

Administração Central
Cetec Capacitações

Apostila de ASP.NET

Administração Central
Cetec Capacitações

1 Tecnologia 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:

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.

Tabela 53 – 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 correspondente para o ambiente de desenvolvimento. Nesta, apostila utilizaremos o banco de dados **MySQL** como banco principal dos projetos.

1.1 Instalando o MySQL

Para instalar o MySQL, devemos fazer o download no site oficial <http://dev.mysql.com/downloads/mysql/>. Após o download deve realizar a

Administração Central
Cetec Capacitações

instalação do MySQL Server e todos os seus conectores, que poderão ser utilizados na linguagem C#, C++, ODBC, etc.

1.2 O provedor 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 instalado no computador. Por exemplo:

C:\Program Files (x86)\MySQL\Connector NET 6.9\Assemblies\v2.0\MySQL.Data.Dll
C:\Program Files (x86)\MySQL\Connector NET 6.9\Assemblies\v4.0\MySQL.Data.Dll
C:\Program Files (x86)\MySQL\Connector NET 6.9\Assemblies\v4.5\MySQL.Data.Dll

Figura 74 – Versões de Provedores MySQL

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
- Visual Studio 2015 → Framework Versão 4.5

Versão do Visual Studio	Framework
Visual Studio 2005	Framework Versão 2.0
Visual Studio 2008	Framework Versão 3.5
Visual Studio 2010	Framework Versão 4.0

Administração Central
Cetec Capacitações

Visual Studio 2012	Framework Versão 4.5
Visual Studio 2013	Framework Versão 4.5
Visual Studio 2015	Framework Versão 4.5 ou superior

Tabela 54 – Versões do Framework do Visual Studio

Antes de utilizar os recursos do provedor, é preciso importá-lo ao projeto. 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 na opção **References**, e aparecerá a seguinte tela:

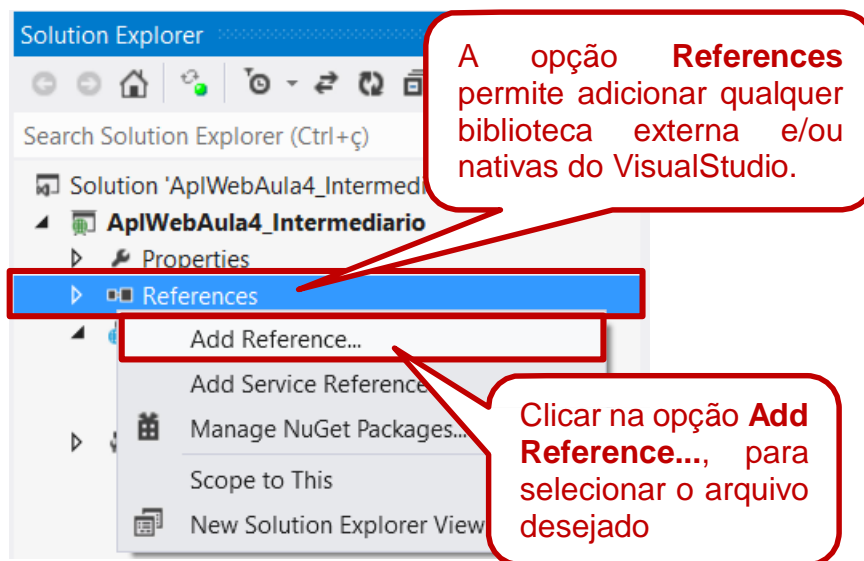


Figura 75 – Adicionando uma referência (Add Reference...)

Administração Central Cetec Capacitações

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

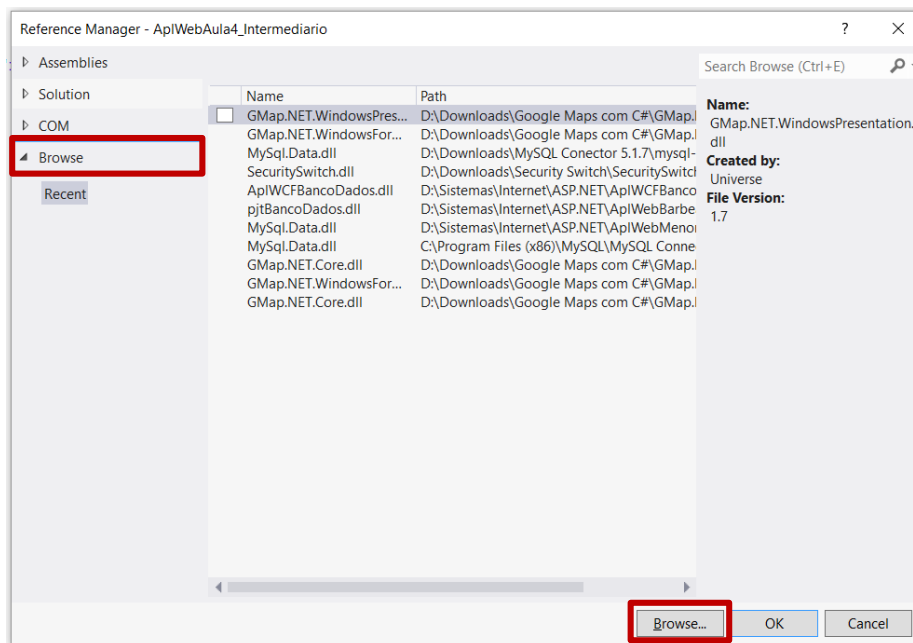


Figura 76 – Busca de Referências externas (Add Rederences...)

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

3. Após clicar no botão **Browse**, aparecerá a seguinte tela:
- a. Selecione o arquivo **MySQL.Data.Dll**, a partir da pasta que estão instalados os conectores do MySQL.
 - b. Selecionar o conector para o ambiente **.NET**, como mostra a figura.
 - c. Selecionar a versão do conector de acordo com a versão do Visual Studio, conforme a figura 77. Em seguida clicar no botão **Add**.

Administração Central Cetec Capacitações

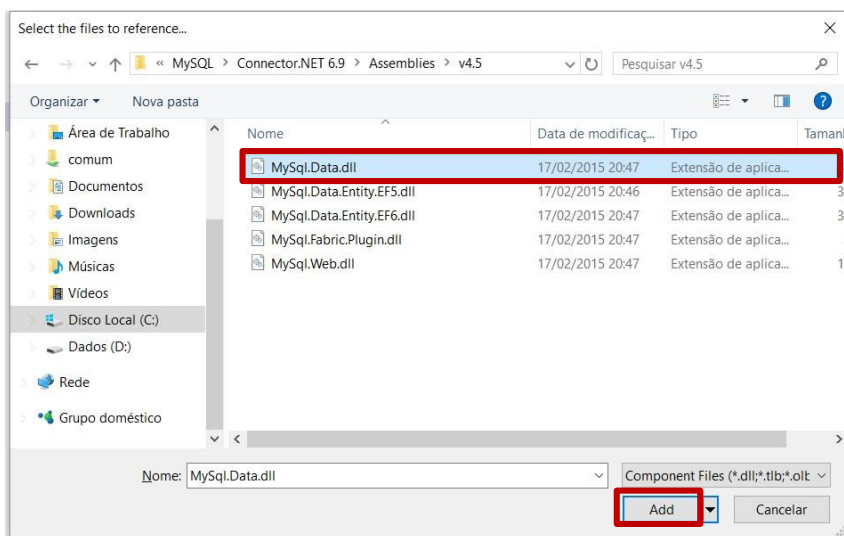


Figura 77 – Selecionando uma referência externa (Add References...)

4. Após a seleção do arquivo (.dll), irá aparecer no gerenciamento de referências, como mostra a tela abaixo:
 - a. Em seguida clicar no botão **OK**.

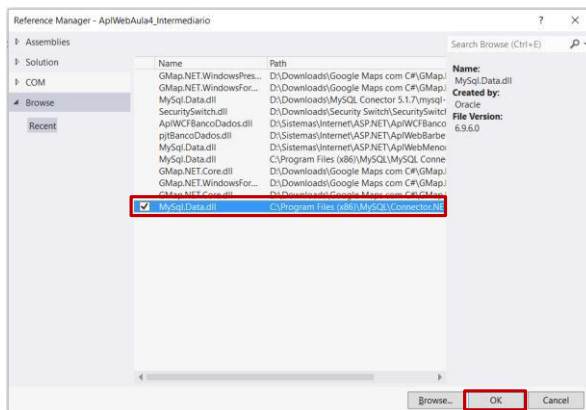


Figura 78 – Confirmando a seleção uma referência externa (Add References...)

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

Administração Central
Cetec Capacitações

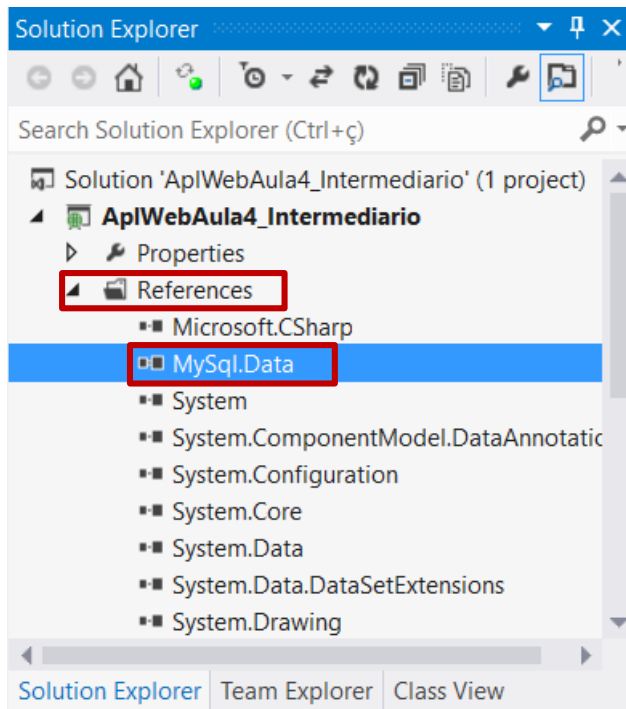


Figura 79 – Relação de Referências (References)

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

Sempre que abrir um projeto no qual a referência MySQL.Data.Dll foi inserida, não será mais necessário adicioná-la.

1.4 Importando a biblioteca ao ambiente de desenvolvimento

Para importarmos a biblioteca para o ambiente de desenvolvimento, devemos utilizar a instrução **using**, como mostra a figura 80:

Administração Central Cetec Capacitações

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using MySql.Data.MySqlClient;

namespace AplWebAula4_Intermediario
{
    public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
    }
}

```

Figura 80 – Importando provedor MySQL

Não esqueçam, sem adicionar a referência no projeto, NÃO será possível importar a classe (Provedor) MySQL

1.5 Arquitetura ADO.NET

O diagrama a seguir ilustra as relações do provedor de dados .NET Framework.

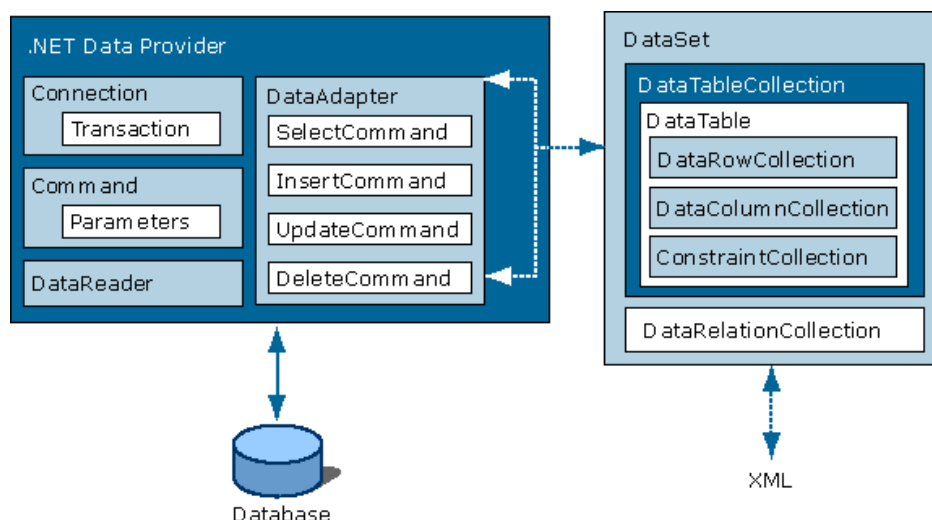


Figura 81 – Arquitetura ADO.NET (Fonte: Site Macoratti)

Administração Central
Cetec Capacitações

Como o provedor de dados disponibiliza os objetos **Connection**, **Command** e **DataReader**, os provedores SQL Server, MySQL, etc. irão diferenciar os seus objetos adicionando o seu prefixo relativo. Por exemplo, se está usando o provedor SQL Server na programação, os objetos mencionados anteriormente, devem ficar assim: **SQLConenction**, **SQLCommand**, **SQLDataReader**. Caso seja o MySQL, **MySQLConnection**, **MySQLCommand** e **MySQLDataReader**. Cada provedor tem o seu prefixo relacionado.

1.6 Modos de conexões - ADO.NET

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.

Administração Central
Cetec Capacitações

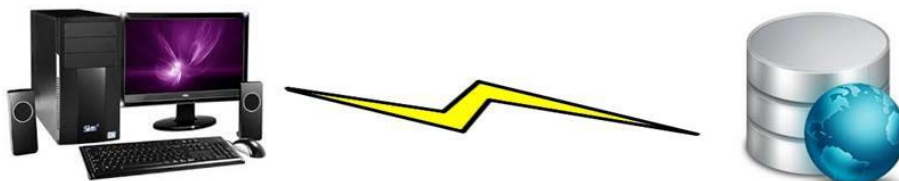


Figura 82 – Modo Conectado – ADO.NET

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.

Administração Central
Cetec Capacitações

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;

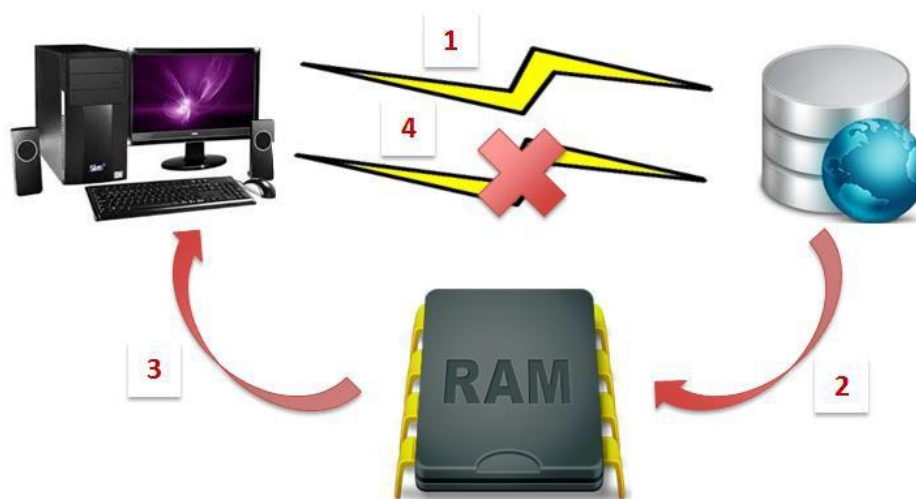


Figura 83 – Modo Desconectado – ADO.NET

1.7 Principais pacotes ADO.NET

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 IDbDataAdapter 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

Administração Central
Cetec Capacitações

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.

1.8 Revisão das instruções SQL

Para desenvolver aplicações com banco de dados é preciso entender o básico das instruções de SQL. 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)**

A tabela a seguir descreve as instruções do grupo DTC – Data Type Commands.

Instruções	Descrição
BIGINT	Números inteiros

Administração Central
Cetec Capacitações

BINARY	Números binários.
BIT	Número 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.

Tabela 55 – Grupo DTC – Data Type Commands

A tabela a seguir descreve as instruções do grupo DDL – Data Definition Language.

Instruções	Descrição
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

Administração Central
Cetec Capacitações

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

Tabela 56 – Grupo DDL – Data Definition Language

A tabela a seguir descreve as instruções do grupo DQL – Data Query Language.

Instruções	Descrição
SELECT	Permite ao usuário especificar uma consulta ("query") como uma descrição do resultado desejado.

Tabela 57 – Grupo DQL – Data Query Language

A tabela a seguir descreve as instruções do grupo DML – Data Manipulation Language.

Instruções	Descrição
SELECT	Permite ao usuário especificar uma consulta ("query") como uma descrição do resultado desejado.
INSERT	Permite inserir um registro (formalmente uma tupla) a uma tabela existente.

Administração Central
Cetec Capacitações

UPDATE	Permite alterar os valores de dados em uma ou mais linhas da tabela existente.
DELETE	Permite remover linhas existentes de uma tabela.
TRUNCATE	Permite remover todas as linhas da tabela, esvaziando-a.
COMMIT	Permite efetivar a transação atualmente executada.
ROLLBACK	Permite desfazer a transação corrente, fazendo com que todas as modificações realizadas pela transação sejam rejeitadas.

Tabela 58 – Grupo DML – Data Manipulation Language

A tabela a seguir descreve as instruções do grupo DCL – Data Control Language

Instruções	Descrição
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

Tabela 59 – Grupo DCL – Data Control Language

1.9 Sintaxe das instruções SQL

1.9.1 CREATE DATABASE

Sintaxe:

➤ **Create Database <nome-banco>;**

Administração Central
Cetec Capacitações

Exemplo:

- **Create Database bdIntegrado;**

1.9.2 CREATE DATABASE e USE

Sintaxe:

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

Exemplo:

- **Create Database bdIntegrado;**
- **Use bdIntegrado;**

1.9.3 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
);
```

Administração Central
Cetec Capacitações

1.9.4 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';**

1.9.5 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');**

1.9.6 UPDATE

Sintaxe:

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

Administração Central
Cetec Capacitações

Exemplo:

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

1.9.7 DELETE

Sintaxe:

Delete from <tabela> [Where <condição>]

Exemplo:

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

2 Exemplo de projeto com banco de dados

Primeiramente iremos desenvolver o layout da página. Como mostra a figura



Figura 84 – Layout do exemplo de Banco de Dados

Administração Central **Cetec Capacitações**

Após a criação do layout, iremos criar o banco de dados, a partir do script a seguir.

```

create database bdcapacitacao;
use bdcapacitacao;
create table tblagenda(
    agdid int(11) not null primary key,
    agdnome varchar(50) null,
    agdtelefone varchar(15) null
);
    
```

Figura 85 – Script do Banco de Dados exemplo

Para criarmos o Banco de Dados e a tabela, utilizaremos a aplicação **MySQL Workbench**. Na figura 86, mostra a tela principal da aplicação, na qual lista todas as conexões que foram criadas no ambiente.

Para acessar o localhost (Servidor Local), área onde serão criados o Banco de Dados e a tabela, temos que clicar sobre a instância **localhost**. Como mostra a figura 86.

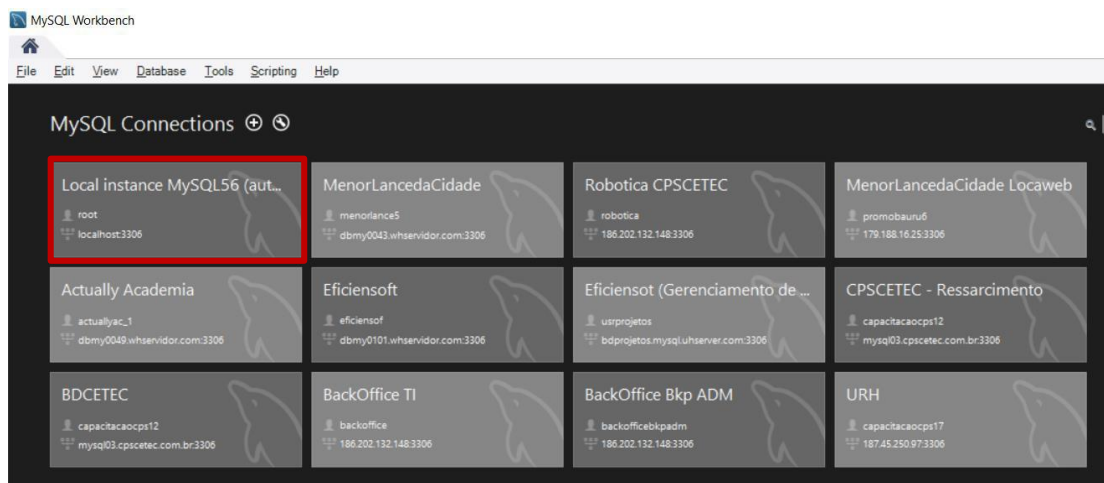


Figura 86 – MySQLWorkbench – Lista das conexões disponíveis

Após o servidor (**localhost**) estar carregado o ambiente está pronto para trabalhar, como mostra a figura 87.

Administração Central Cetec Capacitações

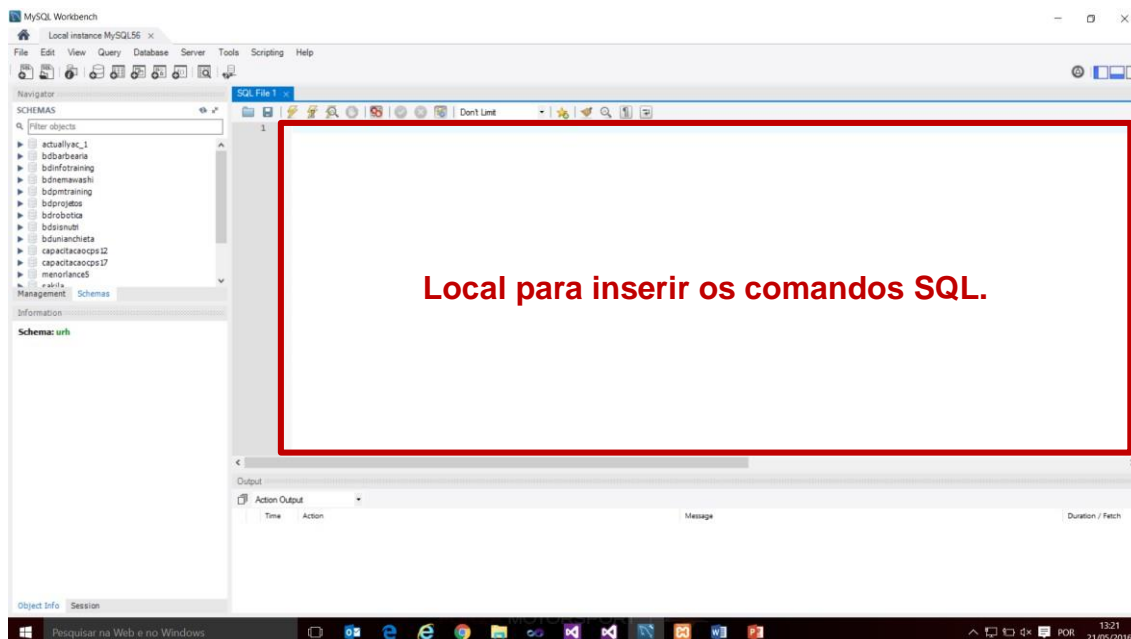


Figura 87 – MySQLWorkbench – Servidor localhost carregado

Como mostramos na **figura 85**, iremos inserir o script no local indicado, com o mostra a figura 88.

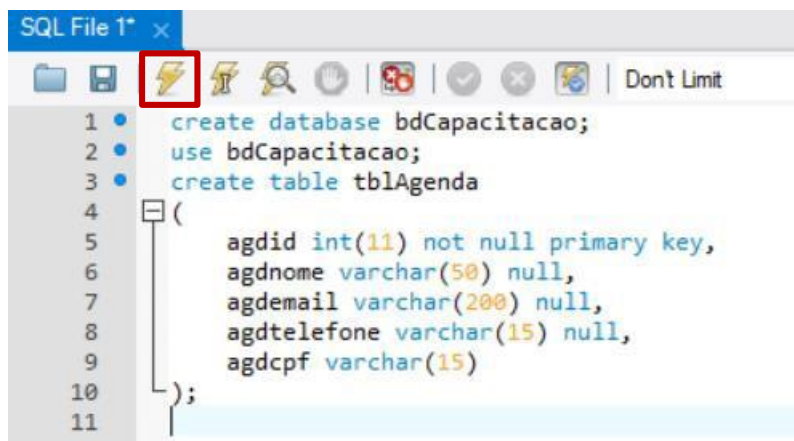



Figura 89 - MySQLWorkbench com script de banco de dados

Após digitar as instruções do script, clicar no botão , para executar as instruções, e assim criando o banco de dados e a tabela correspondente.

Nesta etapa o banco de dados está pronto para ser vinculado ao projeto. Antes de programar os botões do formulário, temos que declarar todas as variáveis que representam a **Connection**, o **Command** e o **DataReader**, como públicas.

Administração Central Cetec Capacitações

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

Assim, todos os métodos do formulário poderão reconhecer-las, como mostra a figura 91.

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

namespace AplWebAula4_Intermediario
{
    public partial class frmPrincipal : System.Web.UI.MasterPage
    {
        public MySqlConnection pblCnx = new MySqlConnection();
        public MySqlCommand pblCmd = new MySqlCommand();
        public MySqlDataReader pblDados;
```

Figura 90 – Declaração das variáveis públicas do MySQL (ADO.NET)

No momento que a página principal for carregada, queremos que o banco de dados esteja conectado e pronto para usá-lo. Para isso, devemos usar o evento **Page_Load (Carregamento da página)**, como mostra a figura 91.

```
protected void Page_Load(object sender, EventArgs e)
{
    try
    {
        pblCnx.ConnectionString = "Server=localhost;Database=bdcapitacao;user=root;password=master";
        pblCnx.Open();
    }
    catch (Exception Erro)
    {
        lblMensagem.Text = Erro.Message.ToString();
    }
}
```

Figura 91 – Conexão com o Banco de Dados (Page_Load)

Para enviar os dados de conexão para o banco de dados, vamos utilizar o método **ConnectionString**, e com isso devemos utilizar os seguintes parâmetros:

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

Administração Central
Cetec Capacitações

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="**

O código acima demonstra que ao carregar a página será feita a conexão com o banco de dados, caso retorne alguma exceção, a label **lblMensagem**, irá exibir a mensagem de erro.

Para manipular os dados armazenados no banco de dados, devemos entender como funciona algumas instruções SQL com os objetos **Connection**, **Command** e **DataReader**, como mostra a seguir:

- **SELECT** → Através do objeto **Command** utilizaremos o método **ExecuteReader()**. Este método é responsável por devolver uma resposta da query solicitada, ou seja, retorna um Reader com ou sem registros encontrados.
- **INSERT, UPDATE e DELETE** → Através do objeto **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.

A partir de agora, iremos programar os botões do nosso projeto, como mostra a seguir.

Administração Central Cetec Capacitações

2.1 Botão Incluir

```
protected void btnCadastrar_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        string strSql;

        strSql = "Select * From tblagenda Where agdid = " + txtCodigo.Text;

        pblCmd.Connection = pblCnx;
        pblCmd.CommandText = strSql;
        pblDados = pblCmd.ExecuteReader();
        if (!pblDados.HasRows)
        {
```

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

Figura 92 – Método HasRows (ADO.NET)

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

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.

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

strSql = "Insert into tblagenda (agdid,agdnome,agdtelefone) values (";
strSql += txtCodigo.Text + ",";
strSql += "'" + txtNome.Text + "',";
strSql += "'" + txtFone.Text + "')";
```

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

Figura 93 – Validar de DataReader aberto

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 do método **ExecuteNonQuery()**, como mostra a figura 94:

Administração Central Cetec Capacitações

Para entender melhor o envio dos dados ao BD, a variável que representa o objeto **Command (pblCmd)**, possui alguns métodos que irá disparar a solicitação do **Insert**. São eles:

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 banco de dados.

```
strSql = "Insert into tblagenda (agdid,agdnome,agdtelefone) values (";
strSql += txtCodigo.Text + ",";
strSql += "'" + txtNome.Text + "',";
strSql += "'" + txtFone.Text + "')";
pblCmd.Connection = pblCnx;
pblCmd.CommandText = strSql;
pblCmd.ExecuteNonQuery();
lblMensagem.Text = "Registro incluído com sucesso !!!";

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

Figura 94 – Envio da instrução (Insert) o para o Banco de Dados

2.2 Botão Alterar

O botão alterar é similar ao botão incluir, na verdade no botão incluir será verificado se o código (**agdid**) existe, através do método **HasRows**, caso não exista ele irá incluí-lo. No botão alterar será verificado se o código (**agdid**) existe, caso positivo, então os dados serão modificados. Como mostra a figura 95.

Administração Central Cetec Capacitações

