



# **PBP - Capacitação em Programação .NET (WFA – *Windows Forms Application*)**

## **Semana 4**

Tecnologia ADO.NET, conexão com o banco de dados MySQL, criação de banco de dados e tabela, instruções SQL (Select, Insert, Update, Delete).

**Prof. Fabrício Braoios Azevedo**

**Prof. Tiago Jesus de Souza**

# ADO.NET

O ADO.NET é uma tecnologia de acesso a banco de dados que oferece diversas classes que fornecem inúmeros serviços de operações relacionadas a banco de dados, permitindo o acesso a diferentes plataformas, tais como: SQL Server, MySQL, Oracle, Sybase, Access, XML, arquivos textos, etc. Essas conexões podem ser realizadas de três formas diferentes: OLE DB , SQL e ODBC. Os provedores de dados que acompanham o ADO.NET, permitem a utilização de várias classes que interagem diretamente com a base de dados, as quais são identificadas por um prefixo, conforme tabela abaixo:

Provedor	Descrição
<b>ODBC Data Provider</b> <b>API Prefixo: Odbc</b>	Geralmente usada para banco de dados mais antigos, que utilizam a interface ODBC.
<b>OleDb Data Provider</b> <b>API Prefixo: OleDb</b>	Conexão do tipo OleDb, como por exemplo o Access ou Excel;
<b>Oracle Data Provider</b> <b>API Prefixo:Oracle</b>	Para implementação de Banco de Dados Oracle.
<b>SQL Data Provider</b> <b>API Prefixo:Sql</b>	Para implementação de Banco de Dados Microsoft SQL Server.



# ADO.NET – MySQL

O provedor do MySQL não faz parte diretamente da tecnologia ADO.NET, portanto, este provedor será incluído no projeto manualmente. O nome do arquivo (provedor) que será incluído é o **MySQL.Data.Dll**, que se encontra na pasta do MySQL Server instalado no computador. Por exemplo:

C:\Program Files (x86)\MySQL\Connector NET 6.7.4\Assemblies\v2.0\MySQL.Data.Dll

C:\Program Files (x86)\MySQL\Connector NET 6.7.4\Assemblies\v4.0\MySQL.Data.Dll

C:\Program Files (x86)\MySQL\Connector NET 6.7.4\Assemblies\v4.5\MySQL.Data.Dll

Antes de selecionar a biblioteca desejada, verificar a versão do Framework na qual está sendo desenvolvida a aplicação. As versões dos Framework's são:

- ✓ Visual Studio 2005 → Framework Versão 2.0
- ✓ Visual Studio 2008 → Framework Versão 3.5
- ✓ Visual Studio 2010 → Framework Versão 4.0
- ✓ Visual Studio 2012 → Framework Versão 4.5
- ✓ Visual Studio 2013 → Framework Versão 4.5

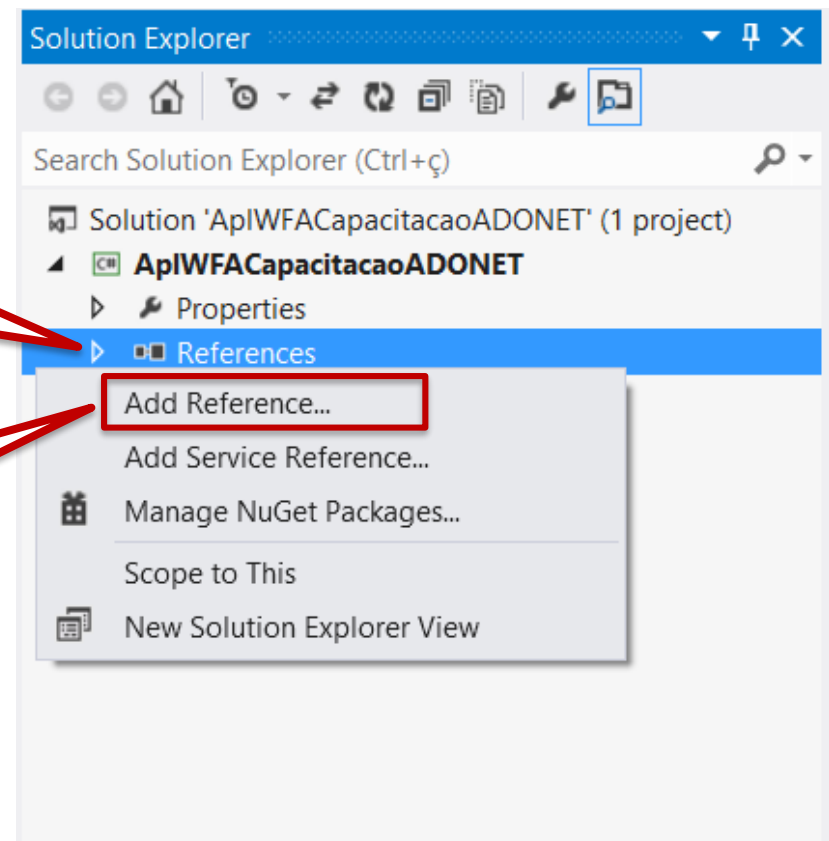
# ADO.NET – MySQL

Para adicionar um provedor externo, ou seja, adicionar uma referência externa, devemos seguir os seguintes passos:

1. Na **Solution Explorer**, ao lado direito da tela, devemos clicar com o botão direito do mouse em cima da opção **References**, e aparecerá a seguinte tela:

A opção **References** permite adicionar qualquer biblioteca externa e/ou nativas do VisualStudio.

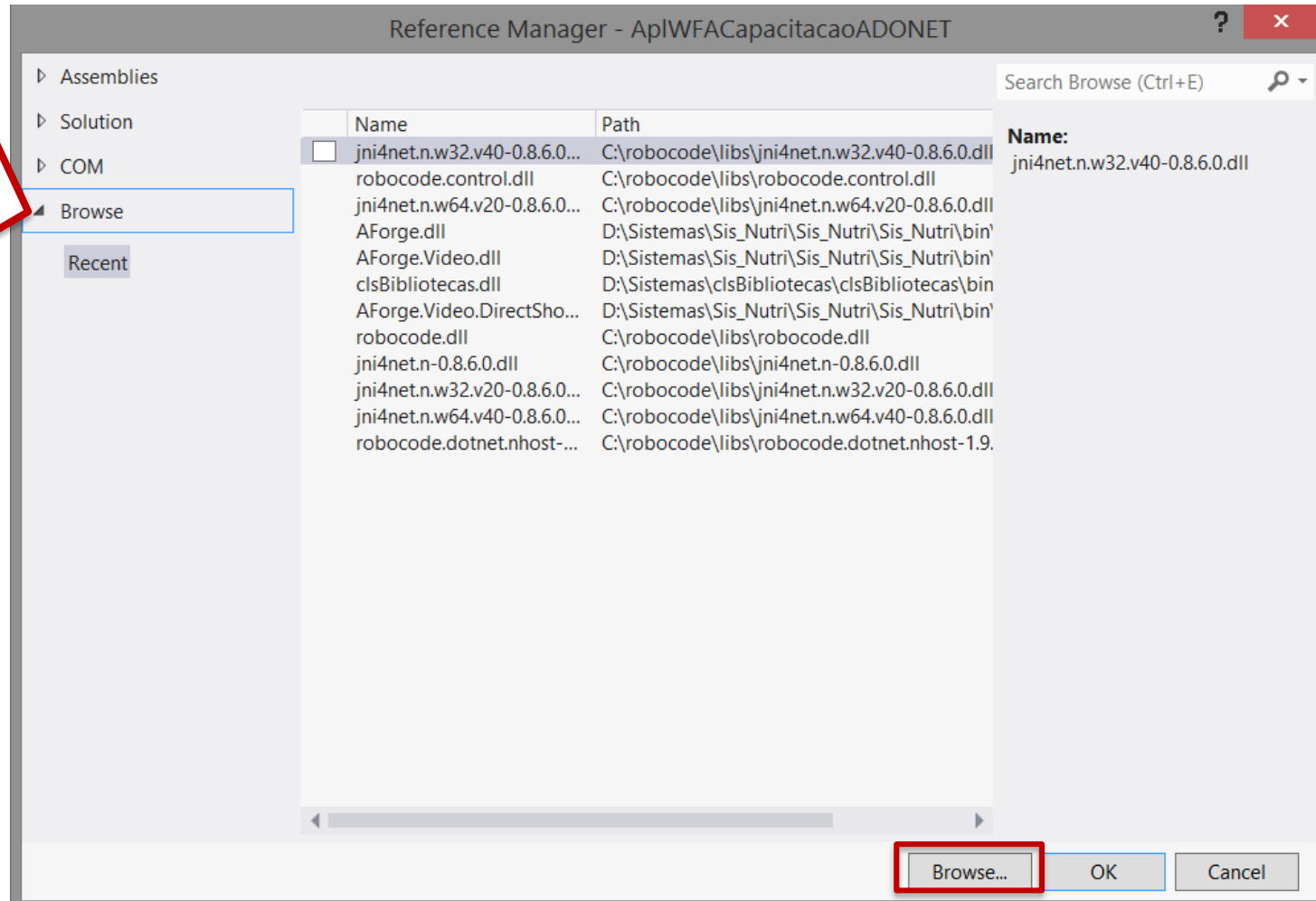
Clicar na opção **Add Reference...**, para selecionar o arquivo desejado.



# ADO.NET – MySQL

2. Após clicar no botão **Add References...**, aparecerá a seguinte tela:

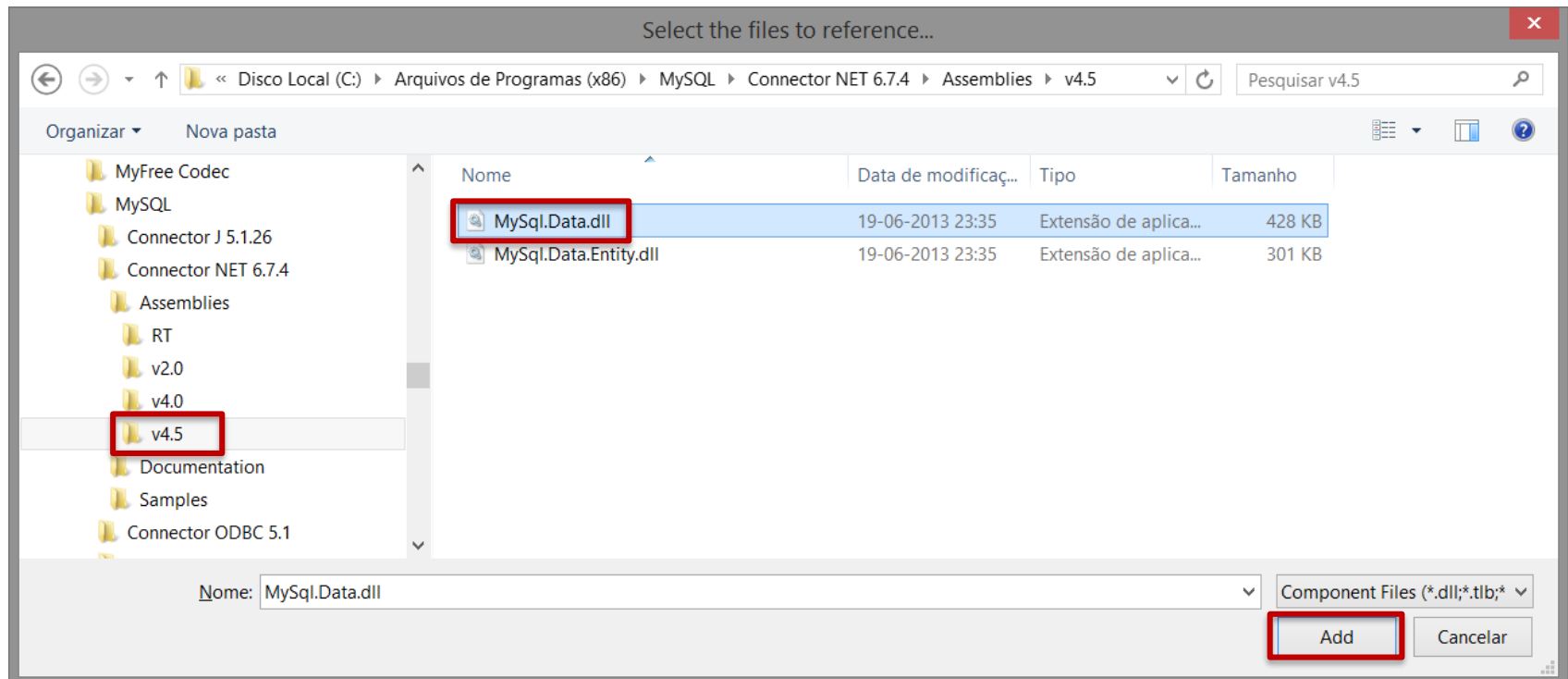
Na opção **Browse**, mostrará todas as bibliotecas que já foram incluídas anteriormente no seu Visual Studio.



# ADO.NET – MySQL

3. Após clicar no botão **Browse**, aparecerá a seguinte tela:

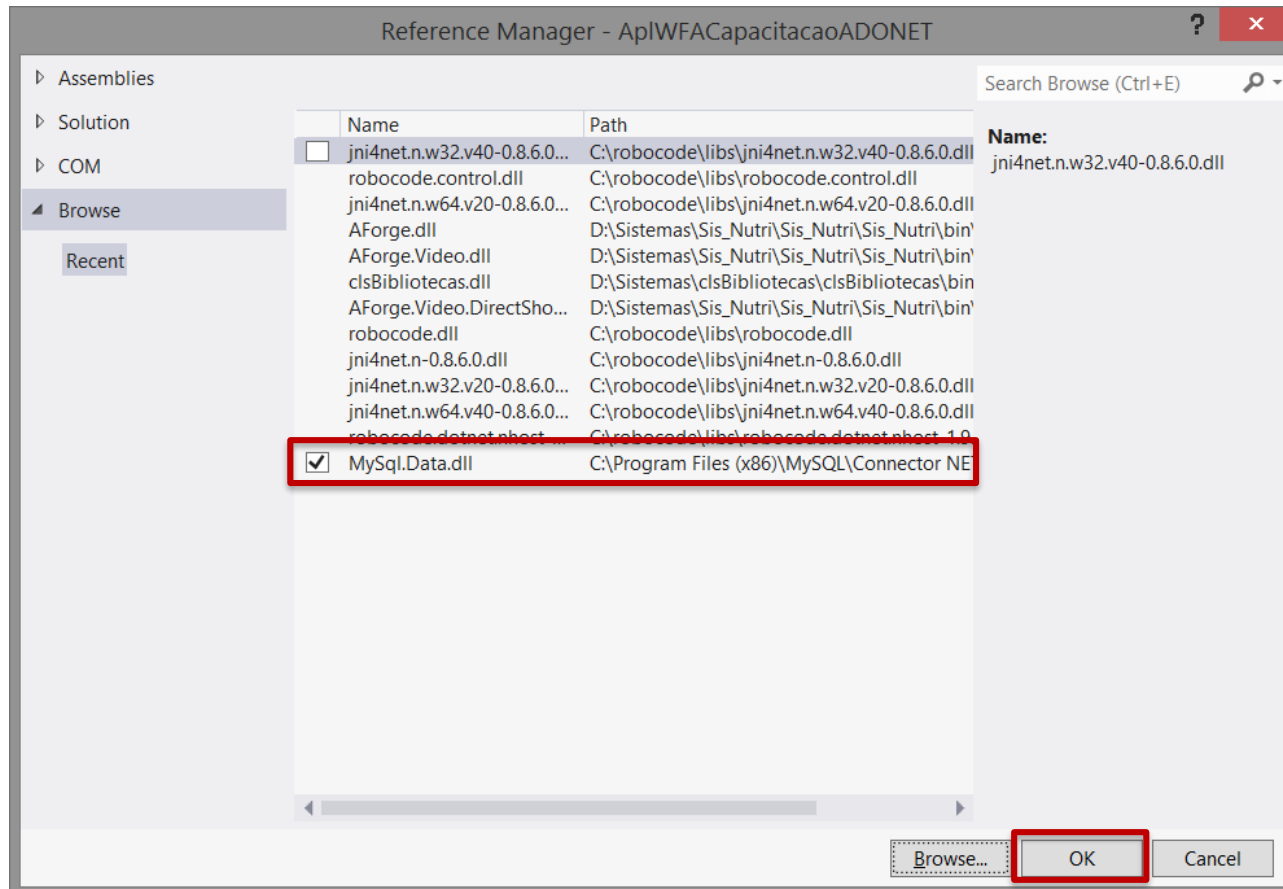
- Selecione o arquivo **MySQL.Data.Dll**, a partir da pasta que estão instalados os conectores do MySQL.
- Selecionar o conector para o ambiente **.NET**, como mostra a figura.
- Selecionar a versão do conector de acordo com a versão do Visual Studio (**Ver slide 5**). Em seguida clicar no botão **Add**.



# ADO.NET – MySQL

3. Após a seleção do arquivo (.dll), irá aparecer no gerenciamento de referências, como mostra a tela abaixo:

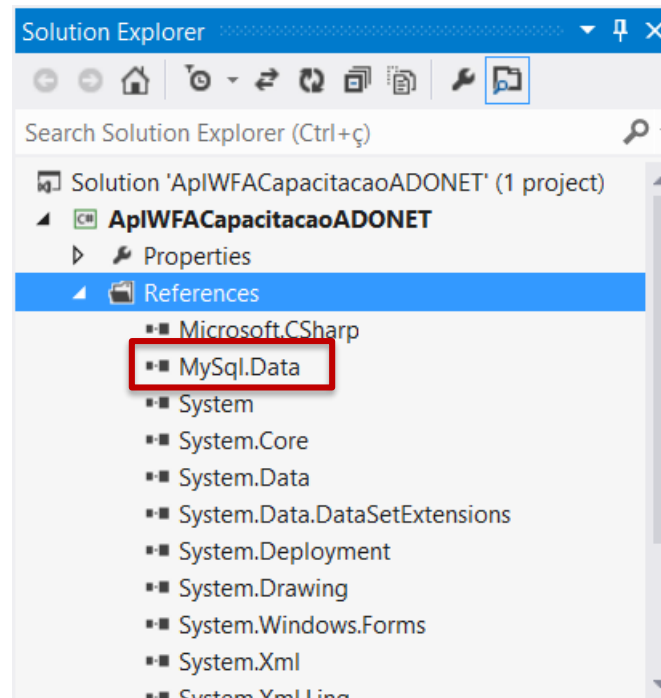
a. Em seguida clicar no botão **OK**.





# ADO.NET – MySQL

3. Após a seleção do arquivo (.dll), o mesmo irá aparecer na relação de referências, como mostra a tela abaixo:



A partir de agora, a biblioteca está pronta para ser usada no desenvolvimento.

**Sempre que abrir o mesmo projeto, não precisará mais adicionar a referência MySQL.Data.Dll sempre que abrir o projeto.**

# ADO.NET – MySQL

Para importarmos a biblioteca para o ambiente de desenvolvimento, devemos utilizar a instrução **using**, como mostra a figura abaixo:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MySql.Data.MySqlClient;

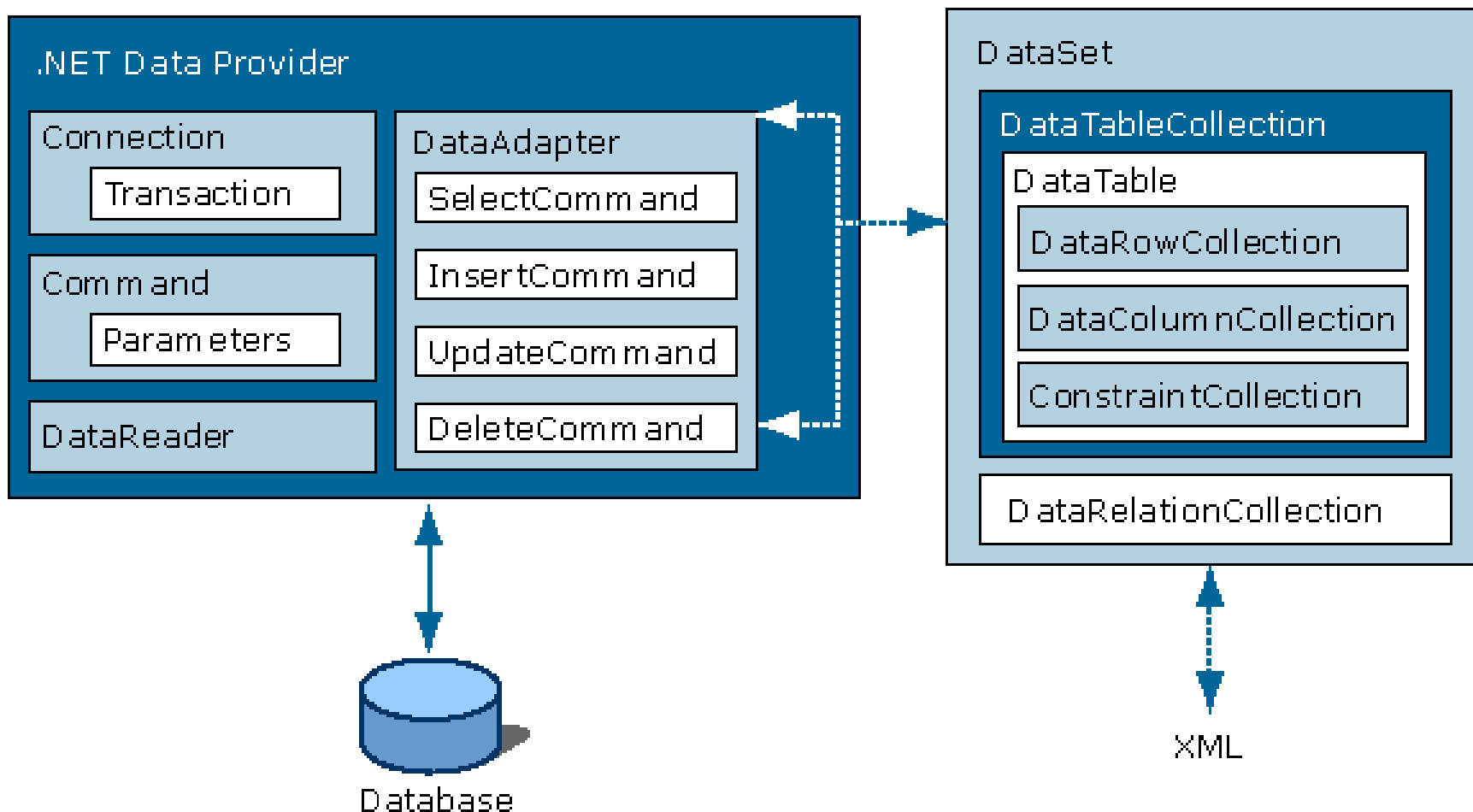
namespace AplWFACapacitacaoADONET
{
    public partial class frmBancoDados : Form
    {
        public frmBancoDados()
        {
            InitializeComponent();
        }

        private void btnSair_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

**\*\*\*\*\* IMPORTANTE \*\*\*\*\***

**Sem adicionar a referência no projeto, NÃO será possível importar a classe (Provedor) MySQL.**

# ADO.NET – Arquitetura



# ADO.NET – Modo Conectado

A arquitetura ADO.NET disponibiliza dois modos de conexão: **Conectado e Desconectado**.

No modo **Conectado** utilizamos os seguintes componentes:

- ✓ **Connection** → Classe (objeto) disponível para conexão com diversos bancos de dados.
- ✓ **Command** → Classe (objeto) disponível para enviar as instruções (select, insert, update, delete, etc) ao banco de dados. Para executar uma **Stored Procedure**, devemos utilizar uma propriedade **Parameters**, para enviar os parâmetros de entrada da **Stored Procedure**.
- ✓ **DataReader** → Classe (objeto) disponível para **receber os dados** de uma tabela após a execução de um **Select**. Este componente só funciona com a instrução **Select**, porque após a busca dos dados no banco a instrução devolverá uma resposta, se encontrou ou não, o que foi solicitado anteriormente. Toda resposta de um **Select** sempre será associado a um DataReader.



# ADO.NET – Modo Desconectado

No modo **Desconectado** utilizamos os seguintes componentes:

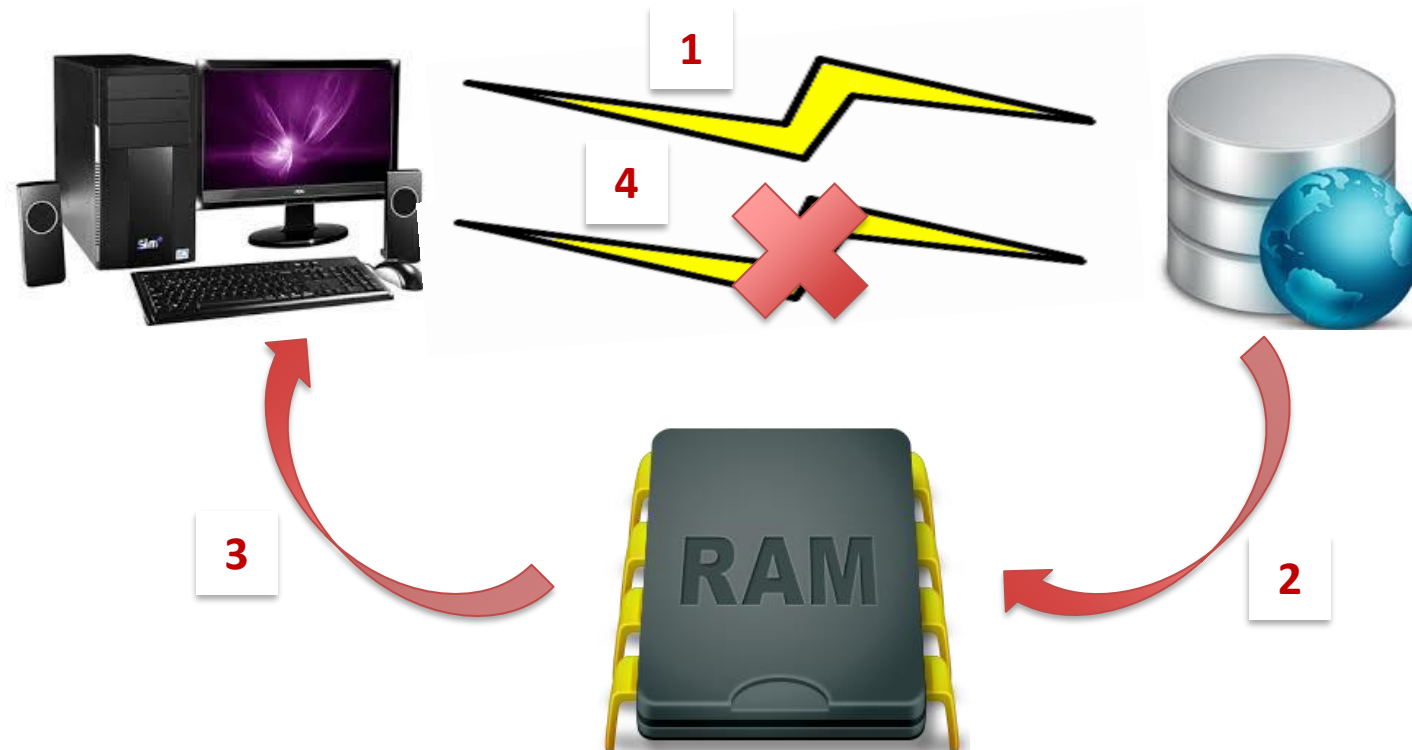
- ✓ **Connection** → Classe (objeto) disponível para conexão com diversos bancos de dados.
- ✓ **Command** → Classe (objeto) disponível para enviar as instruções (select, insert, update, delete, etc) ao banco de dados. Para executar uma **Stored Procedure**, devemos utilizar uma propriedade **Parameters**, para enviar os parâmetros de entrada da **Stored Procedure**.
- ✓ **DataAdapter** → Classe (objeto) disponível para **receber os dados** de uma tabela após a execução de um **Select**.
- ✓ **DataSet** → O objeto Recordset (ADO), que armazena somente uma coleção de tabelas, entra em desvantagem com o DataSet, membro do System.Data, o qual passa a controlar uma cópia do banco de dados em memória, representando um conjunto de dados em memória cache que não está conectado com o banco de dados.
- ✓ **DataTable** → O objeto DataTable pode representar uma ou mais tabelas de dados, as quais permanecem alocadas em memória e pode ser manipulado através de métodos.
- ✓ **DataView** → As operações de pesquisa, ordenação e navegação pelos dados, podem ser feitas através do DataView, que permite a ligação da fonte de dados com a interface do usuário, portanto, utilizamos um DataView para visualizar as informações contidas em DataTable.



# ADO.NET – Modo Desconectado

No modo **desconectado** fazemos:

1. **Conexão com o Banco de Dados;**
2. **Realizamos a busca através do DataAdapter pelo SelectCommand;**
3. **Os dados retornados ficam disponíveis na memória RAM, para consulta, entre outros;**
4. **Desconecta-se do banco de dados;**



# ADO.NET – Principais Pacotes

Os principais pacotes utilizados pelo ADO.NET, são:

- ✓ **System.Data:** contém as classes que representam tabelas, colunas, linhas e também a classe DataSet de todos os provedores, além das interfaces: IDbCommand, IDbConnection, e IDataAdapter que são usadas por todos os provedores de conexão.
- ✓ **System.Data.Common:** Define as classes para os provedores de dados: DbConnection e DbDataAdapter.
- ✓ **System.Data.OleDb:** Fonte de dados OLE DB usando o provedor .NET OleDb.
- ✓ **System.Data.Odbc:** Fonte de dados ODBC usando o provedor .NET Odbc.
- ✓ **System.Data.SqlTypes:** Dados específicos para o SQL Server.

Além disso o ADO.NET, oferece classes referenciadas:

- ✓ **Disconnected:** Fornece classes que são capazes de armazenar dados sem a dependência da fonte de dados de um determinado provedor, como por exemplo: DataTable.
- ✓ **Shared:** São classes que podem ser acessadas por todos os provedores.
- ✓ **Data Providers:** São classes utilizadas em diferentes fontes de dados, para gerenciamento.

# SQL – Grupos

As instruções SQL são divididas por grupos. São eles:

- ✓ **DTC (Data Type Commands)**
- ✓ **DDL (Data Definition Language)**
- ✓ **DQL (Data Query Language)**
- ✓ **DML (Data Manipulation Language)**
- ✓ **DCL (Data Control Language)**
- ✓ **SRC (Stored Routines Commands)**

# DTC – Data Type Commands

- ✓ **BIGINT** → Números inteiros.
- ✓ **BINARY** → Números binários.
- ✓ **BIT** → Numero inteiro. Só armazena 0 ou 1.
- ✓ **CHAR** → Caractere.
- ✓ **DATETIME** → Armazena data no formato “yyyy-mm-dd” e/ou hora no formato “HH:mm:ss”
- ✓ **DECIMAL** → Números reais. Usado para campos monetários.
- ✓ **FLOAT** → Números reais.
- ✓ **INT** → Números inteiros.
- ✓ **SMALINT** → Números inteiros.
- ✓ **TEXT** = Sequência de caracteres. Campo MEMO.
- ✓ **TINYINT** → Números inteiros.
- ✓ **VARCHAR** → Cadeia de caracteres.

