

# **J PHP**

**Versão 1.0**

**Biblioteca de classes para auxílio no desenvolvimento de aplicativos  
em PHP com banco de dados MySQL**

Autor:

**Jonatas Vieira Coutinho**  
jonnyleg@gmail.com

**Cataguases – MG, janeiro de 2007**

# Índice

<b>Introdução.....</b>	<b>03</b>
<b>1. Configurações iniciais .....</b>	<b>03</b>
1.1 Definido a classe aplicativo .....	03
1.2 Definindo o layout .....	04
1.3 Cabeçalhos das páginas .....	05
<b>2 Criando formulários .....</b>	<b>05</b>
2.1 Lista de componentes para formulários .....	07
<b>3. Conectando ao Banco de Dados MySQL .....</b>	<b>10</b>
3.1 Executando uma consulta .....	10
3.2 Outros métodos para exibição de resultados de consultas .....	12
3.3 Tabelas dinâmicas .....	14
3.3.1 Redefinindo as imagens .....	17
3.4 Tabela dinâmica com consulta por palavra-chave .....	17
3.4.1 Conversão de datas .....	20
3.4.2 Ordenando a tabela .....	21
3.4.3 Inserindo novas cláusulas .....	21
<b>4 Voltando ao formulário – list box com valores do Banco de Dados .....</b>	<b>22</b>
4.1 list box para Banco de Dados com valor padrão selecionado.....	23
<b>5 Aplicação – Inclusão, alteração e exclusão de cadastros .....</b>	<b>23</b>
5.1 Configurações iniciais .....	24
5.2 Página de pesquisa.....	24
5.3 Formulário para inclusão de registros .....	25
5.4 Formulário para alteração de registros .....	27
5.5 Formulário para exclusão de registros .....	30
<b>6 Upload de arquivos .....</b>	<b>31</b>
6.1 Upload de arquivos com timestamp .....	33

# Introdução - J PHP

Esta biblioteca de classes proporciona ao desenvolvedor facilidade na conexão com o banco de dados MySQL, realização de consultas, exibição dos resultados e criação de formulários Web. Ainda em fase de desenvolvimento, tem como objetivo tornar-se um pequeno framework para desenvolvimento em PHP. Apesar de ainda não ter todas as funcionalidades de um framework, já é uma idéia de como se pode iniciar o desenvolvimento de um modelo de aplicação. Este manual explica de maneira simples como utilizar esta biblioteca para agilizar o desenvolvimento de pequenos aplicativos em PHP, não abordando técnicas avançadas de programação.

Existem diversos frameworks e ferramentas disponíveis, livres ou não, que possuem excelentes estruturas para desenvolvimento de aplicações, como o Zend Framework e o PRADO. Também destacam-se novos modelos de programação como o AJAX, que proporcionam ganho de desempenho e novas funcionalidades com maior interatividade. O objetivo que levou ao desenvolvimento destas classes não é alcançar ganhos de desempenho ou grandes funcionalidades, o principal objetivo deste projeto é mostrar que é possível a um desenvolvedor criar seu modelo para agilizar o desenvolvimento de um escopo de aplicações.

## 1. Configurações iniciais

As classes estão divididas nos seguintes arquivos:

- *apl.php* – Classe principal que contém as informações para conexão ao banco de dados e a requisição para as demais classes (deve ser chamado no cabeçalho dos arquivos da aplicação que utilizar as classes);
- *componentesform.php* – Classes para componentes de formulários;
- *form.php* – Classe para geração de formulários HTML;
- *sqlmy.php* – Classe para conexão, comandos SQL e consultas no banco de dados MySQL;
- *formbd.php* – Classes para geração de scripts para integrar formulários com banco de dados (formulários de inserção, alteração e exclusão de registros);
- *localiza.php* – Classe que possibilita a criação de um formulário para consultas por palavra-chave no banco de dados e obtenção das respostas em uma tabela HTML;
- *recebearq.php* – Classe para realização de uploads de arquivos;
- *defcss.css* – Arquivo com as definições em CSS do layout dos formulários e tabela do aplicativo.

Para criar um programa que utilize estas classes, basta salvar a pasta “classes” (que contém os arquivos das classes) e “imagens” (que contém algumas imagens que serão utilizadas) na pasta do programa e fazer as referências aos arquivos principais (“classes/apl.php” e “classes/defcss.css”) nas páginas do aplicativo, conforme será descrito na seção 1.3.

### 1.1 Definindo a classe aplicativo

No arquivo *apl.php* existem algumas configurações gerais que devem ser definidas:

- As informações de host, login e senha do banco de dados;
- O caminho das imagens para os links de edição, exclusão e voltar para os formulários e tabelas dinâmicas geradas pelas classes (já vêm configurados com imagens padrão);
- Informação para o alinhamento de tabelas e formulários.

```
<?php
/*
J PHP - Versão 0.5
Autor: Jonas Vieira Coutinho
Última modificação: 10/01/2007

Este arquivo apl.php deve ser refetenciado no cabeçalho das páginas da aplicação
*/
// =====
//Verifica a versão do PHP. Se for menor que 5, aborta o aplicativo
if (0 > version_compare(PHP_VERSION, '5')) {
    die('Este aplicativo só funciona com PHP versão 5 ou superior');
}
//Classe principal do aplicativo
class aplicativo {
    protected
        //Aqui entram as informações para o banco de dados
        //=====
        /*
        Aqui entra o host do servidor MySQL
        Geralmente o servidor MySQL está na mesma máquina do servidor Apache/PHP, sendo o
        host definido então como localhost. Caso seja outra máquina, substitua o localhost
        pelo endereço IP do servidor MySQL
        */
        $hostname = "localhost",
        //Nome do banco de dados
        $database = "teste",
        //Usuário do banco de dados
        $userDb = "teste",
        //Senha do banco de dados
        $passwordDb = "33212a",
        //=====
        /*
        $con = mysql_connect($hostname,$userDb,$passwordDb);
        if (!$con) {
            die('Não foi possível estabelecer conexão com o banco de dados: ' . mysql_error());
        }
        $db = mysql_select_db($database,$con);
        if (!$db) {
            die('Não foi possível selecionar o banco de dados: ' . mysql_error());
        }
        */
    }
}
```

### 1.3. Cabeçalhos das páginas

Em todas as páginas do aplicativo que irá utilizar as classes, deverão ser referenciados os arquivos *apl.php* e *defcss.css* conforme a figura a seguir:

```
<?php //Referenciando o arquivo com a classe aplicativo e demais classes
require_once('classes/apl.php'); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- Referenciando o arquivo com as definições CSS -->
<link rel="StyleSheet" type="text/css" href="classes/defcss.css">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Teste com J PHP</title>
</head>

<body>
|
</body>
</html>
```

Página com as referências para as classes

## 2. Criando formulários

Para a criação de formulários simples, será criado um objeto da classe form:

A Classe form deve ser instanciada recebendo como parâmetros:

- Nome do formulário;
- Método (post ou get);
- Ação (página que será chamada ao submeter os dados);
- Texto e nome do botão (separados pelo caracter : ).
- Página para o link de “voltar” (caso seja necessário).

Ex: `$form1 = new form('nome','método','ação','Texto botão:nome botão','linkvoltar');`

O método *setForm* que monta o formulário, deve receber, em sintaxe pré-definida, a lista de seus componentes.

Ex. *Nome::campoTexto,,nome,,20*

Irá gerar um campo texto cujo nome no código HTML será “nome” de tamanho 20, com uma label “Nome” (parâmetro antes dos dois pontos) antes do campo.

