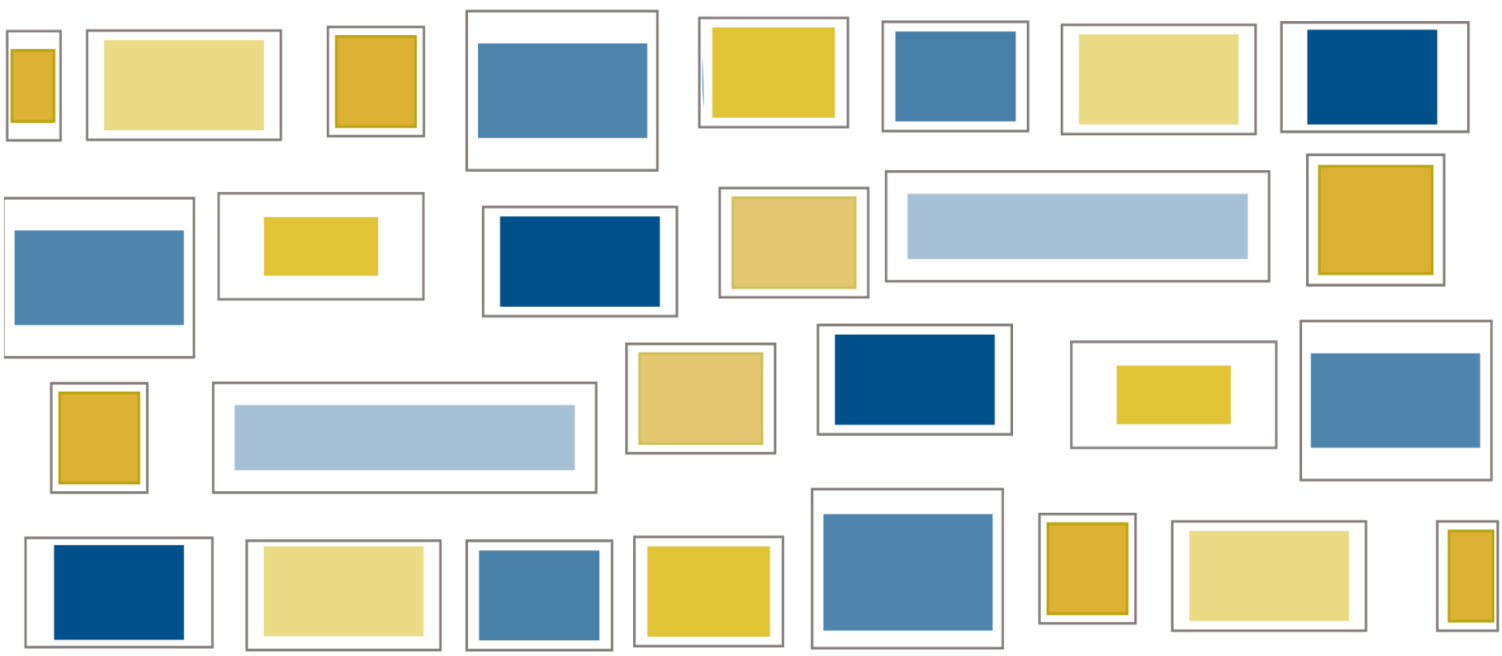
Programação Web I



Universidade Estadual de Goiás

GOIÁS ESTADO INOVADOR

MEDIOtec

BRASIL GOVERNO FEDERAL

GOVERNO FEDERAL

MICHEL TEMER

Presidente

MINISTÉRIO DA EDUCAÇÃO

JOSÉ MENDONÇA BEZERRA FILHO

Ministro

GOVERNO DO ESTADO DE GOIÁS

MARCONI FERREIRA PERILLO JÚNIOR

Governador do Estado de Goiás

UNIVERSIDADE ESTADUALDE GOIÁS

HAROLDO REIMER

Reitor

JULIANA OLIVEIRA ALMADA

Chefe de Gabinete

MARIA OLINDA BARRETO

Pró-Reitor de Graduação

IVANO ALESSANDRO DEVILLA

Pró-Reitor de Pesquisa e Pós-Graduação

MARCOS ANTÔNIO CUNHA TORRES

Pró-Reitor de Extensão, Cultura e Assuntos Estudantis

LACERDA FERREIRA MARTINS

Pró-Reitor de Gestão e Finanças

CHRISTIANO DE OLIVEIRA E SILVA

Pró-Reitor de Planejamento e Desenvolvimento Institucional

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.

Atenção: indica pontos de maior relevância no texto.

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.

Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Apresentação da disciplina

Nesta disciplina estudaremos programação para *web* sob o enfoque do servidor, ou seja, depois de enviado algum dado a partir de uma página em HTML. Esta disciplina é uma continuação de **Fundamentos do Desenvolvimento *Web*** (FDW).

Várias são as linguagens para criação de sistemas *web*. Vamos nos focar na linguagem PHP, por ser uma linguagem de fácil aprendizado, comparada com as demais, e bastante popular.

Como é uma linguagem de programação, todos os conceitos aprendidos até agora serão bastante utilizados. Também trabalharemos com banco de dados, armazenando e manipulando as informações via páginas *web*. Nesse momento, será muito útil o conhecimento adquirido da disciplina **Banco de Dados** (BD).

Por fim, veremos uma arquitetura para sistemas *web* como forma de organizarmos melhor as páginas de um sistema, com a finalidade de obtermos produtividade no desenvolvimento e facilitarmos a manutenção posterior.

Lembre-se, a melhor forma de aprender programação é praticando!

Aula 1 – A linguagem PHP

Objetivos

Comparar as vantagens e desvantagens do PHP em relação a outras linguagens.

Instalar e configurar o ambiente de desenvolvimento *web*.

Conhecer a sintaxe da linguagem PHP.

Construir páginas *web* com PHP.

1.1 O que é PHP?

PHP é uma linguagem que permite criar *sites web* dinâmicos, fundamentada nos dados submetidos pelo usuário e derivada dos dados contidos no banco de dados, que são alterados frequentemente. Vamos pegar o exemplo de uma loja virtual. Os produtos estão sempre sofrendo alterações, seja no preço, na quantidade em estoque, nos produtos em promoções, nos lançamentos, etc. Hoje, quando você entra em uma loja virtual, verá alguns produtos em promoção, outros em lançamentos, com um determinado preço. Na próxima semana que você visitar o *site*, pode ser que os preços estejam mais baixos, por causa de novas promoções, ou aquele produto que você tinha visto não se encontre mais em estoque, já tenha sido vendido.

Passaremos a programar para *web* sob a visão do servidor. Para isso, utilizaremos a linguagem PHP. Como pré-requisito, é necessário o conhecimento de HTML, principalmente de formulário, que será utilizado para enviar dados para o servidor. Portanto é extremamente importante que você revise o conteúdo visto em **Fundamentos do Desenvolvimento *Web***.

O código PHP é executado no servidor, sendo enviado para o cliente apenas HTML. Dessa maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente.

Pesquise sobre as outras linguagens de programação *web* e discuta com os colegas e tutores, no ambiente virtual de ensino-aprendizagem, destacando as vantagens e desvantagens de cada uma.

O PHP foi criado em 1995 por Rasmus Lerdorf com o nome de *Personal Home Page Tools* (Ferramentas Para Página Pessoal), para auxiliar no desenvolvimento de páginas simples. Como teve boa aceitação e muitos programadores utilizando-as, novas versões foram desenvolvidas com cada vez mais recursos.

Existem outras linguagens de programação que podemos utilizar para criar as páginas dinâmicas, como Java, Perl, ASP, etc.

1.2 Instalação

Para testar as páginas PHP, não basta dar um duplo clique nos "arquivos. php", como se faz com os .htm ou .html. É necessário ter um **servidor *web*** configurado para isso. Um dos servidores *web* mais utilizados é o **Apache**.

servidor *web*
É um programa de computador responsável por aceitar pedidos HTTP de clientes, geralmente os navegadores, e servi-los com páginas de respostas (em HTML) incluindo dados, imagens, sons e *links*. Esse programa disponibiliza um local específico no computador servidor (*site*) para armazenar e processar as páginas de resposta.

Apache
É o resultado de um esforço coletivo de vários colaboradores, para o desenvolvimento de um *software* gratuito, robusto e com qualidade, para a implementação de um servidor HTTP. É o servidor *web* mais usado no mundo. É capaz de executar códigos em PHP, Perl, Shell Script, ASP, dentre outros. Sua utilização mais conhecida é a que combina o Apache com a linguagem PHP e o banco de dados MySQL.

1.2.1 Instalação no Windows

Você pode instalar o Apache e o PHP separados. Para isso basta pegar os arquivos de instalação nos respectivos *sites* oficiais.

Porém, configurações manuais deverão ser feitas para os dois funcionarem perfeitamente.

A forma mais fácil de instalar é utilizar pacotes que instalam e configuram todos os programas necessários para o desenvolvimento de páginas *web* de uma única vez. Um conjunto muito utilizado consiste do **Apache** (servidor *web*), **MySQL** (banco de dados) e **PHP** (linguagem para as páginas *web* dinâmicas), conhecido como **AMP** (inicial de cada produto). Quando esses produtos são instalados no Linux, chamamos de LAMP. Quando são instalados no Windows, chamamos de WAMP.

Para mais detalhes sobre o Apache, acesse o *site* oficial: http://www.apache.org

Os principais instaladores são: WAMP5 (http://www.wampserver.com) e EasyPHP (http://www.easyphp.org).

