.Nome = textNome.Text;
39
40    dataGridViewEmpregado.DataSource = empregado.Pesquisar();
46  }
```

O método `Pesquisar` deste formulário está localizado entre as linhas 28 e 46 do código-fonte e sua finalidade é trazer os registros do banco de dados por meio de um objeto chamado empregado, uma instância da classe `Empregado`, que invoca o método `Pesquisar` da classe `Empregado`, apresentado anteriormente. Note que toda a regra está no método `Pesquisar` da classe `Empregado` e, no formulário, apenas a chamada é realizada.

As linhas 32 e 33 definem se o conteúdo do campo `Matrícula` terá um valor igual a zero, caso seu conteúdo esteja vazio, ou se ele terá a matrícula de um empregado já cadastrado, por meio da instrução `Int32.TryParse`, que faz a conversão para um número inteiro. Lembre-se que esse campo não pode ser editado, já que possui uma restrição AI no banco de dados, isto é, ele se incrementa automaticamente e não precisa ser preenchido quando se deseja incluir registros.

A linha 35 cria um objeto da classe `Empregado`. As linhas 36 a 38 inserem nos atributos `Matricula`, `Cpf` e `Nome` os dados preenchidos no formulário. Esses dados serão utilizados no método `Pesquisar` da classe `Empregado` para realizar os filtros, caso sejam preenchidos no formulário.

A linha 40 atribui ao *DataSource* do elemento `dataGridViewEmpregado` do formulário o resultado da instrução SQL executada no banco de dados. Isto significa que os registros e seus respectivos campos serão automaticamente populados no `dataGridViewEmpregado`.

Os demais métodos deste arquivo se comportam de forma semelhante. Eles trazem um objeto da classe Empregado com os dados dos campos do formulário e invocam os respectivos métodos para incluir, alterar e excluir os registros.

Publicação no GitHub

O GitHub é um serviço *open source* que fornece hospedagem de códigos-fonte, assim como controla o versionamento desses códigos, recebendo e enviando atualizações por meio de requisições realizadas pelos clientes Git (GITHUB..., 2019).

O conjunto do projeto da aplicação, modelo de dados e *script* do banco de dados já estão disponíveis no GitHub.

Link

Acesse o *link* a seguir para realizar o *download* ou clonar o projeto da aplicação local em C# com SGBD MySQL implementado especialmente para este capítulo:

https://qrgo.page.link/exz44

A tela do repositório está ilustrada na Figura 5, na qual é possível visualizar o nome e o endereço do repositório *AppLocalCSharp*, os *commits* que já foram realizados e alguns dos arquivos que fazem parte do projeto do aplicativo.

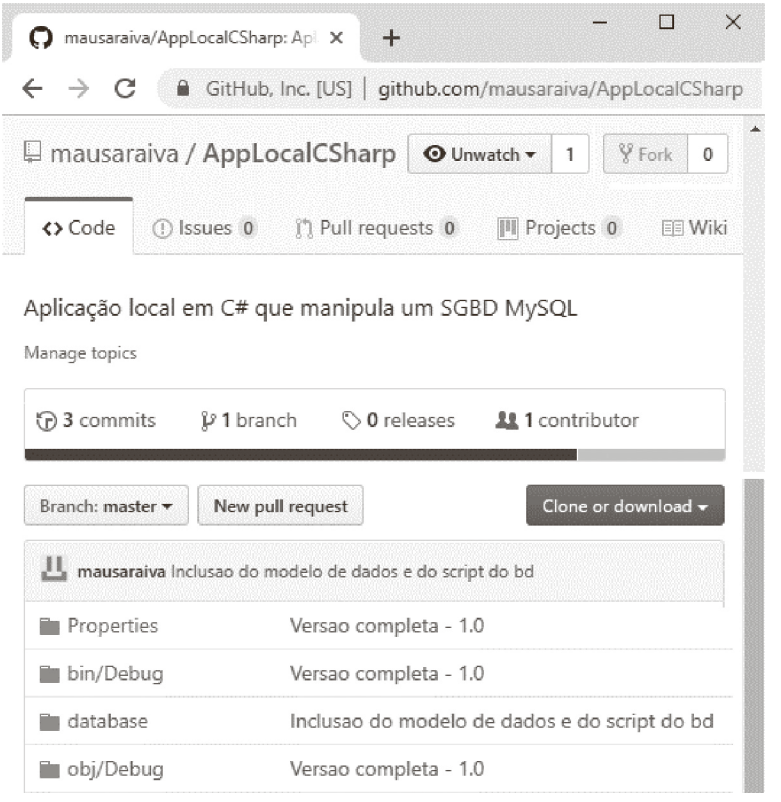

Figura 5. GitHub do *AppLocalCSharp*.

Referências

ESTRUTURA de um programa em C#. *Microsoft*, Redmond, 9 ago. 2016. Disponível em: https://docs.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/program-structure. Acesso em: 8 set. 2019.

GITHUB — The world's leading software development platform. *GitHub*, San Francisco, 2019. Disponível em: https://github.com/. Acesso em: 8 set. 2019.

UM TOUR pela linguagem C#. *Microsoft*, Redmond, 4 abr. 2019. Disponível em: https://docs.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/. Acesso em: 8 set. 2019.

Leitura recomendada

USANDO propriedades (Guia de Programação em C#). *Microsoft*, Redmond, 19 jul. 2015. Disponível em: https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/using-properties. Acesso em: 8 set. 2019.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS