Apostila de ASP.NET

1. Tecnologia ADO.NET

Relembrando sobre a tecnologia de banco de dados. 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:

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

Figura 1 – Provedores de acesso ADO.NET

Estes são os provedores nativos da tecnologia ADO.NET. Caso seja necessário utilizar um provedor externo, por exemplo MySQL, então devemos importar a classe corresponde para o ambiente de desenvolvimento. Nesta, apostila utilizaremos o banco de dados **MySQL** como banco principal dos projetos.

1.1. Criando a classe BancoDados

Vamos criar o projeto do tipo **Class Library**, com o nome **BancoDados**, e devemos definir os atributos como mostra a figura 2.

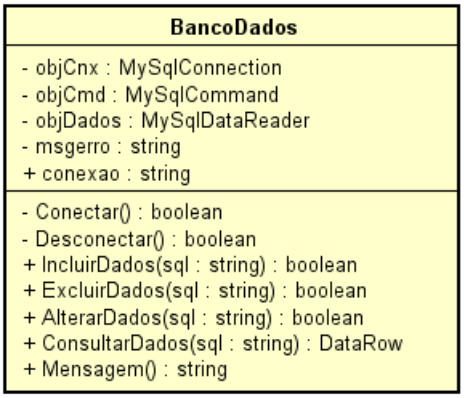

Figura 2 - Diagrama da classe BancoDados

1.2. Tipos de classes com Banco de Dados

Podemos desenvolver uma classe para Banco de Dados de três formas:

1.2.1. Classe Monolíticas ou 1-Tier

Neste modelo todas as camadas, Acesso a Dados, Regras de Negócio ou Persistência e Apresentação, são definidas somente em uma camada, como mostraremos a seguir:

1. **Camada de Acesso a Dados, Regras de Negócio ou Persistência e Apresentação**

   É a camada onde iremos exibir informações e coletar dados do usuário, por exemplo as páginas **.aspx**. Além disso, escrevemos a lógica, as regras de negócio da aplicação e comunicação direta da aplicação com o back-end, realizando transações, como inclusão, consulta de dados, etc.

1.2.2. Classe 2 camadas ou 2-Tier

Neste modelo as camadas são definidas em Acesso a Dados, Regras de Negócio ou Persistência e Apresentação, como mostraremos a seguir:

1. **Camada de Acesso a Dados**

   Camada responsável pela comunicação direta da aplicação com o back-end, realizando transações, como inclusão, consulta de dados, etc.

2. **Camada de Negócio ou Persistência e Apresentação**

   É a camada onde iremos exibir informações e coletar dados do usuário, por exemplo as páginas **.aspx**. Além disso, escrevemos a lógica e as regras de negócio da aplicação.

1.2.3. Classe 3 camadas ou 3-Tier

Neste modelo temos as seguintes camadas:

1. **Camada de Acesso a Dados**

   Camada responsável pela comunicação direta da aplicação com o back-end, realizando transações, como inclusão, consulta de dados, etc.

2. **Camada de Negócio ou Persistência**

   É onde escrevemos a lógica e as regras de negócio da aplicação. É a camada intermediária entre o usuário e o back-end.

3. **Camada de Apresentação**

   É a camada onde iremos exibir informações e coletar dados do usuário, por exemplo as páginas **.aspx**.

Agora que conhecemos as definições de camadas o Diagrama de Classe (figura 2) representa um modelo de classe 2 camadas.

Vamos criar o projeto do tipo **Class Library**, com o nome **BancoDados**, e devemos definir os atributos como mostra a figura 3.

```
private MySqlConnection objCnx = new MySqlConnection();
private MySqlCommand objCmd = new MySqlCommand();
private MySqlDataReader objDados;
private string msgerro = "";
public string conexao { private get; set; }
```

Figura 3 - Programação da classe BancoDados

Em seguida iremos criar os métodos baseado no diagrama de classe conforme a figura 2.

