Prof. Roberto de Castro

Escola Técnica Estadual "São Paulo" - ETESP
Linguagem C# - Visual Studio 2019

Banco de Dados (Continuação)

ETAPA V

ESCRITA DOS CÓDIGOS (Programação)

A programação seguirá um roteiro.

Começaremos a escrita do código pela camada "LibEntidade", pois será a camada responsável pelo "tráfego dos dados" (troca de informações) entre todas as outras camadas.

A camada "LibEntidade" é um projeto do tipo "Class Library". E o que isto significa?? Significa que não possui uma interface gráfica, mas será a responsável por apresentar ao sistema as "tabelas" (entidades) e seus respectivos campos.

Esta camada já possui uma classe, chamada "Class1", **devemos renomear a classe "Class1" para ClsTabClientes**. Certifique-se de que esteja realmente no local necessário. Expandir a "LibEntidade" e pressionar com o botão direito do mouse em "Renomear". Digitar o novo nome "ClsTabClientes" e confirmar a alteração.

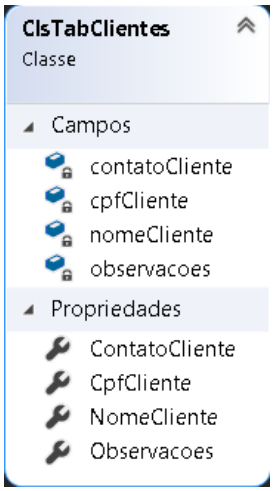
Vamos começar a implementação da "ClsTabClientes". Incluiremos os "campos" e "definiremos as propriedades" (gets / sets). No Gerenciador de Soluções, dê dois clique sobre a classe "ClsTabClientes" para abri-la:

Vamos incluir os "campos" e "definir as propriedades" (gets / sets). No Gerenciador de Soluções, dê dois clique sobre a classe "ClsTabClientes" para abri-la. Ao final deverá estar igual ao modelo abaixo:

```
7   namespace LibEntidade
8   {
        4 referências
9       public class ClsTabClientes
10      {
11          //Declaração dos campos/atributos
12          private long cpfCliente;
13          private string nomeCliente;
14          private string contatoCliente;
15          private string observacoes;
16
17          //Declaração das propriedades - gets/sets
                3 referências
18          public long CpfCliente { get => cpfCliente; set => cpfCliente = value; }
                3 referências
19          public string NomeCliente { get => nomeCliente; set => nomeCliente = value; }
                3 referências
20          public string ContatoCliente { get => contatoCliente; set => contatoCliente = value;
                2 referências
21          public string Observacoes { get => observacoes; set => observacoes = value; }
22      }
23  }
```

Podemos visualizar o diagrama de classes gerado pelo Visual Studio. Clique com o botão direito do mouse em "LibEntidade" → Exibir → Exibir em Diagrama de Classe (veja figura abaixo):

Biblioteca → LibEntidade e a classe "ClsTabClientes"



Continuando na programação das "Bibliotecas de Classe", vamos trabalhar agora com a "LibBLL" que será responsável pela lógica do processo.

A biblioteca "LibBLL" precisará da ClsBLLClientes (classe responsável pela implementação da lógica do cadastro de clientes: Novo / Salvar / Excluir....

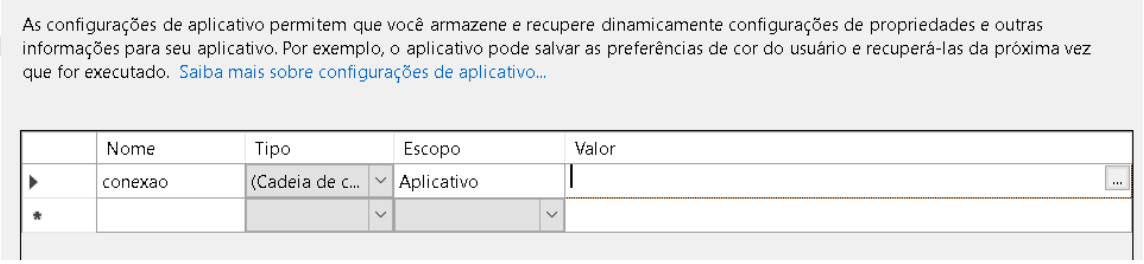
IMPORTANTE: Dentro da "LibBLL" já existe uma classe, a "Class1". Vamos alterar o nome para "ClsBllClientes". (botão direito → renomear).

Vamos trabalhar agora com a "LibDAL" que será responsável pela manipulação dos dados no Banco de Dados (esta será a única classe a possuir "privilégios" de incluir / alterar / excluir e consultar dados na tabela).

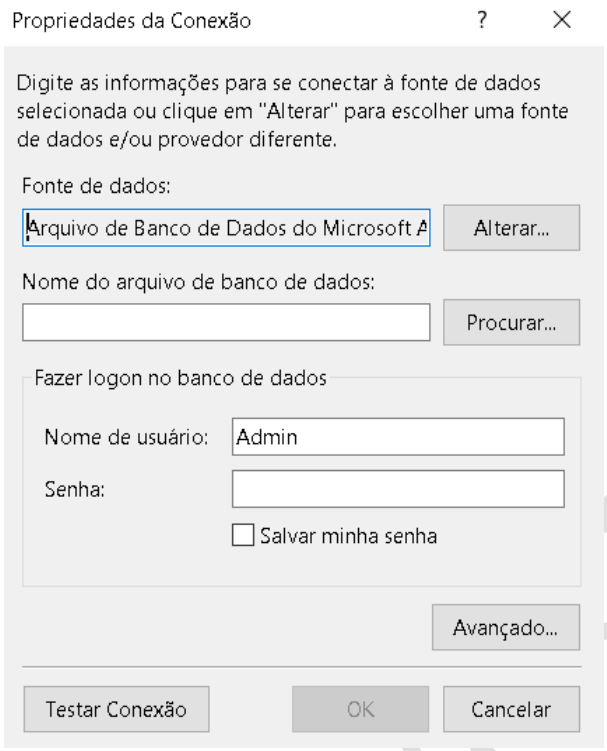
IMPORTANTE: Dentro da "LibDAL" já existe uma classe, a "Class1". Vamos alterar o nome para "ClsDaoClientes". (botão direito → renomear).

Aproveitando o momento que estamos com o "foco" na camada "DAL", vamos criar a **string de conexão**, informação que será necessária para a manipulação dos dados na Base de Dados. Botão direito sobre o projeto "LibDAL" → Propriedades → Configurações.

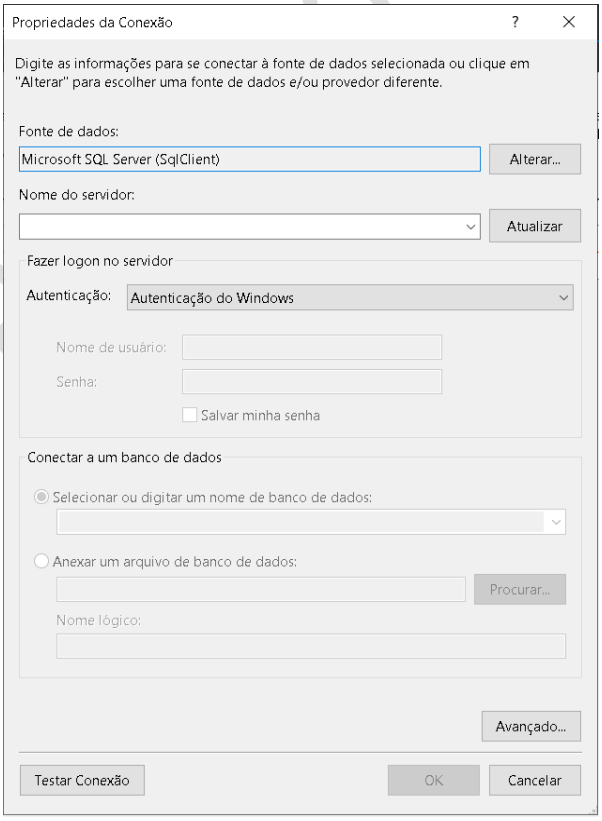
Será exibida mensagem: "Este projeto não contém um arquivo de configurações padrão. Clique aqui para criar um.". Clique e continue....



Na "coluna" nome digite "conexao", em tipo "selecione" → Cadeia de conexão (conections string). O escopo será alterado para "Aplicativo" e na última caixa "Valor" clique nos "...". Será exibida a caixa de "Propriedades da conexão".

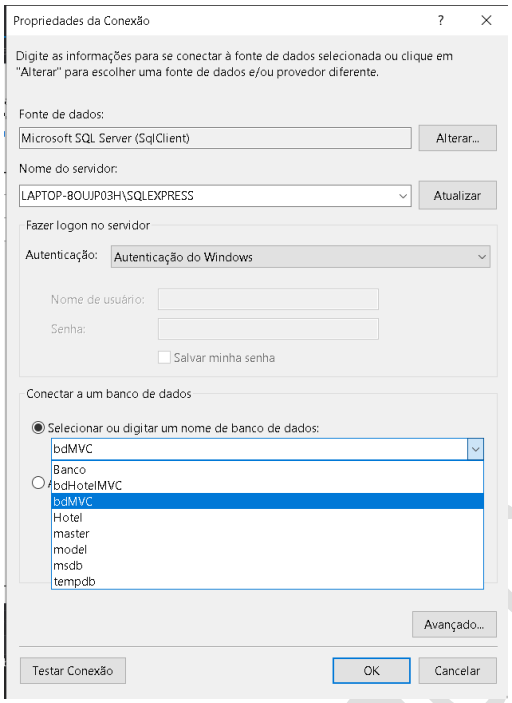


Clique em “Alterar” para informar nova “Fonte de Dados” e selecione “Microsoft Sql Server”.Clique em “OK”.



Na caixa “Nome do Servidor”, clique na seta, para que sejam exibidos os “servidores disponíeis”. E selecione o servidor que está instalado em sua máquina.

Proximo passo selecionar o “Banco de Dados” → **BDMVC**.

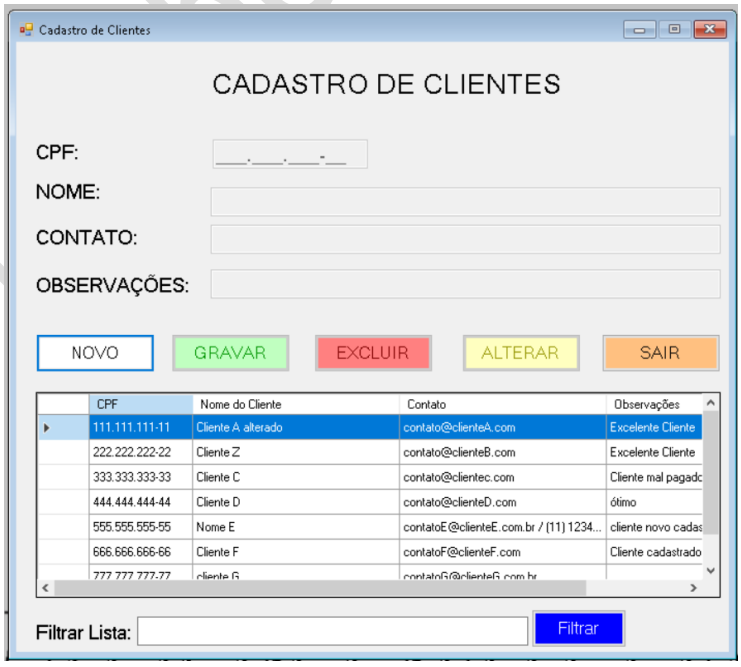


Clique em “Testar Conexão” para verificar a integridade do banco e para finalizar, clique em OK.

Vamos salvar: Arquivo → Salvar tudo.

Vamos retornar à camada “View” para darmos prosseguimento na programação do “cadastro de clientes” (FrmCadCli).

Quando o projeto estiver em execução, o formulário “Cadastro de Clientes” terá a aparência abaixo:



Inicialmente os “TextBox” estarão desabilitados (enabled = false).

Enabled “false” também para os botões “GRAVAR” “EXCLUIR” E “EDITAR”!

A proposta é de, ao carregar o formulário, exibir os clientes cadastrados no “DataGridView” e caso o usuário desejar fazer alguma manipulação em algum cliente (Excluir / Alterar), deverá clicar no “DataGridView” e os dados serão exibidos nos “TextBox”. Caso desejar incluir um novo cliente, o usuário deverá clicar em “NOVO”, que destravará os TextBox e habilitará o botão “GRAVAR”.

Vamos incluir no Form_Load deste formulário a chamada para um método e faremos, na sequência, a criação/programação deste método:

```
private void FrmCadCli_Load(object sender, EventArgs e)
{
 ListarGrid();
}

private void ListarGrid()
        {
            try
            {
                //cria um objeto do tipo "DataTable". Ver “Notas Explicativas” adiante!!
                DataTable dt;

                //Cria a instância da classe ClsBllClientes
                LibBLL.ClsBllClientes objBllClientes = new LibBLL.ClsBllClientes();

                //executa o método ListarCliente (que retorna um DataTale)
                //O parametro "buscaNome" (é uma variável do tipo “string” declarada na área
                //de escopo global do projeto) e terá conteúdo somente quando o método for
                //chamado pelo botão "Buscar". Caso contrário o parametro estará em branco
                //indicando que a pesquisa na tabela será geral (todos os clientes)
                //
               //IMPORTANTE: No momento da digitação da instrução abaixo, o método
                //não existe, porém, o Visual Studio criará, para posterior implementação!!
                //
                dt = objBllClientes.ListarCliente(buscaNome);

                //Associa o data table ao DataGrid
                dgvClientes.DataSource = dt;

                //Formata o DataGrid (obrigatóriamente deve ocorrer após o "preenchimento")
                dgvClientes.Columns[0].HeaderText = "CPF";
                dgvClientes.Columns[1].HeaderText = "Nome do Cliente";
                dgvClientes.Columns[2].HeaderText = "Contato";
                dgvClientes.Columns[3].HeaderText = "Observações";

                dgvClientes.Columns[0].DefaultCellStyle.Format = @"###\.###\.###-##";
                dgvClientes.Columns[1].Width = 200; //tamanho da coluna em pixel
                dgvClientes.Columns[2].Width = 200;
                dgvClientes.Columns[3].Width = 200;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Erro ao listar: " + ex.Message);
            }

        }
```

Vamos implementar o método “ListarCliente” que o assistente criou dentro da biblioteca LibBLL, classe “clsBllClientes”.

```
using LibEntidade;
using System;

using System.Data; //Necessário para se criar o DataTable


namespace LibBLL
{
    public class ClsBllClientes
    {
        public DataTable ListarCliente(string buscaNome)
        {
            //Ao digitar a instrução abaixo, o método "ListarCliente"
            //não existe na biblioteca "LibDal" --> Classe "ClsDaoClientes"
            //mas o assistente criará e implementaremos na sequência.
            //Executa o método e retorna um "DataTable".
            return new LibDAL.ClsDaoClientes().ListarCliente(buscaNome);

        }

    }

}
```

E agora vamos implementar o método "ListarCliente" que o assistente criou dentro da biblioteca LibDAL, classe "clsDaoClientes", utilizando, como ponto de partida, um "roteiro básico" para o acesso/manipulação dos dados:

- Definir a "string de conexão" (contém informações referentes ao caminho do BD, nome do banco...)

- Criar o objeto "Conection", reponsável por estabelecer a conexão com o BD. Se a base for SQL (SqlConnection) se "Access" (OledbConnection).

- Criar o objeto "Command", responsável por "armazenar/conter" as instruções SQL (select, insert, update, delete). Se a base for SQL (SqlCommand), se "Access" (OledbCommand).

- Definir/criar os "parâmetros" (variáveis) necessários para as instruções SQL.

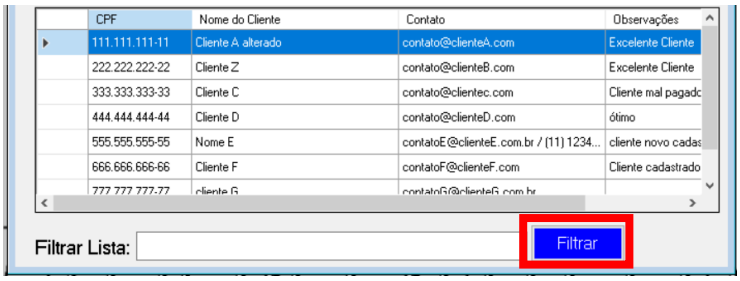
- Abrir a conexão.

- Executar a instrução SQL.

- Fechar a conexão.

```
using LibEntidade;
using System;
using System.Data; //Declaração necessária para a criação do objeto DataTable

using System.Data.SqlClient; //este  using é necessário para se utilizar/acessar BD SQL!!

namespace LibDAL
{
    public class ClsDaoClientes
    {
        //
        //O método abaixo é executado após a carga do formulário FrmClientes
        //para listar os clientes cadastrados.
        //
        public DataTable ListarCliente(string buscaNome)
        {
            //declara o objeto de conexão. OBS: Ver notas explicativas aditante!!
            SqlConnection conexao = new SqlConnection();

            //declara a variável de conexão.
            string strConexao = Properties.Settings.Default.conexao;

            conexao.ConnectionString = strConexao;

            //declaração do objeto command
            SqlCommand comando = new SqlCommand();

            comando.CommandType = CommandType.Text;

            //declara a instrução SQL
            //OBS: Se a variável buscaNome estiver em branco (vazia), a pesquisa será geral.
            comando.CommandText = "select * from TabClientes where NomeCliente like @Nome";

            comando.Parameters.Add("@Nome", sqlDbType: SqlDbType.VarChar, 100).Value = "%" +
                  buscaNome + "%";

            //Inicializa a conexão e abre o banco
            comando.Connection = conexao;

            conexao.Open();

            // Preenche um DataTable com dados da tabela Clientes
            SqlDataAdapter da = new SqlDataAdapter(comando);
            DataTable dt = new DataTable();
            da.Fill(dt);

            conexao.Close();

            return dt;
        }
```

Ainda dentro do "contexto" de "exibir dados no DataGrid", e antes de algumas considerações (Notas Explicativas) necessárias, veremos a implementação/programação do botão "Filtrar" (formulário FrmCadCli), que irá atualizar a lista de clientes exibidos no DataGrid.



O usuário poderá digitar na caixa de texto (ao lado do botão Filtrar) "parte do nome do cliente" que deseja visuzalizar no DataGrid, para que sejam exibidos somente os clientes que satisfaçam a condição do "nome". Se o usuário clicar no botão "Filtrar" sem digitar nenhuma informação no textbox, TODOS os clientes cadastrados no Banco de Dados serão exibidos!!

```
private void BtnBuscar_Click(object sender, EventArgs e)
        {
            //
            //OBS: Se nada for digitado no TextBox "TxtBuscar" a pesquisa será
            //feita de global (em toda a tabela de clientes!!
            buscaNome = TxtBuscar.Text;
            ListarGrid();
            TxtBuscar.Text = "";
            buscaNome = "";

        }
```

Perceba que a programação é a mesma utilizada no Form_Load, chamando o método "ListarGrid", com a diferença de transferir para o método o conteúdo de "buscaNome" (que é uma variável declarada na área "Global".

***** NOTAS EXPLICATIVAS *****

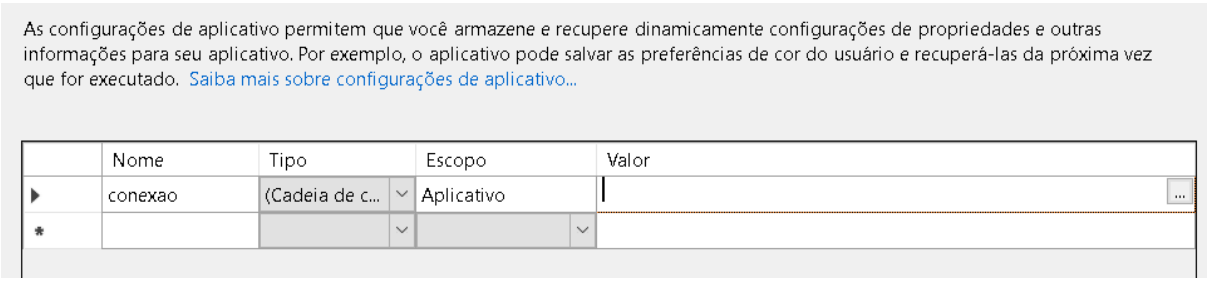
O que estamos "vendo" até o momento é a comunicação de informações e chamadas de métodos entre as camadas: do formulário (View) para a classe "BLL" e da classe "BLL" para a "DAL", respeitando desta forma o princípio de que cada "camada" seja responsável por implementar somente o que cabe a ela. Toda a operação de acesso aos dados está contida na biblioteca "DAL" (no caso específico de acesso à entidade de clientes) na classe ClsDaoClientes e por esta razão alguns "erros" poderão ocorrer nesta classe. Por exemplo:

I - SqlConnection conexao = new SqlConnection();

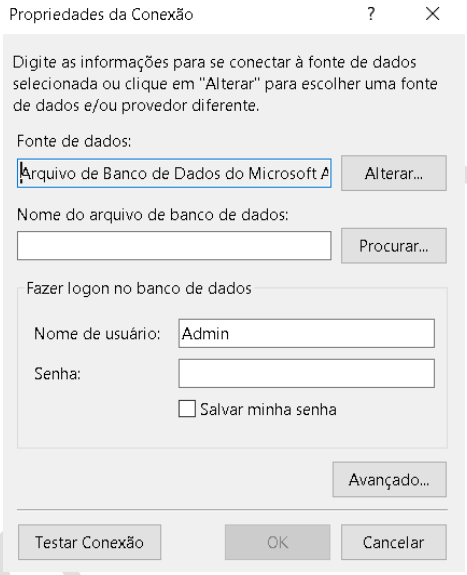
Possibilidades de erros:

1. Não está declarada a clausula "using System.Data.SqlClient" (classe que fornece os métodos necessários para acessar Banco de Dados SQL). Esta declaração DEVE ser utilizada para se acessar bando de dados SQL Server a partir da versão 7.0. Em versões anteriores do SQL Server DEVE-SE utilizar o provedor "SQLOLEDB".
2. O parâmetro "conexao" não é reconhecido!! Ou está "ausente" ou "está declarado com nome diferente". O parâmetro "conexao" contém os dados necessários sobre o Banco de Dados: tipo, local, nome... Este "parâmetro" DEVERIA ter sido criado previamente. Veja um "breve resumo" abaixo:

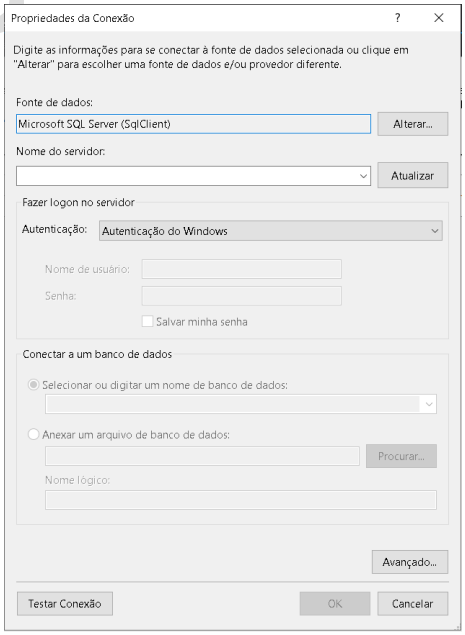
   Botão direito sobre a LibDAL → Propriedades → Configurações.



Confirme na coluna "Name" se "conexao" está declarada (sem acentuação), se necessário basta corrigi-la. Caso não esteja presente prossiga: digite "conexao" e, em tipo "selecione" → Cadeia de conexão (string conections). O escopo será convertido para "Aplicativo" e na última caixa 'Valor" clique nos "...". Será exibida a caixa de "Propriedades da conexão".

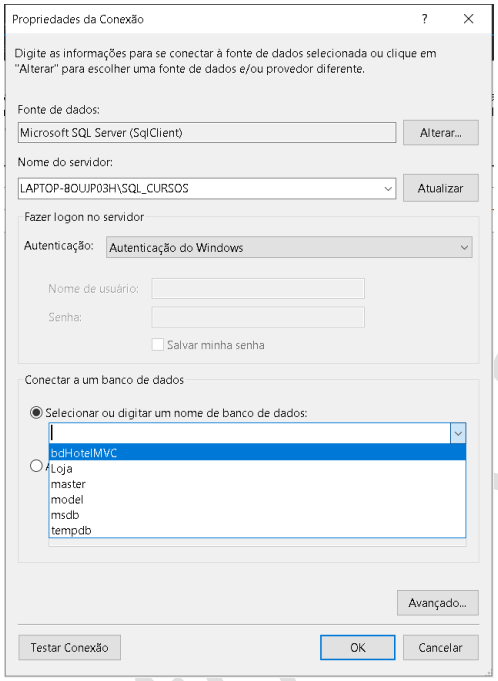


Clique em "Alterar" para informar nova "Fonte de Dados" e selecione "Microsoft Sql Server" e clique em "OK".



Na caixa “Nome do Servidor”, clique na seta para que sejam exibidos os “servidores disponíeis”. E selecione o servidor que está instalado em sua máquina.

Proximo passo selecionar o “Banco de Dados” → bdMVC.



Clique em “Testar Conexão” para verificar a integridade do banco e para finalizar, clique em OK.

Outra possibilidade de erro poderá ocorrer na instrução abaixo:

II - “comando.CommandText = "select * from TabClientes where NomeCliente like @Nome";

Possibilidades de erros:

1. Divergência na digitação do nome da tabela “TabClientes” (para isto, abra a tabela de clientes no SQL Server Management Studio (**SSMS**), confirme e ajuste, se necessário, o nome da tabela de clientes!
2. Divergência na digitação do nome do campo “NomeCliente” (confirme no SSMS).
3. O parâmetro “@Nome” está declarado errado na linha:

   comando.Parameters.Add("@Nome", sqlDbType: SqlDbType.VarChar, 100).Value = "%" + buscaNome + "%";

   A busca pelo “cliente” na tabela, será realizada por “parte” do nome (clausula like).

Agora, uma descrição de cada “termo” presente no método “ListarCliente” da classe “ClsDaoCliente”:

1. public DataTable ListarCliente(string buscaNome) → O método recebe um parâmetro do tipo “texto” (string buscaNome) e irá retornar um conjunto de dados do tipo DataTable (representa uma tabela de dados na memória).
2. “SqlConnection” →Necessário para se estabelecer uma conexão com um banco de dados SQL Server.
3. “SqlCommand”→ Utilizado para se representar uma instrução SQL ou Stored Procedure para incluir, alterar, consultar ou excluir dados em uma tabela.

4. "CommandType" → Define a forma como as instruções "SQL" serão fornecidas.
    a. "Text" →As instruções serão fornecidas diretamente pelo programa (CommandType.Text)
    b. "StoredProcedure" → As instruções estão declaradas no próprio servidor SQL (CommandType.StoredProcedure).

5. "Parameters.Add" → Antes de irmos diretamente ao "ponto", nossa declaração do "select" para selecionar clientes dependendo de uma condição (nome), poderia ser algo do tipo:

   comando.CommandText = "select * from TabClientes where NomeCliente like '%" + buscaNome + "%'";

   Porém, EVITE utilizar este tipo de sintaxe em seus programas. O exemplo acima abre a possibilidade de um ataque conhecido como injeção SQL, onde um código malicioso pode ser injetado no seu código.

   Para evitar esse tipo de problema devemos **usar parâmetros** pois qualquer informação colocada como parâmetro será tratada como um campo de dados, e não como parte da instrução SQL tornando assim sua aplicação mais segura.

   comando.CommandText = "select * from TabClientes where NomeCliente like @Nome";

   A sintaxe para se utilizar parâmetros segue o modelo abaixo:

   Command.Parameters.Add(@NomeParametro, SqlDbType, Tamanho).value = "conteúdo";

   Exemplo:

   comando.Parameters.Add("@Nome", sqlDbType: SqlDbType.VarChar, 100).Value = buscaNome;

   ou (na forma "abreviada):

   comando.Parameters.AddWithValue("@Nome", buscaNome);

   Para o caso acima, o "tipo do parametro" (SqlDbType) será inferido, baseado na informação que está sendo fornecida!! Este recurso poderá consumir mais tempo...

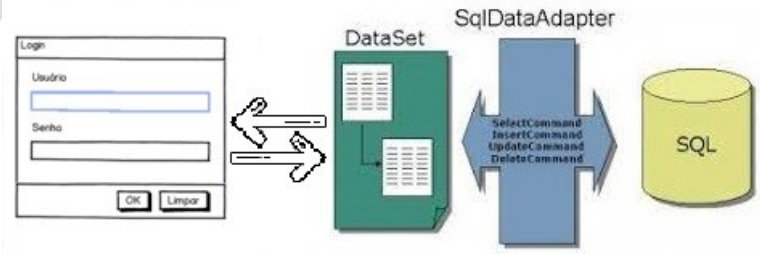
   DICA IMPORTANTE:

   - Utiliza-se o "@" (seguido do nome do parâmetro). *A ordem de declaração não importa, quando o banco for "SQL Server".*

   - Quando utilizamos o "Access", a ordem de inserção deve ser considerada!

   Para saber mais: https://www.macoratti.net/09/07/c_adn_7.htm

6. "SqlDataAdapter" → Observe a imagem abaixo:

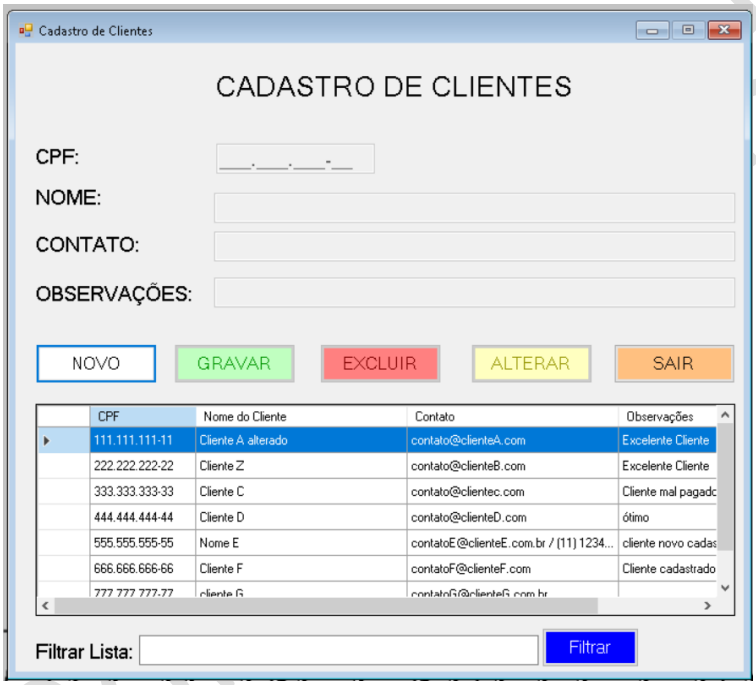


O "esquema" apresentado na figura acima ilustra as etapas para "obter" dados da Base de Dados física. O "adapter" (SqlDataAdapter ou OleDbDataAdpater) é utilizado como um "conector" (ponte) entre o banco de dados real e o banco de dados "virtual" (DataSet - conjunto de tabelas - desconectado) que será disponibilizado para a aplicação exibir/alterar/excluir dados.

O método "Fill" (preencher) do DataAdapter recupera os dados do BD (segundo o comando Select previamente fornecido) e preenche (popula) um DataTable com os dados (registros). O DataTable representa uma tabela de banco de dados em memória. Um DataSet (coleção de DataTables) representa todo o banco de dados em memória.

```
SqlDataAdapter da = new SqlDataAdapter(comando);
DataTable dt = new DataTable();
da.Fill(dt);
return dt;    // ← a tabela foi retornada
```

Finalmente, vamos retornar ao projeto... Se tudo estiver "OK", quando for executado, a tela de cadastro de clientes deverá exibir imagem abaixo, com o dataGrid exibindo os clientes cadastrados.



Vamos prosseguir com a programação do botão "Novo", que não envolverá nenhuma instrução específica de conexão com o Banco de Dados, mas apenas destravar as caixas de texto e habilitar o botão "Gravar":

```
private void BtnNovo_Click(object sender, EventArgs e)
        {
            DestravarText();

            //destrava botão gravar
            BtnGravar.Enabled = true;

            BtnNovo.Enabled = false;

            //trava DatagridView
            dgvClientes.Enabled = false;
            TxtCPF.Focus();
        }

        private void DestravarText()
        {
            TxtCPF.Enabled = true;
            TxtNome.Enabled = true;
            TxtContato.Enabled = true;
            TxtObs.Enabled = true;
        }
```

Agora, a programação do botão “Gravar”. Lembrando que estamos desenvolvendo um projeto em camadas (utilizando o padrão MVC). O botão gravar, em resumo deverá repassar as informações digitadas (CPF, Nome, Contato....) para a camada de Negócios, que será a responsável em analisar os dados recebidos e, se tudo estiver OK encaminhará os para a camada de manipulação dos dados.

```
private void BtnGravar_Click(object sender, EventArgs e)
 {
     try
     {
         //Cria a instância da classe de entidade: ClsTabClientes
         LibEntidade.ClsTabClientes ObjTabClientes = new
                LibEntidade.ClsTabClientes();

         //retira a máscara do CPF do cliente
         string sCPF = TxtCPF.Text.Replace(".", "").Replace("-", "");
         long numCPF = Convert.ToInt64(sCPF);


         //Transfere os conteúdos para a classe:
         ObjTabClientes.CpfCliente = numCPF;
         ObjTabClientes.NomeCliente = TxtNome.Text;
         ObjTabClientes.ContatoCliente = TxtContato.Text;
         ObjTabClientes.Observacoes = TxtObs.Text;

         //Cria a instância da classe "BLLClientes" e executa o método para incluir
           clientes e transfere para o método incluir a tabela de clientes!!
         LibBLL.ClsBllClientes ObjBllClientes = new LibBLL.ClsBllClientes();

         //IMPORTANTE: Ao digitar a linha abaixo, o método "INCLUIR" não
         //existe na classe "LibBLL --> ClsBllClientes" mas será criado pelo próprio
         //visual studio!!
         ObjBllClientes.Incluir(ObjTabClientes);

         MessageBox.Show("Cliente cadastro com sucesso!!", "Mensagem");

         Travar();
         LimparText();
         ListarGrid();

     }
     catch (Exception ex)
     {
         MessageBox.Show("Erro:" + ex.Message);
     }
 }


 private void Travar()
 {
     TxtCPF.Enabled = false;
     TxtNome.Enabled = false;
     TxtContato.Enabled = false;
     TxtObs.Enabled = false;

     //trava botões gravar / editar / excluir
     BtnGravar.Enabled = false;
     BtnEditar.Enabled = false;
     BtnExcluir.Enabled = false;

     BtnNovo.Enabled = true;
```

```
        //destrava DatagridView
        dgvClientes.Enabled = true;
    }

    private void LimparText()
    {
        TxtCPF.Text = "";
        TxtNome.Text = "";
        TxtContato.Text = "";
        TxtObs.Text = "";

    }
```

Programação do método "Incluir" na classe ClsBllClientes: (OBS: A assinatura do método já deverá estar criada, faltando apenas a "implementação" abaixo). Lembrando que o método "incluir" na classe de "regra de negócios" fará os testes necessários de validação dos campos!!!

```
    public class ClsBllClientes

    ..

    ..

    public void Incluir(ClsTabClientes objTabClientes)
    {
        //Faz as validações para verificar as regras de validação:
        //CPF, Nome e contato são obrigatório!!
        //
        //converte o campo NumCPF em Texto para verificar a qtde de caracteres
        string numCpf = Convert.ToString(objTabClientes.CpfCliente);
        if (numCpf.Trim().Length < 11)
        {
            throw new Exception("Numero CPF incorreto!!");
        }


        if (objTabClientes.NomeCliente.Trim().Length == 0)
        {
            throw new Exception("Nome do cliente é obrigatorio");
        }

        if (objTabClientes.ContatoCliente.Trim().Length == 0)
        {
            throw new Exception("Contato é obrigatório!!");
        }

        //Ao digitar a instrução abaixo, o método "Incluir"
        //não existe na biblioteca "LibDal" --> Classe "ClsDaoClientes"
        //mas o assistente criará e implementaremos na sequencia.
        new LibDAL.ClsDaoClientes().Incluir(objTabClientes);

    }
```

Se todas as validações estirem "OK", o método "Incluir" da classe "ClsBllClientes", transferirá os dados (objTabClientes) para o método "Incluir" na classe "ClsDaoClientes".

Seguiremos agora para a implementação do método "Incluir" da classe "ClsDaoClientes" (A assinatura do método deverá existir, faltando apenas a implementação abaixo):

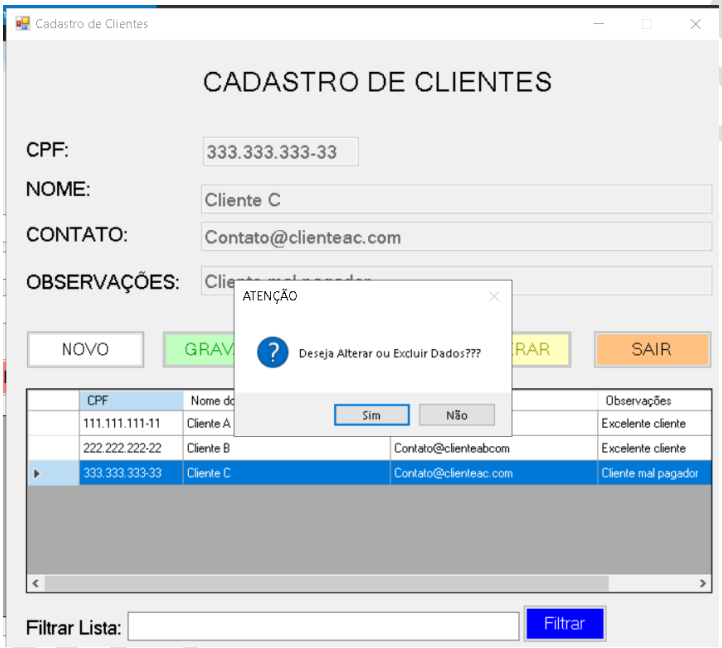
```
public void Incluir(ClsTabClientes objTabClientes)
{
    //declara o objeto de conexao
    SqlConnection conexao = new SqlConnection();

    try
    {
        //declara a variável de conexão.
        string strConexao = Properties.Settings.Default.conexao;

        conexao.ConnectionString = strConexao;

        //declaração do objeto command
        SqlCommand comando = new SqlCommand();

        comando.CommandType = CommandType.Text;

        //Parametros
        comando.Parameters.Add("@CPF", sqlDbType: SqlDbType.BigInt).Value =
                               objTabClientes.CpfCliente;
        comando.Parameters.Add("@Nome", sqlDbType: SqlDbType.VarChar, 100).Value =
                               objTabClientes.NomeCliente;
        comando.Parameters.Add("@Contato", sqlDbType: SqlDbType.VarChar, 100).Value
                               = objTabClientes.ContatoCliente;
        comando.Parameters.Add("@Obs", sqlDbType: SqlDbType.VarChar, 100).Value =
                               objTabClientes.Observacoes;

        //declara a instrução SQL
        string SQL = "insert into TabClientes (CPFCliente, NomeCliente,
               ContatoCliente, Observacoes) values (@CPF, @Nome, @Contato, @Obs)";

        comando.CommandText = SQL;

        //Inicializa a conexão e abre o banco
        comando.Connection = conexao;

        conexao.Open();

        //Executa a instrução SQL
        comando.ExecuteNonQuery();
    }

    catch (SqlException ex)
    {

        throw new Exception("Erro Servidor!!!" + ex.Number);
    }

    catch (Exception ex)
    {
        throw new Exception("Erro na gravação!!" + ex.Message);
    }

    finally
    {
        conexao.Close();
    }
}
```

Observe o "fluxo" de comunicação entre as camadas: do formulário (View) para a classe "BLL" e da classe "BLL" para a "DAL", respeitando desta forma o princípio de que cada "camada" seja responsável por implementar somente o que cabe a ela. Toda a operação de acesso aos dados está contida na biblioteca "DAL" (no caso específico de acesso à entidade de clientes) na classe ClsDaoClientes e por esta razão alguns "erros" poderão ocorrer nesta classe. Reveja as "anotações" previamente comentadas em "*** Notas Explicativas ***".

Se tudo ocorrer "perfeitamente", após a inclusão dos dados na base, o DataGrid do formulário de cadastro de clientes (FrmCadCli) será atualizado com as informações mais atualizadas!!

Avançaremos agora para os métodos de alteração e exclusão.

Os botões destas opções (Alterar / Excluir) estão travados e somente serão liberados após a realização de uma consulta e exibição dos dados no formulário, e isto será realizado clicando-se em uma linha do "DataGrid". Veja a imagem abaixo:



Por exemplo, ao "clicar" na terceira linha do DataGrid, o sistema exibe os dados nos "TextBox" e solicita uma confirmação se deseja continuar com a operação de Alterar ou Excluir dados". Se clicar em "NÃO", o sistema limpará os "TextBox". Se clicar em "SIM" destravará os "TextBox", permitindo a atualização dos dados e destravará os botões "Excluir" e "Alterar".

Veja a programação necessária, no evento "CellClick" do DataGrid:

```
private void dgvClientes_CellClick(object sender, DataGridViewCellEventArgs e)
 {
     //Exibe, nos TextBox, o conteúdo da linha selecionada no DataGrid
     //A propriede SelectonMode do Grid foi alterada para FullRowSelected.
     //
     TxtCPF.Text = dgvClientes.CurrentRow.Cells[0].Value.ToString();
     TxtNome.Text = dgvClientes.CurrentRow.Cells[1].Value.ToString();
     TxtContato.Text = dgvClientes.CurrentRow.Cells[2].Value.ToString();
     TxtObs.Text = dgvClientes.CurrentRow.Cells[3].Value.ToString();

     if (MessageBox.Show("Deseja Alterar ou Excluir Dados???", "ATENÇÃO",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No)
     {
         LimparText();
```

```
                TxtBuscar.Text = "";
                buscaNome = "";
                ListarGrid();
                return;
            }

            DestravarText();

            //OBS: O TextBox do CPF deverá ficar travado
            TxtCPF.Enabled = false;

            //Destrava - trava botões
            BtnNovo.Enabled = false;
            BtnGravar.Enabled = false;
            BtnExcluir.Enabled = true;
            BtnEditar.Enabled = true;
        }
```

Se o usuário NÃO desejar fazer nenhuma alteração ou exclusão, clicará em “NÃO” e o sistema limpará os “TextBox” e atualizará o DataGridView.

Se o usuário desejar fazer alguma alteração ou exclusão, clicará em “SIM” e o sistema habilitirá os botões Excluir e Editar (para que a operação de exclusão ou alteração possa ser realizada) e desabilitará os botões Novo e Gravar.

Observe que o TextBox do CPF ficará travado, impedindo que o usuário altere este campo, pois é o “campo chave” da tabela de clientes!

Neste momento o usuário poderá alterar informações ou excluir o registro do cliente.

Se o usuário desejar alterar alguma informação (Nome, Contato, Observações), bastará digitar a informação que deseja alterar e clicar em “Alterar”. O sistema exibirá a mensagem “Cliente alterado com sucesso”, atualizará o “DataGridView”, travará os TextBox, habilitará o botão novo, travará os botões Excluir e Editar.

Veremos, abaixo, a programação do procedimento EDITAR, do formulário de “Cadastro de Clientes”:

```
private void BtnEditar_Click(object sender, EventArgs e)
        {
            //
            // Esta opção estará disponível após uma consulta - clique no DataGrid
            //
            try
            {
                //Cria a instância da classe que contém a tabela Clientes
                LibEntidade.ClsTabClientes ObjTabClientes = new
                           LibEntidade.ClsTabClientes();
                //
                //retira a máscara do CPF do cliente
                //
                string sCPF = TxtCPF.Text.Replace(".", "").Replace("-", "");
                long numCPF = Convert.ToInt64(sCPF);


                //Transfere os conteúdos para a classe:
                ObjTabClientes.CpfCliente = numCPF;
                ObjTabClientes.NomeCliente = TxtNome.Text;
                ObjTabClientes.ContatoCliente = TxtContato.Text;
                ObjTabClientes.Observacoes = TxtObs.Text;
```

```
            //Cria a instancia da classe "BLLClientes" e executa o método para alterar
            //clientes transfere para o método incluir a tabela de clientes!!

            LibBLL.ClsBllClientes ObjBllClientes = new LibBLL.ClsBllClientes();

            //IMPORTANTE: Ao digitar a linha abaixo, o método "ALTERAR" não
            //existe na classe "LibBLL --> ClsBllClientes" nas será criado pelo próprio
            //visual studio!!
            ObjBllClientes.Alterar(ObjTabClientes);

            MessageBox.Show("Cliente alterado com sucesso!!", "Mensagem");

            Travar();
            LimparText();
            ListarGrid();

        }
        catch (Exception ex)
        {
            MessageBox.Show("Erro:" + ex.Message);
        }
    }
```