```
protected void btnAlterar_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        string strSql;

        strSql = "Select * From tblogenda Where agdid = " + txtCodigo.Text;

        pblCmd.Connection = pblCnx;
        pblCmd.CommandText = strSql;
        pblDados = pblCmd.ExecuteReader();
        if (pblDados.HasRows)
        {
            if (!pblDados.IsClosed) { pblDados.Close(); }

            strSql = "Update tblogenda set ";
            strSql += "agdnome = '" + txtNome.Text + "',";
            strSql += "agdtelefone = '" + txtFone.Text + "' ";
            strSql += "Where agdid = " + txtCodigo.Text;
            pblCmd.Connection = pblCnx;
            pblCmd.CommandText = strSql;
            pblCmd.ExecuteNonQuery();
            lblMensagem.Text = "Registro alterado com sucesso !!!";
        }
        else
        {
            lblMensagem.Text = "Código Inexistente !!!";
            txtCodigo.Focus();
        }
    }
    catch (Exception Erro)
    {
        lblMensagem.Text = Erro.Message.ToString();
    }
    finally
    {
        if (!pblDados.IsClosed) { pblDados.Close(); }
    }
}
```

Figura 95 – Código-fonte do botão Alterar

2.3 Botão Excluir

O botão excluir será verificado se o código (**agdid**) existe, através do método **HasRows**, caso positivo, será eliminado da tabela. Lembre-se uma vez eliminado, **não** recupera mais os dados. Como mostra a figura 96.

Administração Central Cetec Capacitações

```
protected void btnExcluir_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        string strSql;

        strSql = "Select * From tblogenda Where agdid = " + txtCodigo.Text;

        pblCmd.Connection = pblCnx;
        pblCmd.CommandText = strSql;
        pblDados = pblCmd.ExecuteReader();
        if (pblDados.HasRows)
        {
            if (!pblDados.IsClosed) { pblDados.Close(); }
            strSql = "Delete From tblogenda Where agdid = " + txtCodigo.Text;
            pblCmd.Connection = pblCnx;
            pblCmd.CommandText = strSql;
            pblCmd.ExecuteNonQuery();
            lblMensagem.Text = "Registro excluído com sucesso !!!";
        }
        else
        {
            lblMensagem.Text = "Código Inexistente !!!";
            txtCodigo.Focus();
        }
    }
    catch (Exception Erro)
    {
        lblMensagem.Text = Erro.Message.ToString();
    }
}
```

Figura 96 – Código-fonte do botão Excluir

2.4 Botão Consultar

O botão consultar será verificado se o código (**agdid**) existe, através do método **HasRows**, caso positivo, os dados serão carregados para os campos do formulário. Para resgatar os dados do registro consultado e apresentar nos componentes visuais, devemos deixar o registro no modo editável através do método **Read()**, como mostra a figura 97.

Administração Central
Cetec Capacitações

```
protected void btnConsultar_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        string strSql;

        strSql = "Select * From tblagenda Where agdid = " + txtCodigo.Text;

        pblCmd.Connection = pblCnx;
        pblCmd.CommandText = strSql;
        pblDados = pblCmd.ExecuteReader();
        if (pblDados.HasRows)
        {
            pblDados.Read();
            txtCodigo.Text = pblDados["agdid"].ToString();
            txtNome.Text = pblDados["agdnome"].ToString();
            txtFone.Text = pblDados["agdtelefone"].ToString();
        }
        else
        {
            lblMensagem.Text =
            txtCodigo.Focus();
        }
    }
    catch (Exception Erro)
    {
        lblMensagem.Text = Erro.Message.ToString();
    }
    finally
    {
        if (!pblDados.IsClosed) { pblDados.Close(); }
    }
}
```

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

Para regatar a informação de cada campo da tabela devemos usar a seguinte sintaxe:
DataReader["<campo>"].ToString()

Figura 97 – Código-fonte do botão Consultar

2.5 Botão Limpar

O botão limpar serve para limpar todos os componentes visuais do formulário, como mostra a figura 98.

Administração Central
Cetec Capacitações

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

Método utilizado para colocar o cursor dentro do componente desejado.

Figura 98 – Código-fonte do botão Limpar