1.2.2 Instalação no Linux (Ubuntu)

No Ubuntu também podemos instalar os programas separados, usando algum instalador, como o **apt-get**. Na mesma linha de comando é possível instalar todos os pacotes necessários, basta digitar:

sudo apt-get install apache2 php5 libapache2-mod-php5 mysql-server libapache2-mod-auth-mysql php5-mysql

Nas versões mais novas do Ubuntu, também é possível instalar um pacote com todos os programas. O comando é:

sudo tasksel install lamp-server

Faça o *download* do Apache e instale o *software* em seu computador: http://www.apache.org
Após instalar o Apache, faça o *download* do PHP e instale-o: http://www.php.net/downloads

Depois da instalação será necessário reinicializar o Apache. Para isso digite: sudo /etc/init.d/apache2 restart

1.2.3 Testando o ambiente

Em qualquer instalação, seja no Windows ou no Linux, um diretório específico será criado para colocar as páginas em PHP, chamado **www**. Quando o Apache recebe uma solicitação para exibir uma página, ele irá buscar nesse diretório.

Para mais detalhes sobre instalação no Linux: https://help.ubuntu.com/community/ApacheMySQLPHP

No **Windows**, o diretório **www** fica dentro do diretório de instalação do produto. Por exemplo, se você usou o **EasyPHP**, o diretório é:

C:\Arquivos de programas\EasyPHP-5.3.2\www

O diretório "EasyPHP" pode mudar de nome de acordo com a versão instalada.

No **Linux**, o diretório é:

/var/www

Podem ser criados subdiretórios dentro do diretório **www**. É até recomendado que se faça isso, para organizar melhor as páginas.

Para testar o ambiente, primeiro devemos verificar se os programas estão em execução. Você pode configurá-los para iniciar automaticamente, quando o computador for ligado, ou manualmente. Tanto o WAMP5 quanto o EasyPHP colocam um ícone próximo ao relógio que contém as opções para inicializar (*start*) ou parar (*stop*) os processos. No WAMP5, o ícone é parecido com um velocímetro de um carro. No EasyPHP, o ícone é o desenho de letra 'E'. Na Figura 1.1 vemos o exemplo do WAMP5 e na Figura 1.2 vemos o exemplo do EasyPHP.

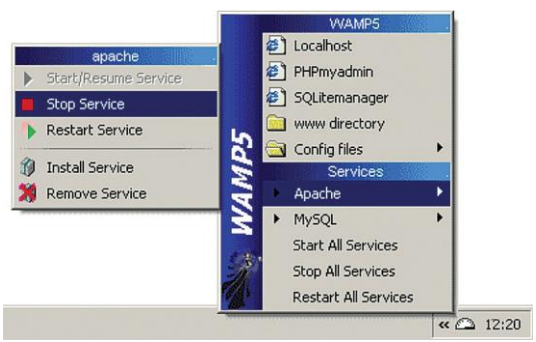


Figura 1.1: Exemplo do menu de opções do WAMP5
Fonte: WAMP5 (2010) instalado no Microsoft Windows XP

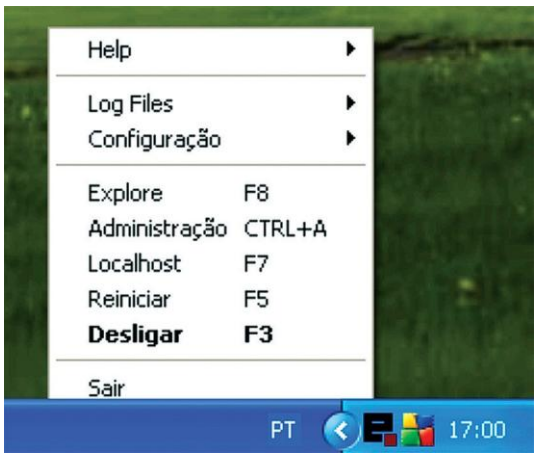


Figura 1.2: Exemplo do menu de opções do EasyPHP
Fonte: EasyPHP 5.3.2 (2010) instalado no Microsoft Windows XP

Agora vamos criar um arquivo com extensão .php (teste.php, por exemplo) na pasta base www. Abra-o com qualquer editor de texto e digite:

```
<?php phpinfo(); ?>
```

Salve-o e em seguida digite em seu navegador favorito o seguinte endereço: **http://localhost/teste.php**.

teste.php é o nome do arquivo PHP que você criou. **localhost** significa que o seu navegador irá procurar o arquivo no seu próprio computador, no diretório **www** configurado na instalação. Na internet, como o servidor está em outro local, substitui-se *localhost* pelo endereço *web* da empresa, como por exemplo: **http://www.empresaxyz.com.br/cadastro.php**

Se a instalação estiver correta, uma tela com informações sobre a configuração do PHP deverá ser exibida, como indicado pela Figura 1.3 a seguir.