Abaixo, a programação do método “Alterar” na biblioteca “LibBLL”, classe ClsBllClientes”:

```
public void Alterar(ClsTabClientes objTabClientes)
    {
        //Faz as validações para verificar as regras de validação:
        //Nome e contato são obrigatório!!
        //O CPF não será alterado, mas será transferido pois é o campo chave

        if (objTabClientes.NomeCliente.Trim().Length == 0)
        {
            throw new Exception("Nome do cliente é obrigatorio");
        }

        if (objTabClientes.ContatoCliente.Trim().Length == 0)
        {
            throw new Exception("Contato é obrigatório!!");
        }

        //Ao digitar a instrução abaixo, o método "Alterar"
        //não existe na biblioteca "LibDal" --> Classe "ClsDaoClientes"
        //mas o assistente criará e implementaremos na sequencia.
        new LibDAL.ClsDaoClientes().Alterar(objTabClientes);
    }
```

Observe que o método “Alterar” na camada da “Regra de Negócios” (BLL), faz as validações necessárias nos campos e, se houver alguma inconformidade, devolverá uma mensagem de erro para a rotina que chamou o método e NÃO continuará com o procedimento de atualização.

Se tudo estiver em conformidade, fará uma chamada ao método “Alterar” da camada de acesso a dados (DAL), transferindo os dados da tabela (ObjTabClientes) para esta camada efetuar a alteração necessária.

Veja abaixo programação do método “Alterar” da biblioteca “LibDAL”, classe “ClsDaoClientes”:

```
public void Alterar(ClsTabClientes objTabClientes)
        {
            //declara o objeto de conexao
            SqlConnection conexao = new SqlConnection();

            try
            {
                //declara a variável de conexão.
                string strConexao = Properties.Settings.Default.conexao;

                conexao.ConnectionString = strConexao;

                //declaração do objeto command
                SqlCommand comando = new SqlCommand();

                comando.CommandType = CommandType.Text;

                //Parametros
                comando.Parameters.Add("@CPF", sqlDbType: SqlDbType.BigInt).Value =
                       objTabClientes.CpfCliente;

                comando.Parameters.Add("@Nome", sqlDbType: SqlDbType.VarChar, 100).Value =
                       objTabClientes.NomeCliente;

                comando.Parameters.Add("@Contato", sqlDbType: SqlDbType.VarChar, 100).Value
                       = objTabClientes.ContatoCliente;

                comando.Parameters.Add("@Obs", sqlDbType: SqlDbType.VarChar, 100).Value =
                      objTabClientes.Observacoes;


                //declara a instrução SQL
                string SQL = "update TabClientes set NomeCliente = @Nome, ContatoCliente =
                              @Contato, Observacoes = @Obs where CPFCliente = @CPF";

                comando.CommandText = SQL;

                //Inicializa a conexão e abre o banco
                comando.Connection = conexao;

                conexao.Open();

                //Executa a instrução SQL
                comando.ExecuteNonQuery();
            }

            catch (SqlException ex)
            {
                throw new Exception("Erro Servidor!!!" + ex.Number);
            }

            catch (Exception ex)
            {
                throw new Exception("Erro na Alteração dos dados!!" + ex.Message);
            }

            finally
            {
                conexao.Close();
            }

        }
```

Finalizamos, desta forma, a programação necessária para a "alteração" de dados. Seguiremos para a "exclusão" de clientes!!

De volta à camada "View", formulário "FrmCadCli", botão EXCLUIR:

```
private void BtnExcluir_Click(object sender, EventArgs e)
        {
            //
            // Esta opção estará disponível após uma consulta - clique no DataGrid
            //
            try
            {
                //
                //retira a máscara do CPF do cliente
                //
                string sCPF = TxtCPF.Text.Replace(".", "").Replace("-", "");
                long numCPF = Convert.ToInt64(sCPF);

                //Cria a instancia da classe "BLLClientes" e executa o método para excluir
                  clientes
                LibBLL.ClsBllClientes ObjBllClientes = new LibBLL.ClsBllClientes();

                //IMPORTANTE: Ao digitar a linha abaixo, o método "EXCLUIR" não
                //existe na classe "LibBLL --> ClsBllClientes" nas será criado pelo próprio
                //visual studio!!

                ObjBllClientes.Excluir(numCPF);

                MessageBox.Show("Cliente excluído com sucesso!!", "Mensagem");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Erro:" + ex.Message);
            }
            finally
            {
                Travar();
                LimparText();
                ListarGrid();
            }
        }
```

Programação do método Excluir, da biblioteca "LibBLL", classe "ClsBLLClientes":

```
public void Excluir(long numCPF)
        {
            new LibDAL.ClsDaoClientes().Excluir(numCPF);

        }
```

Programação do método Excluir da biblioteca "LibDAL", classe "ClsDaoClientes":

```
public void Excluir(long numCPF)
        {
            //declara o objeto de conexao
            SqlConnection conexao = new SqlConnection();

            try
            {
                //declara a variável de conexão.
                string strConexao = Properties.Settings.Default.conexao;

                conexao.ConnectionString = strConexao;

                //declaração do objeto command
                SqlCommand comando = new SqlCommand();

                comando.CommandType = CommandType.Text;

                //declara a instrução SQL
                comando.CommandText = "delete from TabClientes where CPFCliente = @CPF";

                //Parametros
                comando.Parameters.Add("@CPF", sqlDbType: SqlDbType.BigInt).Value = numCPF;

                //Inicializa a conexão e abre o banco
                comando.Connection = conexao;

                conexao.Open();

                //Executa a instrução SQL
                comando.ExecuteNonQuery();
            }

            catch (SqlException ex)
            {

                if (ex.Number == 547) // Erro de violação de integridade.
                {
                    throw new Exception("Cliente não poderá ser excluído pois existem dados
                                         relacionados em outra tabela!!");
                }
                else
                {
                    throw new Exception("Erro Servidor!!!" + ex.Number);
                }
            }

            catch (Exception ex)
            {
                throw new Exception("Erro na exclusão!!" + ex.Message);
            }

            finally
            {
                conexao.Close();
            }

        }
```

***** FIM *****