# DDL - Data Definition Language

- ✓ **ALTER TABLE:** alterar a estrutura de uma tabela.
- ✓ **ALTER VIEW:** alterar a estrutura da tabela virtual.
- ✓ **CREATE DATABASE:** criar banco de dados.
- ✓ **CREATE INDEX:** criar um índice para a tabela.
- ✓ **CREATE TABLE:** criar uma tabela no banco de dados.
- ✓ **CREATE VIEW:** criar uma tabela virtual cujo conteúdo (colunas e linhas) é definido por uma consulta. Use esta instrução para criar uma exibição dos dados em uma ou mais tabelas no banco de dados.
- ✓ **DROP DATABASE:** eliminar um banco de dados.
- ✓ **DROP INDEX:** eliminar um índice de uma tabela.
- ✓ **DROP TABLE:** eliminar uma tabela do banco de dados.
- ✓ **DROP VIEW:** eliminar uma tabela virtual.
- ✓ **USE:** Abrir um banco de dados.



# DQL - Data Query Language

- ✓ **SELECT:** permite ao usuário especificar uma consulta ("query") como uma descrição do resultado desejado.

# DML – Data Manipulation Language

- ✓ **SELECT:** permite ao usuário especificar uma consulta ("query") como uma descrição do resultado desejado.
- ✓ **INSERT** é usada para inserir um registro (formalmente uma tupla) a uma tabela existente.
- ✓ **UPDATE** para mudar os valores de dados em uma ou mais linhas da tabela existente.
- ✓ **DELETE** permite remover linhas existentes de uma tabela.
- ✓ **TRUNCATE:** remove rapidamente todas as linhas da tabela, esvaziando-a.
- ✓ **COMMIT:** efetiva a transação atualmente executada.
- ✓ **ROLLBACK:** desfaz a transação corrente, fazendo com que todas as modificações realizadas pela transação sejam rejeitadas.

# DCL – Data Control Language

- ✓ **GRANT:** concede privilégios a um ou mais usuários para acessar ou realizar determinadas operações em um objetos de dados.
- ✓ **REVOKE:** revoga (remove) ou restringe a capacidade de um usuário de executar operações.
- ✓ **SET:** Define parâmetros em tempo de execução, como por exemplo, o tipo de codificação do cliente e o estilo de representação de data e hora.
- ✓ **LOCK:** Bloqueia explicitamente uma tabela fazendo o controle de acessos concorrente.

# SQL – CREATE DATABASE

Sintaxe:

**Create Database <nome-banco>;**

Exemplo:

✓ Create Database bdIntegrado;

# SQL – CREATE DATABASE e USE

Sintaxe:

**Create Database <nome-banco>;**

**Use <nome-banco>;**

Exemplo:

- ✓ Create Database bdIntegrado;
- ✓ Use bdIntegrado;



# SQL – CREATE TABLE

Sintaxe:

**Create Table <nome-tabela>**

```
(  
    <campo> <tipo-dados> not null [null] [primary key] [auto_increment] [,]  
);
```

Exemplo:

```
Create Table tblfuncionarios
```

```
(  
    funcCodigo int(11) not null primary key auto_increment,  
    funcNome varchar(50) null,  
    funcData date null  
);
```

# SQL – SELECT

Sintaxe:

**Select <campos> from <tabela> [Where <condição>] [Inner Join <tabela> On <campos>] [Group By <campo>] [Having <condição>]**

Exemplo:

- ✓ Select \* from tblfuncionários;
- ✓ Select \* from tblfuncionário Where funcCodigo = 5;
- ✓ Select \* from tblfuncionários Group By funcCargo = 'Gerente';

# SQL – INSERT

Sintaxe:

**Insert into <tabela> [<campos>] values (<valores>)**

Exemplo:

- ✓ Insert into tblfuncionários (funcCodigo,funcNome) values (1,'Gabriel');
- ✓ Insert into tblfuncionários (funcCodigo,funcNome) values (4,'Júlia');
- ✓ Insert into tblfuncionários (funcCodigo,funcNome) values (19,'Marcela');

# SQL – UPDATE

Sintaxe:

**Update <tabela> set <campo1> = <valor1> [, ..., <campoN> = <valorN>]  
[Where <condição>]**

Exemplo:

- ✓ Update tblfuncionários set funcSalario = funcSalario \* 1.20 Where funcDepto = 4;
- ✓ Update tblfuncionários set funcData = '2014-10-03' Where funcDepto = 7;

# SQL – DELETE

Sintaxe:

**Delete from <tabela> [Where <condição>]**

Exemplo:

- ✓ Delete from tblfuncionários Where funcDepto = 4;
- ✓ Delete from tblfuncionários Where funcData = '2012-08-10';



# Exemplo – ADO.NET

The image shows a screenshot of a Windows application window titled "Banco de Dados". The window contains a form with the following elements:

- A **Label** at the top center with the text "ADO.NET" in large red font.
- Five text input fields, each labeled with a field name and containing the text "TextBox":
  - Código: TextBox
  - Nome: TextBox
  - E-Mail: TextBox
  - Telefone: TextBox
  - CPF: TextBox
- Seven buttons arranged in two rows:
  - Row 1: Incluir, Alterar, Consultar, Excluir
  - Row 2: Consultar Lista de Dados, Limpar, Sair

# Exemplo – ADO.NET – Banco de Dados (Estrutura)

```
create database bdCapacitacao;  
use bdCapacitacao;  
create table tblAgenda  
(  
    agdid int(11) not null primary key,  
    agdnome varchar(50) null,  
    agdemail varchar(200) null,  
    agdtelefone varchar(15) null,  
    agdcpf varchar(15)  
);
```

# Exemplo – ADO.NET

Antes de programar os botões foram declaradas as variáveis para representar a Connection, Command e DataReader, como mostra a figura abaixo:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MySql.Data.MySqlClient;

namespace AplWFACapacitacaoADONET
{
    public partial class frmBancoDados : Form
    {
        private MySqlConnection objCnx = new MySqlConnection();
        private MySqlCommand objCmd = new MySqlCommand();
        private MySqlDataReader objDados;

        public frmBancoDados()
        {
            InitializeComponent();
        }
    }
}
```

Para declarar as variáveis como “publicas” dentro do formulário, utilizamos a área **public partial class**.

# Exemplo – ADO.NET – Conexão

A conexão com o Banco de Dados será feita no carregamento do formulário, como mostra a figura abaixo:

```
private void frmBancoDados_Load(object sender, EventArgs e)
{
    try
    {
        objCnx.ConnectionString = "Server=localhost;Database=bdCapacitacao;User=root;Pwd=master";
        objCnx.Open();
    }
    catch (Exception Erro)
    {
        MessageBox.Show("Erro ==> " + Erro.Message, "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

## Parâmetros da String de Conexão (**ConnectionString**)

**Server** → Nome do servidor de conexão. (**localhost** = servidor local onde está instalado o **MySQL Server**)

**Database** → Nome do banco de dados.

**User** → Nome do usuário que tem permissão para acessar o banco de dados.

**Pwd** → Senha (password) de acesso ao banco de dados. **Se o usuário não possuir senha, deixar o parâmetro Pwd em branco. Exemplo: "Pwd="**

# Exemplo – ADO.NET – Propriedades

Se vamos trabalhar com Banco de Dados, então, devemos entender como executamos as instruções:

- ✓ **SELECT** → Através do **Command** utilizaremos o método **ExecuteReader()**. Este método é responsável por devolver uma resposta da query enviada, ou seja, validar se encontrou algum registro e/ou editar os dados do mesmo para alteração.
- ✓ **INSERT, UPDATE e DELETE** → Através do **Command** utilizaremos o método **ExecuteNonQuery()**. Este método é responsável por executar a instrução e não devolve nenhuma resposta, portanto, não se utiliza o **DataReader()** com estas instruções.