O método **Conectar()**, será responsável pela conexão ao banco de dados, como mostra a figura 4.

```
private bool Conectar()
{
    try
    {
        objCnx.ConnectionString = conexao;
        objCnx.Open();
        return true;
    }
    catch (Exception Erro)
    {
        msgerro = Erro.Message;
        return false;
    }
}
```

Figura 4 - Programação do Método Conectar()

O método **Desconectar()**, será responsável pela desconexão do banco de dados, como mostra a figura 5.

```
private bool Desconectar()
{
    try
    {
        if (objCnx.State == ConnectionState.Open)
        {
            objCnx.Close();
        }
        return true;
    }
    catch (Exception Erro)
    {
        msgerro = Erro.Message;
        return false;
    }
}
```

Figura 5 - Programação do Método Desconectar()

O método **IncluirDados()**, será responsável pela inclusão dos dados ao banco de dados, como mostra a figura 6.

```
public bool IncluirDados(string sql)
{
    try
    {
        bool blnRetorno = false;
        if (Conectar())
        {
            objCmd.Connection = objCnx;
            objCmd.CommandText = sql;
            int result = objCmd.ExecuteNonQuery();
            if (result > 0)
            {
                blnRetorno = true;
            }
            else
            {
                msgerro = "Erro na inclusão de dados";
            }
        }
        Desconectar();
        return blnRetorno;
    }
    catch (Exception Erro)
    {
        msgerro = Erro.Message;
        return false;
    }
}
```

Figura 6 - Programação do Método IncluirDados()

O método **AlterarDados()**, será responsável pela alteração dos dados ao banco de dados, como mostra a figura 7.

```
public bool AlterarDados(string sql)
{
    try
    {
        bool blnRetorno = false;
        if (Conectar())
        {
            objCmd.Connection = objCnx;
            objCmd.CommandText = sql;
            int result = objCmd.ExecuteNonQuery();
            if (result > 0)
            {
                blnRetorno = true;
            }
            else
            {
                msgerro = "Erro na alteração de dados";
                blnRetorno = false;
            }
        }
        Desconectar();
        return blnRetorno;
    }
    catch (Exception Erro)
    {
        msgerro = Erro.Message;
        return false;
    }
}
```

Figura 7 - Programação do Método AlterarDados()

O método **ExcluirDados()**, será responsável pela eliminação dos dados ao banco de dados, como mostra a figura 8.

```
public bool ExcluirDados(string sql)
{
    try
    {
        bool blnRetorno = false;
        if (Conectar())
        {
            objCmd.Connection = objCnx;
            objCmd.CommandText = sql;
            int result = objCmd.ExecuteNonQuery();
            if (result > 0)
            {
                blnRetorno = true;
            }
            else
            {
                msgerro = "Erro na exclusão de dados";
                blnRetorno = false;
            }
        }
        Desconectar();
        return blnRetorno;
    }
    catch (Exception Erro)
    {
        msgerro = Erro.Message;
        return false;
    }
}
```

Figura 8 - Programação do Método ExcluirDados()

O método **ConsultarDados()**, será responsável pela busca dos dados ao banco de dados. Neste caso o retorno será feito através de um **DataRow**, porque quando se retorna um DataReader através da classe, não é possível recuperar as informações na camada de apresentação, ou seja, no formulário **.aspx**, como mostra a figura 9.

```
public DataRow ConsultarDados(string sql)
{
    try
    {
        DataRow drPessoa = null;
        if (Conectar())
        {
            objCmd.Connection = objCnx;
            objCmd.CommandText = sql;
            objDados = objCmd.ExecuteReader();
            if (objDados.HasRows)
            {
                DataTable dtPessoa = new DataTable();
                dtPessoa.Columns.Add("idpessoa");
                dtPessoa.Columns.Add("nomepessoa");
                dtPessoa.Columns.Add("emailpessoa");
                dtPessoa.Columns.Add("telefonepessoa");

                drPessoa = dtPessoa.NewRow();
                objDados.Read();
                drPessoa["idpessoa"] = objDados["idpessoa"].ToString();
                drPessoa["nomepessoa"] = objDados["nomepessoa"].ToString();
                drPessoa["emailpessoa"] = objDados["emailpessoa"].ToString();
                drPessoa["telefonepessoa"] = objDados["telefonepessoa"].ToString();
                dtPessoa.Rows.Add(drPessoa);
            }
            else
            {
                msgerro = "Erro na consulta de dados";
            }
        }
        Desconectar();
        return drPessoa;
    }
    catch (Exception Erro)
    {
        msgerro = Erro.Message;
        return null;
    }
}
```

Figura 9 - Programação do Método ConsultarDados()

O método **Mensagem()**, será responsável pelo retorno do erros tratados dentro da classe, como mostra a figura 10.

```
public string Mensagem()
{
    return msgerro;
}
```

Figura 10 - Programação do Método Mensagem()

Antes de utilizarmos a classe devemos inseri-la ao nosso projeto através da **References**, localizada na aba **Solution Explorer** (vide figura 13 da aula 2).