Neste caso, foi utilizado o componente *campoTexto*. Um exemplo com mais componentes pode ser visto a seguir:

```

<?php //Referenciando o arquivo com a classe aplicativo e demais classes
require_once('classes/apl.php'); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- Referenciando o arquivo com as definições CSS -->
<link rel="StyleSheet" type="text/css" href="classes/defcss.css">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Teste com J PHP</title>
</head>

<body>
<?php
//Instancia o formulario
$form1 = new form('formulário1','post','','Enviar Dados corretos:enviar','');

//Monta e exibe o formulario incluindo os campos descritos
$form1->setForm('Nome::campoTexto,,nome,,30;;Endereco::campoTexto,,endereco,,50;;Data
Nasc::campoData,,datanasc,,15');

?>
</body>
</html>

```

### Exemplo de formulário

Neste exemplo, ao instanciar o formulário, não foram inseridas as informações de página de destino (ação) e link “voltar”, sendo seus espaços preenchidos com “”.

Nota-se também, que para separar um componente do outro utiliza-se “;”.

O resultado ao executar o script é este:

The screenshot shows a web browser window with the address bar displaying 'http://localhost/teste/formulario.php'. Below the address bar, there is a form with the following elements:

- A label 'Nome' followed by a text input field.
- A label 'Endereco' followed by a text input field.
- A label 'Data' followed by a text input field.
- A label 'Nasc' followed by a text input field.
- A button labeled 'Enviar Dados corretos'.

### Formulário gerado pela classe form

Agora será inserida a informação do link para voltar:

```

//Instancia o formulario
$form1 = new form('formulário1','post','','Enviar Dados corretos:enviar','main.php');

//Monta e exibe o formulario incluindo os campos descritos
$form1->setForm('Nome::campoTexto,,nome,,30;;Endereco::campoTexto,,endereco,,50;;Data
Nasc::campoData,,datanasc,,15');

?>

```

**Código com link para voltar definido no último parâmetro da instância do objeto**


O script executado ficará assim:

Endereço <http://localhost/teste/formulario.php>

Nome

Endereco

Data Nasc



Formulário já com o link para voltar

## 2.1. Lista de componentes para formulários

A lista completa dos tipos de campos disponíveis e suas respectivas sintaxes é a seguinte:

- campoTexto – cria um campo para digitação. Ex. *Nome::campoTexto,,nome,,20*

Nome

- campoTextoLongo – cria um campo para digitação de textos longos (*textarea* em HTML). Recebe depois do nome HTML, a quantidade de linhas e quantidade de colunas para o campo. Ex. *Histórico:: campoTextoLongo,,hist,,70,,10*

Histórico

- campoSenha – a mesma sintaxe de campoTexto, porém, ao digitar neste campo, os caracteres estarão mascarados (campo *password* HTML). Ex. *Senha::campoSenha,,passwd,,10*

Senha

- campoData – A mesma sintaxe de campoTexto, porém, ao digitar neste campo, uma máscara de data automaticamente será inserida ao digitar. Ex. *Data Cadastro::campoData,,data,,15*

Data Cadastro

- campoDataHoje – Da mesma forma que campoData, possui máscara de entrada, mas já iniciado com a data atual do servidor preenchida. Ex. *Data Inclusão::campoDataHoje,,datainc,,15*

Data Inclusão

- **campoOculto** – Na sintaxe deste campo, obviamente, não preenchemos a label antes dos ::. Ex. `::campoOculto,teste`. Nada impede que a label seja preenchida, mas isto não é usual em campos ocultos.
- **campoArq** – Campo para upload de arquivos. É preenchido apenas o nome do campo. Ex. `Carta::campoArq,,carta`

**Carta**

Todos estes campos (exceto o *campoDataHoje* e *campoArq*) ainda possuem um parâmetro opcional que é o valor padrão, que pode ser definido depois da informação do tamanho do campo. Ex.

`Cidade::campoTexto,,cidade,,30,,Cataguases`

**Cidade**

Há ainda o campo *select* que cria um *list box* com valores a serem selecionados.

Para os que conhecem HTML, sabe-se que o *list box* exibe labels que representam valores que serão postados. Exemplo. 1 para Cataguases e 2 para Ubá. O usuário verá Cataguases e Ubá como opções, mas os valores que serão postados serão 1 e 2.

A sintaxe deste campo é:

*Label do campo::select,,nomedocampo,,opções*

As opções têm a seguinte sintaxe:

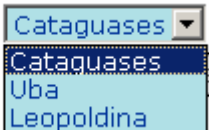
*valor#label/valor#label*

Os elementos separados por #, e em cada elemento, o valor e sua respectiva label separados por /

Ex.

`Cidade::select,,cidade,,1/Cataguases#2/Uba#3/Leopoldina`

Irá gerar o seguinte campo:


**Cidade** 

Onde Cataguases terá o valor 1, Ubá o valor 2 e Leopoldina o valor 3

Também é possível definir um dos valores para ser selecionado por padrão. No exemplo acima, definiremos o valor 2, que corresponde a Ubá, como valor a ser selecionado por padrão. Basta inserir, depois de duas vírgulas, o valor correspondente à opção padrão. Ex.

`Cidade::select,,cidade,,1/Cataguases#2/Uba#3/Leopoldina,,2`

A combo já iniciará com o valor “Ubá” selecionado:

**Cidade** 

Este recurso é interessante quando este valor padrão é definido pelo resultado de uma consulta no banco de dados.



## Formulário com todos os exemplos:

Um formulário com todos estes exemplos seria definido assim:

```
<body>
<?php
//Instancia o formulario
$form1 = new form('formul1', 'post', 'insere.php', 'Enviar Dados:enviar', 'main.php');

//Monta e exibe o formulario incluindo os campos descritos
$form1->setForm('Nome::campoTexto,,nome,,20;;Senha::campoSenha,,passwd,,10;;Data
Cadastro::campoData,,data,,15;;Inclusão::campoDataHoje,,datainc,,15;;;campoOculto,,teste;;Carta::camp
oArq,,carta;;Estado::campoTexto,,estado,,30,,Minas
Gerais;;Cidade::select,,cidade,,1/Cataguases#2/Uba#3/Leopoldina;;Histórico::campoTextoLongo,,hist,,70,
,10');

?>
</body>
```

### Formulário com todos os componentes

Resultado do código:

Nome

Senha

Data Cadastro

Inclusão

Carta

Estado

Cidade

Histórico

### Formulário completo gerado pela classe form

#### Importante –

Será abordado também um outro componente tipo *select* que busca os valores do banco de dados, mas antes disto, será explicada a classe responsável pela conexão ao banco e realização de consultas.

Os formulários que tiverem campos para upload de arquivos devem ser preparados para o upload através do método *defArq()*. Mais detalhes na seção **Upload de arquivos**.

### 3. Conectando ao banco de dados MySQL

A classe responsável pela conexão ao banco de dados é a classe *sqlMy*, que possui diversos comandos para execução de consultas, exibição de resultados e criação de tabelas dinâmicas com os resultados.

**Observação** – Alguns scripts dos exemplos deste manual estão disponíveis no “pacote” da biblioteca, bem como um arquivo .sql para a criação do Banco de dados para teste (Mais informações no arquivo LEIAME.txt)

#### 3.1. Executando uma consulta

Para executar uma consulta no banco de dados, é necessário, após instanciar o objeto *sqlMy*, utilizar o método *executaConsulta()*, onde *sql* é a consulta a ser executada no banco.

Ex.

```
<?php
//Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
$ pessoas = new sqlMy();
//Executa a consulta
$ pessoas->executaConsulta("select nome, endereco, telefone from pessoas order by nome");
```

Para uma exibição rápida do resultado, pode ser utilizado o método *exibeResultado*, que monta o resultado numa tabela HTML simples.

Ex.

```
<?php
//Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
$ pessoas = new sqlMy();
//Executa a consulta
$ pessoas->executaConsulta("select nome, endereco, telefone from pessoas order by nome");
//Exibe o resultado
$ pessoas->exibeResultado();

?>
```

Ao executar o script, será exibido o resultado da consulta:

---

nome	endereco	telefone
Fernando Ciclano Silva	Rua Testando outro endereço, 25	(32) 4125-8788
José Fulano de Souza	Rua Teste de Endereço, 80	(32) 8744-1122
Maria Beltrana Camargo	Alameda dos testes, 44	(32) 8744-1124
Otacílio Manoel de Souza	Testando Rua Mais um teste, 44	(32) 4125-8744

**Resultado de consulta gerado pelo método *exibeResultado* da classe *sqlMy***

A Classe *SqlMy* utiliza como usuário e senha padrão para conexão ao banco de dados, os valores definidos na classe aplicativo no arquivo *apl.php*. Ao instanciar um objeto da classe *sqlMy*, ele realiza uma conexão persistente no banco de dados com este usuário e senha. Se houver necessidade de conexão com outro usuário, utiliza-se o método *reconecta* para substituir a conexão padrão do banco pela redefinida.

Ex.

```

<?php
    //Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
    $pessoas = new sqlMy();
    //Executa a consulta
    $pessoas->executaConsulta("select nome, endereco, telefone from pessoas order by nome");
    echo "<p class='tit'>Consulta utilizando o usuário e senha padrão do programa:</p><br><br>";
    //Exibe o resultado
    $pessoas->exibeResultado();

    echo "<br><br>";

    //Redefinindo o usuário e senha para a consulta
    $pessoas->reconecta("user2","rr0463");
    echo "<p class='tit'>Consulta utilizando outro usuário e senha:</p><br><br>";
    //Executa a consulta
    $pessoas->executaConsulta("select nome, endereco, telefone from pessoas order by nome");
    //Exibe o resultado
    $pessoas->exibeResultado();

?>

```

#### Mudando de usuário e senha no banco de dados

Note que os títulos inseridos antes da consulta (echo "...") estão utilizando a classe CSS "tit" definida no arquivo *defcss.css*.

O resultado do script:

Consulta utilizando o usuário e senha padrão do programa:

nome	endereco	telefone
Fernando Ciclano Silva	Rua Testando outro endereço, 25	(32) 4125-8788
José Fulano de Souza	Rua Teste de Endereço, 80	(32) 8744-1122
Maria Beltrana Camargo	Alameda dos testes, 44	(32) 8744-1124
Otacílio Manoel de Souza	Testando Rua Mais um teste, 44	(32) 4125-8744

Consulta utilizando outro usuário e senha:

nome	endereco	telefone
Fernando Ciclano Silva	Rua Testando outro endereço, 25	(32) 4125-8788
José Fulano de Souza	Rua Teste de Endereço, 80	(32) 8744-1122
Maria Beltrana Camargo	Alameda dos testes, 44	(32) 8744-1124
Otacílio Manoel de Souza	Testando Rua Mais um teste, 44	(32) 4125-8744

#### A mesma consulta realizada por 2 usuários do banco de dados

Este recurso de troca de usuário é necessário quando o controle de usuários e senhas do aplicativo também é feito por restrições no banco de dados. O desenvolvedor pode definir um usuário sem privilégios de alteração ou exclusão no banco como padrão na classe aplicativo e realizar a troca pelos usuários referentes às pessoas logadas no sistema com privilégios maiores para operações que envolvam alterações nos dados do banco.

## 3.2. Outros métodos para exibição de resultados de consultas

Além do método *exibeResultado*, que mostra o resultado em uma tabela HTML, é possível exibir este resultado de outras formas com os seguintes métodos:

- *resetaLinha()* – Vai para a primeira linha do resultado da consulta;
- *mostraItem('item')* – Onde *item* é o nome da coluna do resultado da consulta a ser exibida na linha atual;
- *avancaLinha()* – Vai para a próxima linha do resultado da consulta;

Ex.

```
<?php
//Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
$ pessoas = new sqlMy();
//Executa a consulta
$ pessoas->executaConsulta("select nome, endereco, telefone from pessoas order by nome");
echo "<p class='tit'>Pessoas cadastradas no Banco de Dados</p><br><br>";
//Inicia a exibição do resultado
$ pessoas->resetaLinha();
//Varre o resultado, exibindo-o de acordo com a formatação desejada pelo programador
do {
    echo "<p class='texto'><b>Nome: </b>". $ pessoas->mostraItem('nome'). "<br>";
    echo "<b>Endereco: </b>". $ pessoas->mostraItem('endereco'). "<br>";
    echo "<b>Telefone: </b>". $ pessoas->mostraItem('telefone'). "<br></p>";
} while ($ pessoas->avancaLinha());
```

?>

Resultado de consulta utilizando *resetaLinha()*, *mostraItem()* e *avancaLinha()*

Execução do código:

---

### Pessoas cadastradas no Banco de Dados

Nome: Fernando Ciclano Silva  
Endereco: Rua Testando outro endereço, 25  
Telefone: (32) 4125-8788

Nome: José Fulano de Souza  
Endereco: Rua Teste de Endereço, 80  
Telefone: (32) 8744-1122

Nome: Maria Beltrana Camargo  
Endereco: Alameda dos testes, 44  
Telefone: (32) 8744-1124

Nome: Otacílio Manoel de Souza Testando  
Endereco: Rua Mais um teste, 44  
Telefone: (32) 4125-8744

Resultado da consulta acima

## Mais métodos para resultados

- *mostraResultadoLinha(linha, campo)* – Exibe o resultado na linha especificada em *linha* e na coluna especificada em *campo*.

Ex.

```
echo "<p class='texto'><b>Primeira pessoa: </b>". $pessoas->mostraResultadoLinha(0, 'nome')." <br>";
```

Será exibida a primeira linha do resultado da consulta na coluna “nome”:

**Primeira pessoa:** Fernando Ciclano Silva

- *totalLinhas()* – Retorna o número de linhas do resultado da consulta.

*mostraResultadoLinha()* e *totalLinhas()* são uma outra opção para exibição do resultado de consultas no banco de dados.

Ex. A mesma consulta acima utilizando estes métodos:

```
<?php
//Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
$pessoas = new sqlMy();
//Executa a consulta
$pessoas->executaConsulta("select nome, endereco, telefone from pessoas order by nome");
echo "<p class='tit'>Pessoas cadastradas no Banco de Dados</p><br><br>";
//Exibindo o resultado com mostraResultadoLinha e totalLinhas
for ($i = 0; $i < $pessoas->totalLinhas(); $i++) {
    echo "<p class='texto'><b>Nome: </b>". $pessoas->mostraResultadoLinha($i, 'nome')." <br>";
    echo "<b>Endereço: </b>". $pessoas->mostraResultadoLinha($i, 'endereco')." <br>";
    echo "<b>Telefone: </b>". $pessoas->mostraResultadoLinha($i, 'telefone')." <br></p>";
}
```

?>

Consulta utilizando *mostraResultadoLinha()* e *totalLinhas()*

Execução do código:

### Pessoas cadastradas no Banco de Dados

**Nome:** Fernando Ciclano Silva  
**Endereço:** Rua Testando outro endereço, 25  
**Telefone:** (32) 4125-8788

**Nome:** José Fulano de Souza  
**Endereço:** Rua Teste de Endereço, 80  
**Telefone:** (32) 8744-1122

**Nome:** Maria Beltrana Camargo  
**Endereço:** Alameda dos testes, 44  
**Telefone:** (32) 8744-1124

**Nome:** Otacilio Manoel de Souza Testando  
**Endereço:** Rua Mais um teste, 44  
**Telefone:** (32) 4125-8744

#### Resultado da consulta

O resultado é o mesmo da consulta utilizando *resetaLinha*, *mostraItem* e *avancaLinha*

### 3.3. Tabelas dinâmicas

Recurso muito utilizado para resultados de consultas que possuem links que redirecionam para outras páginas como edição e exclusão de registros levando um identificador do registro selecionado via url. A classe *sqlMy* possui o método *tabDin* que cria tabelas dinâmicas a partir de resultados de consultas.

Sua sintaxe é:

*\$objeto->tabDin('índices', 'cabecalhos', 'label link1, label link2', 'pagina link1', 'pagina link 2', 'índice da chave', 'variável url', 'mensagem consulta vazia');*

Onde:

- *Índices:* São os números referentes às colunas da consulta que serão exibidas na tabela separados por vírgula em uma string;
- *Cabecalhos:* São os títulos das colunas separados por vírgula na mesma ordem dos índices;
- *Label link 1, Label link 2:* Cabecalhos da colunas do primeiro e segundo link (se não forem preenchidos, assumem os textos “Editar” e “Excluir”);
- *Página link 1:* Caminho para a página de destino do link 1;
- *Página link 2:* Caminho para a página de destino do link 2;
- *Índice da chave:* Índice numérico da coluna da consulta que representa a chave primária que vai ser repassada via url como identificador do registro (a consulta, obviamente, deve selecioná-la);
- *Variável url:* Nome da variável que será o identificador do registro cujo valor vem do índice da chave;
- *Mensagem consulta vazia:* Mensagem que será exibida no lugar da tabela quando a consulta não retornar linhas.

Ex.

Uma tabela dinâmica a partir da mesma consulta que realizamos acima:

```

<?php
    //Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
    $pessoas = new sqlMy();
    //Executa a consulta
    $pessoas->executaConsulta("select idPessoa, nome, endereco, telefone from pessoas order by nome");
    echo "<p class='tit'>Consulta utilizando tabela dinâmica</p><br><br>";
    //Cria a tabela dinâmica
    $pessoas->tabDin('1,3','Nome,Telefone','','edita.php','exclui.php',0,'pessoa','Não existem pessoas cadastradas');
?>

```

#### Gerando uma tabela dinâmica

**Observação** – Os valores para os cabeçalhos dos links 1 e 2 não foram preenchidos (3º parâmetro), dando lugar aos valores padrão.

Resultado:

#### Consulta utilizando tabela dinâmica









Nome	Telefone	Editar	Excluir
Fernando Ciclano Silva	(32) 4125-8788		
José Fulano de Souza	(32) 8744-1122		
Maria Beltrana Camargo	(32) 8744-1124		
Otacílio Manoel de Souza Testando	(32) 4125-8744		

Tabela dinâmica gerada pelo método tabDin

As imagens de edição e exclusão já estão no “pacote” das classes e seus caminhos definidos na classe aplicativo em *apl.php*.

Se for colocado o cursor no link para edição do terceiro registro, por exemplo, a página de destino no rodapé do navegador será a página que foi definida no método com a variável de url indicando o valor da linha correspondente da consulta.

`1/teste/edita.php?pessoa=3`

Agora, serão definidos os cabeçalhos que por padrão vieram como “Editar” e “Excluir” pelo não preenchimento dos valores no método. Definindo seus valores, os mesmos serão exibidos no lugar dos cabeçalhos padrão:

```

<?php
    //Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
    $pessoas = new sqlMy();
    //Executa a consulta
    $pessoas->executaConsulta("select idPessoa, nome, endereco, telefone from pessoas order by nome");
    echo "<p class='tit'>Consulta utilizando tabela dinâmica</p><br><br>";
    //Cria a tabela dinâmica
    $pessoas->tabDin('1,3','Nome,Telefone','Alterar,Deletar','edita.php','exclui.php',0,'pessoa','Não existem pessoas cadastradas');
?>

```

#### Redefinindo os cabeçalhos para os links

**Observação importante:** Este parâmetro dos cabeçalhos dos links, bem como os dois parâmetros anteriores (índices e cabeçalhos), é uma string única separada por vírgula e não dois parâmetros separados.

Resultado:

### Consulta utilizando tabela dinâmica









Nome	Telefone	Alterar	Deletar
Fernando Ciclano Silva	(32) 4125-8788		
José Fulano de Souza	(32) 8744-1122		
Maria Beltrana Camargo	(32) 8744-1124		
Otacílio Manoel de Souza Testando	(32) 4125-8744		

Tabela com os textos para os links redefinidos

Agora será feita a mesma tabela dinâmica, mas sem a segunda coluna. Para isto, basta apenas suprimir a informação da página para o link 2 (5º parâmetro), e conseqüentemente, seu cabeçalho no 3º parâmetro:

```
<?php
//Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
$ pessoas = new sqlMy();
//Executa a consulta
$ pessoas->executaConsulta("select idPessoa, nome, endereco, telefone from pessoas order by nome");
echo "<p class='tit'>Consulta utilizando tabela dinâmica</p><br><br>";
//Cria a tabela dinâmica
$ pessoas->tabDin('1,3','Nome,Telefone','Alterar','edita.php','','0','pessoa','Não existem pessoas cadastradas');
?>
```

Suprimindo a segunda coluna

Resultado:

### Consulta utilizando tabela dinâmica

Nome	Telefone	Alterar
Fernando Ciclano Silva	(32) 4125-8788	
José Fulano de Souza	(32) 8744-1122	
Maria Beltrana Camargo	(32) 8744-1124	
Otacílio Manoel de Souza Testando	(32) 4125-8744	

Tabela dinâmica com apenas uma coluna

**Observação** – O parâmetro para a primeira página de link (4º parâmetro) também pode ser suprimido, fazendo com que a tabela não possua links.



### 3.3.1 Redefinindo as imagens

Se o programador quiser redefinir as imagens para a tabela dinâmica, deverá usar o método *redefineImg1('caminho')* ou *redefineImg2('caminho')*, onde *caminho* é o caminho da imagem que será exibida no link.

Ex:

```
<?php
//Instancia o objeto sqlMy com as informações padrão do banco de dados na classe aplicativo
$pessoas = new sqlMy();
//Executa a consulta
$pessoas->executaConsulta("select idPessoa, nome, endereco, telefone from pessoas order by nome");
echo "<p class='tit'>Consulta utilizando tabela dinâmica</p><br><br>";
//Redefine a imagem para o link 1
$pessoas->redefineImg1('imagens/pdf.gif');
//Cria a tabela dinâmica
$pessoas->tabDin('1,3','Nome,Telefone','Emitir Relatório','rel.php','',0,'pessoa','Não existem
pessoas cadastradas');
?>
```

Resultado:

#### Consulta utilizando tabela dinâmica

Nome	Telefone	Emitir Relatório
Fernando Ciclano Silva	(32) 4125-8788	
José Fulano de Souza	(32) 8744-1122	
Maria Beltrana Camargo	(32) 8744-1124	
Otacílio Manoel de Souza Testando	(32) 4125-8744	

Imagem redefinida

**Observação** – As cores da tabela dinâmica podem ser alteradas no arquivo *defcss.css*

### 3.4. Tabela dinâmica com consulta por palavra-chave

É a união das classes *form* e *sqlMy* em um componente também muito utilizado onde o usuário digita um valor em um formulário de pesquisa e o resultado da pesquisa é retornado em uma tabela dinâmica. Este componente é criado pela classe *localiza*.

Sintaxe:

No momento em que o objeto é instanciado, já são passados alguns parâmetros:

```
$objeto = new localiza('consulta','campo localizado','label do formulário');
```

Onde:

- *Consulta*: A consulta SQL na tabela (apenas a seleção dos campos. Ex. “Select \* from pessoas”);
- *Campo localizado*: O campo da tabela que será pesquisado de acordo com o texto digitado;
- *Label do formulário*: A label do formulário de pesquisa.

Ex.

```
<?php
    //Instancia o objeto localiza passando a consulta, o campo a ser pesquisado e a label do form
    $localizador = new localiza('select idPessoa, nome, endereco, telefone from pessoas','nome',
    'Digite aqui');
```

Após instanciar o objeto passando os parâmetros iniciais, resta montar o localizador e a tabela dinâmica através do método *montaLoc* que tem exatamente na mesma sintaxe do método *tabDin* da classe *sqlMy*.

Ex.

```
<body>
<?php
    //Instancia o objeto localiza passando a consulta, o campo a ser pesquisado e a label do form
    $localizador = new localiza('select idPessoa, nome, endereco, telefone from pessoas','nome',
    'Digite aqui');
    echo "<p class='tit'>Consulta utilizando tabela dinâmica</p><br><br>";
    //Cria a tabela dinâmica
    $localizador->montaLoc('1,3','Nome,Telefone','Alterar,Deletar','edita.php','exclui.php',0,'pessoa',
    'Pessoa não encontrada');
    ?>
```

Resultado:

### Consulta utilizando tabela dinâmica

Digite aqui

### Consulta dinâmica utilizando o a classe localiza

Pesquisando um nome:

## Consulta utilizando tabela dinâmica

Digite aqui

Ao digitar um nome no campo do formulário e pressionar o botão, será exibida uma tabela dinâmica com o resultado da pesquisa, caso o nome seja localizado:

## Consulta utilizando tabela dinâmica

Digite aqui

Nome	Telefone	Alterar	Deletar
José Fulano de Souza	(32) 8744-1122		

**Resultado da busca pelo nome “José”**

Se o nome não fosse localizado, seria mostrada a mensagem definida no último parâmetro do método *montaLoc*:

## Consulta utilizando tabela dinâmica

Digite aqui

## Pessoa não encontrada

**Resultado em caso de nome não encontrado**

**Observação** – Se o botão “Localizar” for acionado sem um texto digitado no campo do formulário, a tabela dinâmica exibirá todos os registros.

### 3.4.1 Conversão de datas

O formato de data padrão do MySQL é aaaa-mm-dd, sendo assim, ao ser realizada uma consulta que busque uma data no banco de dados, o resultado deve ser convertido para o formato convencional (dd/mm/aaaa). Para isto, utiliza-se o método *setConvIndice(colunas,tipos)*, onde *colunas* são os índices das colunas dos campos tipo data, e tipos são os tipos de dados contidos neles que podem ser *datahora* (para datetime ou data (para date)).

Ex.

Uma tabela dinâmica que retornou os seguintes resultados:

Data Expiração	Excluir
2007-12-31	✗
2007-02-10	✗
2007-03-17	✗

Tabela dinâmica com data

Para converter a data para o formato dd/mm/aaaa, basta definir, antes da montagem da tabela dinâmica, a coluna do resultado que contém a data e o tipo (datahora ou data).

```
<div id="area_adm">
    <?php
    //Instancia o localizador
    $localizador = new localiza('select * from noticias','titulo','Digite aqui');
    //Define que as colunas 1 e 6 do resultado sofrerão conversões de datas de mysql para convencional
    $localizador->setConvIndice("6","data");
    //Montagem do localizador
    $localizador->montaLoc('2,6','Titulo,Data Expiração',' ',' ','noti_exc.php',0,'noticia','Não há
    notícias cadastradas');
    ?>
</div>
```

Definindo o campo data que será convertido de aaaa-mm-dd para dd/mm/aaaa

Resultado:

Data Expiração	Excluir
31/12/2007	✗
10/02/2007	✗
17/03/2007	✗
31/12/2007	✗

Campo data convertido para dd/mm/aaaa

Se a tabela possuir mais de uma coluna com este tipo de dado, basta utilizar o método definindo todas as colunas (separadas por vírgula) e todos os tipos na mesma ordem (também separados por vírgula). Ex.

```
//Define que as colunas 1 e 6 do resultado sofrerão conversões de datas de mysql para convencional
$localizador->setConvIndice("1,6","datahora,data");
```

Neste exemplo, as colunas 1 e 6 do resultado da consulta possuem, respectivamente, os tipos de dado “datetime” e “date”.

O tipo datahora que se refere a “datetime”, faz a conversão da data, mantendo a hora na string.

Este método pode também ser utilizado na classe *sqlMy*, com o método *tabDin*.

### 3.4.2. Ordenando a tabela

Para ordenar a tabela por uma de suas colunas, usa-se o método *setIndiceOrdem(coluna,ordem)*, onde *coluna* é o nome do campo da tabela que será ordenado, e *ordem*, pode assumir dois valores: *desc* para decrescente ou *asc* para ascendente.

Ex.

```
//Define que a consulta será ordenada pela data da postagem em ordem decrescente
$localizador->setIndiceOrdem("data","desc");
```

### 3.4.3. Inserindo novas cláusulas

Se houver a necessidade de mais cláusulas do tipo where na consulta SQL do localizador, as mesmas podem ser inseridas com o método *setClausulaEsp(clausula)*. Ex.

```
$localizador->setClausulaEsp("excluida = 0");
```

**Observação** – Este métodos, (setConvIndice, setIndiceOrdem e setClausulaEsp, quando forem utilizados, devem ser inseridos antes do método *montaLoc*)

## 4. Voltando ao formulário – list box com valores do banco de dados

Na classe *form*, existe o componente que cria um *list box* para seleção de valores pré-definidos pelo usuário. Também existe um componente similar que busca estes valores de uma tabela no banco de dados, é o *selectBd*.

Sintaxe:

*Label::selectBd,,nome HTML,,tabela,,campo Valor,,Campo Label,,Texto padrão,,ordem*

Onde:

- *Tabela*: Nome da tabela que contém os dados para o *list box*;
- *Campo Valor*: Coluna da tabela que contém os valores;
- *Campo Label*: Coluna da tabela que contém as labels;
- *Texto Padrão*: Texto que ficará no campo antes de ser selecionado;
- *Ordem*: Ordem da lista, que pode ser *asc* (para ordem crescente) ou *desc* (para ordem decrescente).

Ex:

*Responsável::selectBd,,responsavel,,pessoas,,idPessoa,,nome,,Selecione uma pessoa,,asc*

Exemplo de código de um formulário utilizando este campo:

```
<body>
<?php
//Instancia o formulario
$form1 = new form('formulário1','post','insere.php','Enviar Dados:enviar','main.php');

//Monta e exibe o formulario incluindo os campos descritos
$form1->setForm('Nome::campoTexto,,nome,,20;;Senha::campoSenha,,passwd,,10;;Data
Cadastro::campoData,,data,,15;;Inclusão::campoDataHoje,,datainc,,15;;Responsável::selectBd,,responsave
l,,pessoas,,idPessoa,,nome,,Selecione uma pessoa,,asc');

?>
```

Código para formulário com selectBd

Resultado:

Nome	<input type="text"/>
Senha	<input type="password"/>
Data Cadastro	<input type="text"/>
Inclusão	<input type="text" value="22/01/2007"/>
Responsável	<div><div>Selecione uma pessoa</div><div><div>Selecione uma pessoa</div><div>Fernando Cidano Silva</div><div>José Fulano de Souza</div><div>Manoelina Carvalho</div><div>Maria Beltrana Camargo</div><div>Otacílio Manoel de Souza Testando</div></div></div>

Formulário com *list box* buscando valores no banco de dados

Há ainda um último parâmetro opcional neste método que é o parâmetro *data*, que se for definido (com o valor 1, por exemplo), faz com que as labels sofram conversão de datas do formato do MySQL (aaaa-mm-dd) para o formato brasileiro (dd/mm/aaaa).