# ADO.NET – Botão Incluir (Parte 1)

Após a execução da Query (Select) é retornado ao DataReader (objDados) os dados da busca. Através da propriedade **HasRows** validamos se encontrou ou não registro na tabela, se esta propriedade for **true** quer dizer que encontrou registro(s), caso contrário, não encontrou nenhum registro.

```
private void btnIncluir_Click(object sender, EventArgs e)
{
    try
    {
        string strSql = "Select * from tblagenda Where agdid = " + txtCodigo.Text;
        objCmd.Connection = objCnx;
        objCmd.CommandText = strSql;
        objDados = objCmd.ExecuteReader();
        if (objDados.HasRows)
        {
            MessageBox.Show("Código existente!!!", "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Information);
            txtCodigo.Focus();
        }
    }
}
```

Valida se encontrou algum registro na tabela do banco de dados.

# ADO.NET – Botão Incluir (Parte 2)

Caso o código pesquisado não exista, então, foi criada uma variável (**strSql**) que armazenará a instrução **Insert** com todos os seus parâmetros. Ao término da montagem da instrução (**Insert**), enviaremos os dados através dos seguintes comandos:

Valida se o **DataReader** está aberto, se sim, vai fechá-lo.

Como temos um **DataReader** aberto anteriormente com o resultado do **Select**, e teremos que executar outra instrução (**Insert**) em seguida, para que não ocorra erro de execução devemos fechar o **DataReader** aberto antes de executar a próxima instrução.

```
else  
{
```

```
    if (!objDados.IsClosed) { objDados.Close(); }  
    strSql = "Insert into tblagenda (agdid, agdnome, agdemail, agdtelefone, agdcpf) values (";  
    strSql += txtCodigo.Text + ",";  
    strSql += "'" + txtNome.Text + "',";  
    strSql += "'" + txtEmail.Text + "',";  
    strSql += "'" + txtTelefone.Text + "',";  
    strSql += "'" + txtCpf.Text + "')";
```



# ADO.NET – Botão Incluir (Parte 3)

Para poder enviar a instrução **Insert** ao banco de dados, devemos utilizar as seguintes propriedades do Command: **Connection**, **CommandText** e **ExecuteNonQuery**.

**Connection** → indica qual a conexão que será utilizada para executar o Insert.

**CommandText** → indica qual instrução será enviada ao banco de dados. Neste caso o Insert está armazenado na variável **strSql**.

**ExecuteNonQuery()** → método para executar algumas instruções, e após a sua execução não retorna nada do bando de dados.

```
objCmd.Connection = objCnx;  
objCmd.CommandText = strSql;  
objCmd.ExecuteNonQuery();
```

```
MessageBox.Show("Registro incluído com sucesso!!!", "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

```
}  
if (!objDados.IsClosed) { objDados.Close(); }
```

```
}  
catch (Exception Erro)
```

```
{  
    MessageBox.Show("Erro ==> " + Erro.Message, "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Error);  
}
```

Valida se o DataReader está aberto, se sim, vai fechá-lo.

# ADO.NET – Botão Alterar

O botão Alterar é similar ao botão Incluir, como mostra a figura abaixo:

```
private void btnAlterar_Click(object sender, EventArgs e)
{
    try
    {
        string strSql = "Select * from tblagenda Where agdid = " + txtCodigo.Text;
        objCmd.Connection = objCnx;
        objCmd.CommandText = strSql;
        objDados = objCmd.ExecuteReader();
        if (!objDados.HasRows)
        {
            MessageBox.Show("Código inexistente!!!", "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Information);
            txtCodigo.Focus();
        }
        else
        {
            if (!objDados.IsClosed) { objDados.Close(); }
            strSql = "Update tblagenda set ";
            strSql += "agdnome = '" + txtNome.Text + "',";
            strSql += "agdemail = '" + txtEmail.Text + "',";
            strSql += "agdtelefone = '" + txtTelefone.Text + "',";
            strSql += "agdcpf = '" + txtCpf.Text + "' ";
            strSql += "Where agdid = " + txtCodigo.Text;

            objCmd.Connection = objCnx;
            objCmd.CommandText = strSql;
            objCmd.ExecuteNonQuery();
            MessageBox.Show("Registro incluído com sucesso!!!", "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        if (!objDados.IsClosed) { objDados.Close(); }
    }
    catch (Exception Erro)
    {
        MessageBox.Show("Erro ==> " + Erro.Message, "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

# ADO.NET – Botão Excluir

No botão Excluir validamos somente se o código existe ou não, para poder excluir, como mostra a figura abaixo:

```
private void btnExcluir_Click(object sender, EventArgs e)
{
    try
    {
        string strSql = "Select * from tblagenda Where agdid = " + txtCodigo.Text;
        objCmd.Connection = objCnx;
        objCmd.CommandText = strSql;
        objDados = objCmd.ExecuteReader();
        if (!objDados.HasRows)
        {
            MessageBox.Show("Código inexistente!!!", "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Information);
            txtCodigo.Focus();
        }
        else
        {
            if (!objDados.IsClosed) { objDados.Close(); }

            if (MessageBox.Show("Deseja excluir", "ADO.NET", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2) == DialogResult.Yes)
            {
                objCmd.Connection = objCnx;
                objCmd.CommandText = strSql;
                objCmd.ExecuteNonQuery();
                MessageBox.Show("Registro eliminado com sucesso!!!", "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
        if (!objDados.IsClosed) { objDados.Close(); }
    }
    catch (Exception Erro)
    {
        MessageBox.Show("Erro ==> " + Erro.Message, "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

# ADO.NET – Botão Consultar

No botão Consultar validamos somente se o código existe ou não, se existir carregaremos os dados do registro para os componentes visuais, como mostra a figura abaixo.