Após a inserção da biblioteca devemos importa-la ao ambiente de programação como mostra a figura 11.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using AplClsBancoDados;
using System.Data;
```

Figura 11 - Importação da biblioteca de Banco de Dados

Após importação da biblioteca, devemos programar os botões de manipulação do banco de dados. Primeiramente ao carregar a página (**Page_Load**) vamos definir a conexão ao servidor de banco de dados, como mostra a figura12.

```
protected void Page_Load(object sender, EventArgs e)
{
    objBancoDados.conexao = "Server=localhost;Database=bdcapacitacao;user=root;password=master";
}
```

Figura 12 - String de conexão para o servidor de banco de dados

1.3. Programação dos botões

1.3.1. Botão Limpar

```
protected void btnLimpar_Click(object sender, ImageClickEventArgs e)
{
    txtCodigo.Text = "";
    txtNome.Text = "";
    txtEmail.Text = "";
    txtFone.Text = "";
    lblMensagem.Text = "";
    txtCodigo.Focus();
}
```

Figura 13 - Programação botão Limpar

1.3.2. Botão Consultar

```
protected void btnConsultar_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        DataRow drDados;
        string strSql;
        strSql = "Select * From tblagenda Where idpessoa = " + txtCodigo.Text;
        drDados = objBancoDados.ConsultarDados(strSql);
        if (drDados != null)
        {
            txtCodigo.Text = drDados["idpessoa"].ToString();
            txtNome.Text = drDados["nomepessoa"].ToString();
            txtEmail.Text = drDados["emailpessoa"].ToString();
            txtFone.Text = drDados["telefonepessoa"].ToString();
        }
        else
        {
            lblMensagem.Text = objBancoDados.Mensagem();
            txtCodigo.Focus();
        }
    }
    catch (Exception Erro)
    {
        lblMensagem.Text = Erro.Message.ToString();
    }
}
```

Figura 14 - Programação botão Consultar

1.3.3. Botão Cadastrar

```
protected void btnCadastrar_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        string strSql = "Insert into tblagenda (idpessoa,nomepessoa,emailpessoa,telefonepessoa) values (";
        strSql += txtCodigo.Text + ",";
        strSql += "'" + txtNome.Text + "',";
        strSql += "'" + txtEmail.Text + "',";
        strSql += "'" + txtFone.Text + "')";
        if (objBancoDados.IncluirDados(strSql))
        {
            lblMensagem.Text = "Dados inclu&iacute;dos com sucesso!!!";
        }
        else
        {
            lblMensagem.Text = objBancoDados.Mensagem();
        }
    }
    catch (Exception Erro)
    {
        lblMensagem.Text = Erro.Message.ToString();
    }
}
```

Figura 15 - Programação botão Cadastrar

1.3.4. Botão Alterar

```
protected void btnAlterar_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        string strSql = "Update tblagenda set ";
        strSql += "nomepessoa = '" + txtNome.Text + "',";
        strSql += "emailpessoa = '" + txtEmail.Text + "',";
        strSql += "telefonepessoa = '" + txtFone.Text + "' ";
        strSql += "Where idpessoa = " + txtCodigo.Text;
        if (objBancoDados.AlterarDados(strSql))
        {
            lblMensagem.Text = "Dados alterados com sucesso!!!";
        }
        else
        {
            lblMensagem.Text = objBancoDados.Mensagem();
        }
    }
    catch (Exception Erro)
    {
        lblMensagem.Text = Erro.Message.ToString();
    }
}
```

Figura 16 - Programação botão Alterar

1.3.5. Botão Excluir

```
protected void btnExcluir_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        string strSql = "Delete from tblagenda Where idpessoa = " + txtCodigo.Text;
        if (objBancoDados.ExcluirDados(strSql))
        {
            lblMensagem.Text = "Dados exclu&iacute;do com sucesso!!!";
        }
        else
        {
            lblMensagem.Text = objBancoDados.Mensagem();
        }
    }
    catch (Exception Erro)
    {
        lblMensagem.Text = Erro.Message.ToString();
    }
}
```

Figura 17 - Programação botão Excluir

1.4. Estrutura do Banco de Dados

```
CREATE DATABASE bdcapacitacao;
USE bdcapacitacao;

CREATE TABLE tblagenda (
  idpessoa int(11) NOT NULL primary key,
  nomepessoa varchar(50) NULL,
  emailpessoa varchar(300) NULL,
  telefonepessoa varchar(15) NULL
);
```

Figura 18 - Estrutura do banco de dados