Ex. *Responsável::selectBd,,responsavel,,pessoas,,idPessoa,,nome,,Selecione uma pessoa,,asc,,1*

#### 4.1 list box para banco de dados com valor padrão selecionado

Para criar um *list box* que já seja inicializado com um valor padrão selecionado deve-se usar o componente *selectBdVal*, que tem o parâmetro de texto padrão trocado por um valor padrão que será o valor da label que virá selecionada no *list box*.

Ex:

*Responsável::selectBd,,responsavel,,pessoas,,idPessoa,,nome,,3,,asc*

Neste exemplo, o registro cujo campo *IdPessoa* for igual a 3, será selecionado por padrão:



Nome

Senha

Data Cadastro

Inclusão

Responsável



Formulário com *list box* com valor padrão selecionado

Este recurso é interessante quando há a necessidade, em um formulário de alteração de registros, exibir um nome que na verdade apenas possui sua chave gravada na tabela, não podendo ser exibido diretamente com o resultado da consulta que alimentou os valores dos campos do formulário. No parâmetro do valor padrão, pode ser inserida uma variável com o valor que está no resultado da consulta do registro atual que irá fazer com que o *selectBdVal* busque o nome referente a esta chave na tabela de origem para ser selecionado como padrão.

## 5. Aplicação – Inclusão, alteração e exclusão de cadastros

Como exemplo, foi desenvolvido um aplicativo para gestão dos dados de um pequeno banco de dados (o mesmo dos exemplos anteriores). Este pequeno banco possui uma tabela chamada “pessoas”:

```
mysql> desc pessoas;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| idPessoa   | int(11)       |      | PRI | NULL    | auto_increment |
| nome       | varchar(200)  |      |     |         |                |
| endereco   | varchar(255)  |      |     |         |                |
| telefone   | varchar(15)   | YES  |     | NULL    |                |
| cidade     | varchar(100)  |      |     |         |                |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

Descrição da tabela pessoas

**Observação** – Como o objetivo deste banco é apenas exemplificar o funcionamento dos formulários, sua modelagem não foi detalhada (campos separados como rua, bairro, número, etc).

## 5.1. Configurações iniciais

No arquivo *apl.php* devem ser inseridas as informações para conexão padrão ao banco de dados (host, bd, usuário e senha).

## 5.2 Página de pesquisa

No exemplo, a página principal é a página de pesquisa das pessoas cadastradas, chamada *localiza\_pessoas.php*, onde teremos um link para a página de inserção de dados e também um “localizador” da classe *localiza* que irá gerar uma tabela dinâmica com links para as páginas de edição (*form\_altera\_pessoas.php*) e exclusão (*form\_deleta\_pessoas.php*).

O código da página será este:

```
<?php //Referenciando o arquivo com a classe aplicativo e demais classes
require_once('classes/apl.php'); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- Referenciando o arquivo com as definições CSS -->
<link rel="StyleSheet" type="text/css" href="classes/defcss.css">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Teste com J PHP</title>
</head>

<body>
<p class="tit">Cadastro de Pessoas</p><br><br>
<p class="texto"><a href="form_inserir_pessoas.php">Inserir Nova</a></p>
<?php
    //Instancia o objeto localiza passando a consulta, o campo a ser pesquisado e a label do form
    $localizador = new localiza('select idPessoa, nome, endereco, telefone from pessoas','nome',
    'Pesquisar');
    //Cria a tabela dinamica
    $localizador->montaLoc('1,3','Nome,Telefone','Alterar,Deletar','form_altera_pessoas.php',
    'form_deleta_pessoas.php',0,'pessoa','Pessoa não encontrada');
?>
</body>
</html>
```

Código da página localiza\_pessoas.php



Resultado:

## Cadastro de Pessoas

[Inserir Nova](#)

Pesquisar

Localizar

Página `localiza_pessoas.php`

### 5.3 Formulário para inclusão de registros

A página para inclusão de registros (*form\_inserir\_pessoas.php*) é esta:

```
<?php //Referenciando o arquivo com a classe aplicativo e demais classes
require_once('classes/apl.php'); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- Referenciando o arquivo com as definições CSS -->
<link rel="StyleSheet" type="text/css" href="classes/defcss.css">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Teste com J PHP</title>
</head>

<body>
<p class="tit">Inserir pessoas</p>
<?php
//Instancia o formulario
$form1 = new form('formulari01','post','insere_pessoas.php','Enviar Dados:enviar',
'localiza_pessoas.php');

//Monta e exibe o formulario incluindo os campos descritos
$form1->setForm(
'Nome::campoTexto,,nome,,20;;Endereço::campoTexto,,endereço,,50;;Telefone::campoTexto,,telefone,,15;;C
idade::campoTexto,,cidade,,15');

?>
</body>
</html>
```

Código da página `form_inserir_pessoas.php`

Note que o parâmetro para configuração da ação do formulário (3º parâmetro na instância do objeto) está redirecionando os dados da página para uma página chamada *insere\_pessoas.php* que conterá os comandos para inserção dos registros no banco de dados da classe *formBd*.

Nos scripts que são chamados pelos formulários, a classe *formBd* é instanciada sem passar parâmetros.

O método para execução dos comando SQL do formulário é o método *executa(sql, página destino, campo verificado)*. Onde:

- *Sql*: É a consulta SQL para inserir os registros com base nos dados digitados pelo usuário;
- *Página Destino*: É o caminho da página que será chamada ao executar o script;
- *Campo Verificado*: É o campo do formulário cuja existência o script verificará antes de executar os comandos. Se o campo não existir, o comando não é executado.

Entre a instância do objeto *formBd* e a execução do comando, é necessário definir a SQL que irá ser executada pelo objeto com base nos registros postados pelo formulário de inserção.

A cada componente de formulário foi atribuído um nome. Ex. *Endereço::campoTexto,,endereco,,20*. Neste campo, o nome foi definido depois da definição do tipo do campo, então seu nome será “endereco”, o que significa que o conteúdo deste campo será passado do formulário de inserção para o script de inserção como *\$\_POST['endereco']* (Um índice do array *\$\_POST* no índice “endereco”), sendo assim também para os demais campos do formulário.

A SQL no script de inserção será definida da seguinte forma:

```
$sql = sprintf("insert into pessoas values (NULL, %s, %s, %s, %s)",
               $insere->getSql($_POST['nome'], "text"),
               $insere->getSql($_POST['endereco'], "text"),
               $insere->getSql($_POST['telefone'], "text"),
               $insere->getSql($_POST['cidade'], "text")
            );
```

Os valores vindos do formulário não foram inseridos diretamente no comando SQL, antes foram tratados pelo método *getSql* que acompanha as classes derivadas da classe *aplicativo*. Sua sintaxe é:

*\$objeto->getSql(variável, tipo)*

Onde:

- *Variável*: É a variável (geralmente *\$\_POST*) que será tratada;
- *Tipo*: É o tipo de dado que se espera da variável. Podendo ser “text” para texto, “int” para números inteiros, “double” para valores ponto-flutuante e “date” para datas (Já com a função de converter datas no formato dd/mm/aaaa para datas no formato do MySQL, aaaa-mm-dd).

Este método é importante para evitar invasões bem-sucedidas com *SQL Injection*.

O código da *insere\_pessoas.php* é este:

```
<?php //Referenciando o arquivo com a classe aplicativo e demais classes
require_once('classes/apl.php');

//Instancia o objeto da classe formBd
$insere = new formBd();

//Monta a sql utilizando o método getSql para tratar as strings
$sql = sprintf("insert into pessoas values (NULL, %s, %s, %s, %s)",
               $insere->getSql($_POST['nome'], "text"),
               $insere->getSql($_POST['endereco'], "text"),
               $insere->getSql($_POST['telefone'], "text"),
               $insere->getSql($_POST['cidade'], "text")
            );

//Insere os valores
$insere->executa($sql, "form_insere_pessoas.php", "nome");
?>
```

**Observação importante:** Esta página (*insere\_pessoas.php*) e demais scripts para inserção, alteração e exclusão de dados vindos de formulários não deve conter tags HTML, deve conter apenas os códigos PHP para inserção do registro. O código mostrado acima é o código completo da página *insere\_pessoas.php*.

Se houver a necessidade de mais comandos SQL para o mesmo script, basta definí-los e separá-los pela string “::” no parâmetro do método `executa`.

Ex:

```
$insere->executa($sql2."::".$sql3,"form_insere_pessoas.php","nome");
```

## Inserindo Registros

Com as páginas do formulário e do script de inserção definidas, já é possível inserir registros no banco de dados.

## 5.4. Formulário para alteração de registros

O formulário para alteração de registros, assim como o de exclusão, é mais complexo. Antes do código que define os campos, deve ser executada uma consulta no banco de dados para obtenção dos registros a alterar, então criando o formulário para alteração dos registros com os valores do registro atual já preenchidos como padrão nos campos.

Este formulário de alteração será chamado pelo link de edição da tabela dinâmica da página de pesquisa do programa:

Nome	Telefone	Alterar	Deletar
José Fulano de Souza	(32) 8744-1122		

Tabela dinâmica com os links para alterar e excluir o registro

Ao clicar no botão de alterar, a página para alteração do registro será chamada com um valor via url do registro a ser alterado. Estes valores via url são convertidos em variáveis `$_GET`. Na tabela dinâmica deste programa, o nome da variável de url que definimos para levar o identificador do registro chama-se “pessoa”. Então, no formulário de alteração, o parâmetro para a busca do campo a ser selecionado será a variável `$_GET['pessoa']`.

Antes da consulta, pode-se verificar se esta variável está definida, fazendo com que o script encerre no caso de não existir:

```
<body>
<p class="tit">Alterar pessoa</p>
<?php
//Verifica se uma pessoa foi selecionada antes deste formulário ser chamado
if (!isset($_GET['pessoa'])) die('sem referência');
```

Verificando se a variável de url existe

Logo após, a consulta é executada:

```
//Instancia a consulta para exibir os valores padrão
$ pessoas = new sqlMy();

//Monta a sql com a referência ao código da pessoa selecionada
$sql = sprintf("select * from pessoas where idPessoa = %s",
               $pessoas->getSql($_GET['pessoa'], "int")
            );
//Executa a consulta
$pessoas->executaConsulta($sql);
```

Executando a consulta

**Observação:** Nota-se que o valor vindo via url foi tratado pelo método *getSql* antes de ser incorporado à consulta, para evitar problemas com *SQL Injection*.

Outro tratamento de exceção que pode ser feito é verificar se a consulta não retornou um registro, caso não tenha retornado (no caso do valor via url ser passado manualmente e não corresponder a um registro existente), o código não deve continuar:

```
//Caso o código passado via url não corresponda a uma pessoa no banco de dados o sistema aborta
if ($pessoas->totalLinhas() == 0) die('sem referência');
```

Verificando se a consulta não retornou um registro

Agora, resta instanciar o formulário e definir os valores padrão dos campos com o resultado da consulta realizada acima:

```
//Instancia o formulário
$form1 = new form('formulário1', 'post', 'altera_pessoas.php', 'Alterar dados:alterar',
                 'localiza_pessoas.php');

/*
Monta e exibe o formulario incluindo os campos descritos com o resultado da consulta
como valores padrão
*/
$form1->setForm("Nome::campoTexto,,nome,,20,,", $pessoas->mostraItem('nome').
";Endereço::campoTexto,,endereço,,50,,", $pessoas->mostraItem('endereço').
";Telefone::campoTexto,,telefone,,15,,", $pessoas->mostraItem('telefone').
";Cidade::campoTexto,,cidade,,15,,", $pessoas->mostraItem('cidade').";::campoOculto,,idPessoa,,",
$pessoas->mostraItem('idPessoa'));

?>
```

Código do formulário

O código da página:

```
<body>
<p class="tit">Alterar pessoa</p>
<?php
//Verifica se uma pessoa foi selecionada antes deste formulário ser chamado
if (!isset($_GET['pessoa'])) die('sem referência');

//Instancia a consulta para exibir os valores padrão
$pessoas = new sqlMy();

//Monta a sql com a referência ao código da pessoa selecionada
$sql = sprintf("select * from pessoas where idPessoa = %s",
               $pessoas->getSql($_GET['pessoa'], "int")
            );
//Executa a consulta
$pessoas->executaConsulta($sql);

//Caso o código passado via url não corresponda a uma pessoa no banco de dados o sistema aborta
if ($pessoas->totalLinhas() == 0) die('sem referência');

//Instancia o formulário
$form1 = new form('formulariol', 'post', 'altera_pessoas.php', 'Alterar dados:alterar',
                 'localiza_pessoas.php');

/*
Monta e exibe o formulario incluindo os campos descritos com o resultado da consulta
como valores padrão
*/
$form1->setForm("Nome::campoTexto,,nome,,20,,", $pessoas->mostraItem('nome').
";Endereço::campoTexto,,endereço,,50,,", $pessoas->mostraItem('endereço').
";Telefone::campoTexto,,telefone,,15,,", $pessoas->mostraItem('telefone').
";Cidade::campoTexto,,cidade,,15,,", $pessoas->mostraItem('cidade').";::campoOculto,,idPessoa,,",
$pessoas->mostraItem('idPessoa'));
?>
```

#### Código PHP completo da página form\_altera\_pessoas.php

**Observação importante:** Foi criado um campo oculto no formulário que guardará o campo “idPessoa” que é o identificador do registro que será passado como parâmetro na consulta SQL que atualizará este registro. Isto é de extrema importância para que o script funcione.

O formulário enviará os dados alterados para a página *altera\_pessoas.php* que utilizará a classe *formBd* para a execução do comando SQL para alterar o registro:

```
<?php //Referenciando o arquivo com a classe aplicativo e demais classes
require_once('classes/apl.php');

$altera = new formBd();

$sql = sprintf("update pessoas set nome =%s, endereço =%s, telefone=%s, cidade=%s where idPessoa=%s",
               $altera->getSql($_POST['nome'], "text"),
               $altera->getSql($_POST['endereço'], "text"),
               $altera->getSql($_POST['telefone'], "text"),
               $altera->getSql($_POST['cidade'], "text"),
               $altera->getSql($_POST['idPessoa'], "int")
            );

//Código que deve ser passado via url novamente para o formulário de alteração
$codigoPessoa = $_POST['idPessoa'];