```
private void btnConsultar_Click(object sender, EventArgs e)
{
    try
    {
        string strSql = "Select * from tblogenda Where agdid = " + txtCodigo.Text;
        objCmd.Connection = objCnx;
        objCmd.CommandText = strSql;
        objDados = objCmd.ExecuteReader();
        if (!objDados.HasRows)
        {
            MessageBox.Show("Código inexistente!!!", "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Information);
            txtCodigo.Focus();
        }
    }
}
```

# ADO.NET – Botão Consultar

Para resgatar os dados do registro consultado e apresentar nos componentes visuais, devemos deixá-lo no modo editável através do método **Read()**, como mostra a figura abaixo:

Método que habilita o registro no modo editável, ou seja, permite manipular todas as informações contido nele.

Para resgatar a informação de cada campo da tabela devemos usar a seguinte sintaxe:

**DataReader["<campo>"].ToString()**

```
else
{
    objDados.Read();
    txtNome.Text = objDados["agdnome"].ToString();
    txtEmail.Text = objDados["agdemail"].ToString();
    txtTelefone.Text = objDados["agdtelefone"].ToString();
    txtCpf.Text = objDados["agdcpf"].ToString();
}
if (!objDados.IsClosed) { objDados.Close(); }
}
catch (Exception Erro)
{
    MessageBox.Show("Erro ==> " + Erro.Message, "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

# ADO.NET – Botão Limpar

Limpar todos os componentes visuais do formulário.

```
private void btnLimpar_Click(object sender, EventArgs e)
{
    txtCodigo.Text = "";
    txtNome.Text = "";
    txtEmail.Text = "";
    txtTelefone.Text = "";
    txtCpf.Text = "";
    txtCodigo.Focus();
}
```

# ADO.NET – Botão Consultar Lista de Dados

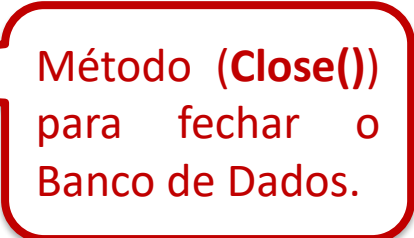
Carrega o formulário para consultar os dados da tabela no grid, como mostra a figura abaixo:

```
private void btnConsultarDados_Click(object sender, EventArgs e)
{
    frmConsultarLista objTela = new frmConsultarLista();
    objTela.ShowDialog();
}
```

# ADO.NET – Botão Sair

Antes de finalizar a aplicação, devemos fechar o banco de dados, como mostra a figura abaixo:

```
private void btnSair_Click(object sender, EventArgs e)
{
    objCnx.Close();
    Application.Exit();
}
```





# Exemplo – ADO.NET

The screenshot shows a Windows form titled "frmConsultarLista". The form has a light blue title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains the following elements:

- A red "Label" above the text "Listar Dados" in a large, bold, red font.
- A DataGridView below the text, which is currently empty. The grid has a header row with four columns: "Imagem", "Código", "Nome", and "Telefone".
- Two buttons at the bottom right: "Consultar" and "Fechar", both labeled as "Button" in red text above them.

# ADO.NET – MySQL

Todas as configurações anterior do MySQL devem ser refeitas nesta formulário, como mostra a figura abaixo:

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MySql.Data.MySqlClient;

namespace AplWFACapacitacaoADONET
{
    public partial class frmConsultarLista : Form
    {
        private MySqlConnection objCnx = new MySqlConnection();
        private MySqlCommand objCmd = new MySqlCommand();
        private MySqlDataReader objDados;
```

# ADO.NET – LOAD() do Form

No carregamento do formulário iremos abrir o banco de dados, como mostra a figura abaixo:

```
private void frmConsultarLista_Load(object sender, EventArgs e)
{
    try
    {
        objCnx.ConnectionString = "Server=localhost;Database=bdCapacitacao;User=root;Pwd=master";
        objCnx.Open();
    }
    catch (Exception Erro)
    {
        MessageBox.Show("Erro ==> " + Erro.Message, "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

# ADO.NET – Botão Consultar (Parte 1)

No botão Consultar iremos carregar os dados da tabela no grid, como mostra a figura abaixo:

```
private void btnConultar_Click(object sender, EventArgs e)
{
    try
    {
        string strSql;

        strSql = "Select * From tblagenda Order By agdnome";
        objCmd.CommandText = strSql;
        objCmd.Connection = objCnx;
        objDados = objCmd.ExecuteReader();
        if (!objDados.HasRows)
        {
            MessageBox.Show("Código Inexistente!!!", "Consultar Lista", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        else
        {
            while (objDados.Read())
            {
                dgvDados.Rows.Add(Properties.Resources.arrow_red_2, objDados["agdidd"].ToString(), objDados["agdnome"].ToString(),
            }
        }
        objDados.Close();
    }
    catch (Exception Erro)
    {
        MessageBox.Show("Erro ==> " + Erro.Message, "ADO.NET", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

# ADO.NET – Botão Consultar (Parte 2)

Linha de comando para preencher o grid com os dados da tabela.

**Continuação:**

```
objDados["agdtelefone"].ToString(), objDados["agdcpf"].ToString());
```

**Linha Completa:**

```
dgvDados.Rows.Add(Properties.Resources.arrow_red_2,  
objDados["agdid"].ToString(), objDados["agdnome"].ToString(),  
objDados["agdtelefone"].ToString(), objDados["agdcpf"].ToString());
```

# ADO.NET – Botão Fechar

Antes de fechar o formulário, devemos fechar o banco de dados, como mostra a figura abaixo:

```
private void btnFechar_Click(object sender, EventArgs e)
{
    objCnx.Close();
    this.Close();
}
```

Método (**Close()**)  
para fechar o  
Banco de Dados.

Método (**Close()**)  
para fechar o  
Formulário corrente..

# Referências

[http://www.lrocha.com.br/arquivos/arquivos/BdWeb%20\(PostgreSQL\)/AULAS/bd\\_web\\_A4.pdf](http://www.lrocha.com.br/arquivos/arquivos/BdWeb%20(PostgreSQL)/AULAS/bd_web_A4.pdf)

[http://www.macoratti.net/08/11/c\\_adn\\_1.htm](http://www.macoratti.net/08/11/c_adn_1.htm)

<http://msdn.microsoft.com/pt-br/library/hh972566.aspx>

<http://www.csharp-station.com/Tutorial/AdoDotNet>