//Executa a alteração
$altera->executa($sql, "form_altera_pessoas.php?pessoa=$codigoPessoa", "idPessoa");
?>
```

#### Código da página altera\_pessoas.php

**Observação:** Neste exemplo, a página *altera\_pessoas.php*, ao executar o comando SQL, chamará novamente a página *form\_altera\_pessoas.php*, e para que o registro seja exibido novamente no formulário, deve-se enviar novamente via url o identificador da pessoa que foi alterada (neste caso, a variável `$_POST['idPessoa']`), conforme feito no código acima. Caso a página a ser chamada seja outra, não é necessário fazer isto.

## Alterando Registros

Com as páginas do formulário e do script de alteração definidas, já é possível alterar registros no banco de dados, selecionando-os na tabela dinâmica da página *localiza\_pessoas.php* e alterando os dados no formulário que será aberto.

## 5.5. Formulário para exclusão de registros

O formulário para exclusão de registros (*form\_deleta\_pessoas.php*) pode ser criado com o mesmo código do formulário para alteração (*form\_altera\_pessoas.php*). Neste exemplo, alteramos o tipo de todos campos do formulário para *label* (exceto o campo oculto que guardará o identificador do registro), pois não é necessária a digitação nestes campos, fazendo com que o formulário funcione como uma página para confirmação de exclusão de registros.

Código da página:

```
<body>
<p class="tit">Remover pessoa</p>
<?php
//Verifica se uma pessoa foi selecionada antes deste formulário ser chamado
if (!isset($_GET['pessoa'])) die('sem referência');

//Instancia a consulta para exibir os valores padrão
$pessoas = new sqlMy();

//Monta a sql com a referência ao código da pessoa selecionada
$sql = sprintf("select * from pessoas where idPessoa = %s",
               $pessoas->getSql($_GET['pessoa'], "int")
            );
//Executa a consulta
$pessoas->executaConsulta($sql);

//Caso o código passado via url não corresponda a uma pessoa no banco de dados o sistema aborta
if ($pessoas->totalLinha() == 0) die('sem referência');

//Instancia o formulário
$form1 = new form('formul1', 'post', 'deleta_pessoas.php', 'Excluir dados:excluir',
                 'localiza_pessoas.php');

/*
Monta e exibe o formulario incluindo os campos descritos com o resultado da consulta
como valores padrão dos campos
*/
$form1->setForm("Nome::label,,", $pessoas->mostraItem('nome').";Endereço::label,,", $pessoas->
mostraItem('endereco').";Telefone::label,,", $pessoas->mostraItem('telefone').";Cidade::label,,",
$pessoas->mostraItem('cidade').";::campoOculto,,idPessoa,,", $pessoas->mostraItem('idPessoa'));

?>
```

Código da página *form\_deleta\_pessoas.php*

Ao submeter o formulário, será chamada a página *deleta\_pessoa.php* que fará a deleção do registro:

```
<?php //Referenciando o arquivo com a classe aplicativo e demais classes
require_once('classes/apl.php');

$deleta = new formBd();

$sql = sprintf("delete from pessoas where idPessoa =%s",
               $deleta->getSql($_POST['idPessoa'], "int")
               );

//Executa a alteração
$deleta->executa($sql, "localiza_pessoas.php", "idPessoa");
?>
```

Código da página *deleta\_pessoas.php*

### Deletando Registros

Com as páginas do formulário e do script de deleção definidas, já é possível deletar registros no banco de dados, selecionando-os na tabela dinâmica da página *localiza\_pessoas.php* e confirmando a exclusão no formulário.

#### Observação:

Nos scripts para inserção, alteração e exclusão de registros, as páginas de destino podem ser definidas para qualquer página. No exemplo acima, os scripts de inserção e alteração voltaram para o formulário e o de exclusão foi para a página inicial, mas poderiam ser criadas páginas de confirmação de operação. Ex. “O registro foi alterado com sucesso”.

## 6. Upload de arquivos

A classe responsável pelo upload de arquivos é a classe *recebeArq*.

Sintaxe:

*\$objeto = new recebeArq(max tam, pasta).*

Onde:

- *Max tam*: Tamanho máximo permitido para o arquivo;
- *Pasta*: Caminho da pasta onde os arquivos postados serão salvos.

O método *recebeArquivo* é o responsável pelo upload.

Sintaxe:

*\$objeto->recebeArquivo(arquivo).*

Onde *arquivo* é a variável que contém o arquivo postado, vinda de um campo para upload de arquivos em um formulário. Este tipo de variável é do tipo \$\_FILES.

Ex. \$\_FILES['teste'], é uma variável que contém um arquivo gerado por um campo tipo arquivo de nome "teste".

Para o upload, é necessário um formulário com um campo para upload de arquivos definido. Neste exemplo, serão criadas duas páginas: A página com o formulário para postagem do arquivo (*exemplo\_formulario\_upload.php*) e a página com o script que fará o upload (*exemplo\_formulario\_upload.php*).

Na página com o formulário, depois de instanciar o formulário que fará o upload, é necessário defini-lo para trabalhar com arquivos, para que seu código HTML que será gerado contenha um parâmetro que o prepare para isto. Para defini-lo para trabalhar com arquivos, utiliza-se o método *defArq()*, que não possui parâmetros.

Código do formulário:

```
<body>
<?php

//Instancia o formulário
$formarq = new form('formulario1','post','exemplo_upload.php','Postar:btnpost','');

//Define o formulário para trabalhar com arquivos
$formarq->defArq();

//Monta o formulário
$formarq->setForm("Arquivo::campoArq,,teste");
?>

</body>
```

Código da página exemplo\_formulario\_upload.php

Código da página de upload:

```
<body>
<?php
//Tamanho máximo, em bytes
$maxtam = 2000000;
//Pasta destino
$pasta = "/var/www/jphp/teste/arquivos/";

//Instancia o objeto
$arquivo = new recebeArq($maxtam, $pasta);

//Recebe o arquivo
if ($arquivo->recebeArquivo($_FILES['teste'])) {
    //Se o arquivo foi postado com sucesso exibe a mensagem abaixo
    echo '<p class="mensconf">Arquivo postado com sucesso</p>';
    //Caso contrário, exibe o erro correspondente
} else echo $arquivo->exibeErro();

?>
</body>
```

Código da página exemplo\_upload.php



### Observações:

- Se o caminho para a postagem for composto de barras invertidas ( \ ), no caso de servidor Windows, será necessário escrevê-las duas vezes. Ex: C:\Arquivo de Programas\Apache\www\Arquivos. Também pode-se utilizar as barras convencionais ( / ) para este caminho, como no exemplo com servidor Linux, não sendo necessário duplicá-las.
- O método *recebeArquivo* retorna verdadeiro caso o upload tenha sucesso. No exemplo acima, foi utilizado um *if* que exibe uma mensagem de confirmação caso o upload tenha sucesso, e a mensagem de erro correspondente (utilizando o método *exibeErro*) caso o upload não tenha sido efetuado com sucesso.
- No caso de servidor Linux, é necessário dar permissões de escrita na pasta destino do upload.

## 6.1. Upload de arquivos com timestamp

Para evitar que dois arquivos com o mesmo nome sejam postados na pasta destino, pode-se utilizar o método *setTime()*, que faz com que os arquivos postados tenham o timestamp atual adicionado à seus nomes, garantindo que cada arquivo postado terá um nome particular.

```
<body>
<?php
//Tamanho máximo, em bytes
$maxtam = 2000000;
//Pasta destino
$pasta = "/var/www/jphp/teste/arquivos/";

//Instancia o objeto
$arquivo = new recebeArq($maxtam, $pasta);

//Define o upload para adicionar o timestamp atual no nome dos arquivos postados
$arquivo->setTime();

//Recebe o arquivo
if ($arquivo->recebeArquivo($_FILES['teste'])) {
    //Se o arquivo foi postado com sucesso exibe a mensagem abaixo
    echo '<p class="mensconf">Arquivo postado com sucesso</p>';
    //Caso contrário, exibe o erro correspondente
} else echo $arquivo->exibeErro();

?>
```

A página exemplo\_upload.php com a utilização do setTime