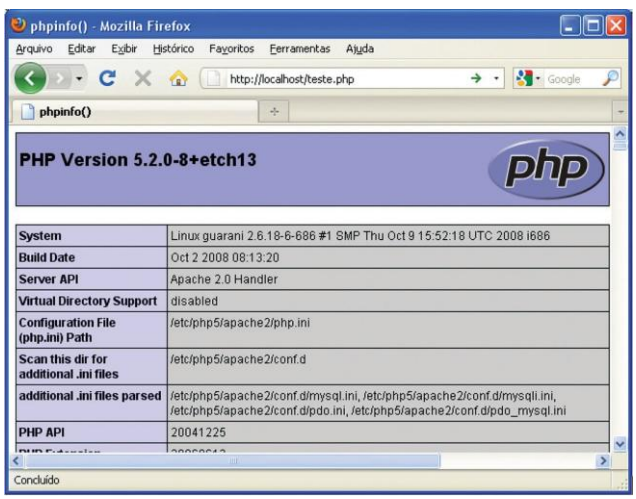


Figura 1.3: Instalação do PHP com sucesso
Fonte: EasyPHP 5.3.2 instalado no Microsoft Windows XP, exibido pelo Mozilla Firefox 3.5

1.3 Sintaxe básica

O código do PHP é embutido dentro de um arquivo HTML, quando for necessário algum processamento pelo servidor. Depois que o servidor processar o código PHP, apenas o que for gerado em HTML será enviado de volta para o usuário; assim, o usuário não conseguirá ver o código em PHP, que ficará apenas no servidor. A Figura 1.4 ilustra o funcionamento de uma página *web* dinâmica.

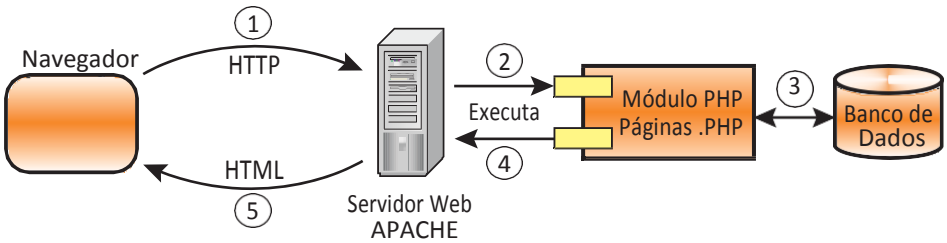


Figura 1.4: Funcionamento de uma página dinâmica PHP
Fonte: Equipe produção CEAD/IFES (2011)

Passo 1 – O usuário em seu navegador solicita uma página em PHP, por exemplo, *consulta.php*. Essa solicitação é enviada pelo protocolo HTTP ao servidor *web* da empresa (por exemplo, o Apache).

Passo 2 – O Apache chama a página PHP que foi solicitada e a executa.

Passo 3 – A página PHP pode ou não fazer acesso ao banco de dados.

Passo 4 – Ao final da execução do programa PHP, uma página de resposta em HTML é enviada ao Apache.

Passo 5 – O Servidor *web* Apache repassa a página de resposta para o navegador que a solicitou, que a exibe.

Para diferenciar o código PHP dentro da página em HTML, podem ser utilizados os delimitadores descritos na Figura 1.5 a seguir.

1) <?php Comandos; **?>**	**2) <?** Comandos; **?>**
3) <script language="php"> Comandos; </script>	**4) <%** Commandos; **%>**

Figura 1.5: Delimitadores de código em PHP
Fonte: Elaborada pelo autor

O primeiro delimitador da Figura 1.5 é o padrão. O segundo é uma simplificação do primeiro. Esses são os mais usados. O terceiro segue o estilo de *scripts* em HTML. O quarto segue o estilo do ASP.

Para utilizar a forma simplificada, bem como o estilo ASP, o arquivo de configuração *php.ini* deve ser alterado, pelos campos **short_open_tags** e **asp_tags**, respectivamente.

A Figura 1.6 mostra a estrutura de uma página HTML com o código PHP embutido.

```
<html>
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=UTF-8">
        <title>Página em PHP</title>
    </head>
    <body>
        <?php
            // Coloque seu código aqui
        ?>
    </body>
</html>
```

Figura 1.6: Exemplo de estrutura de uma programação de uma página em PHP
Fonte: Elaborada pelo autor

Pode haver vários blocos de PHP misturados com vários trechos de HTML.

1.3.1 Gerando código em HTML

Como o código PHP é processado no servidor e apenas o HTML é enviado como resposta, utilizaremos o **comando echo** para gerar esse HTML. Veja o exemplo da Figura 1.7 a seguir.

comando echo
É a instrução que envia para a página de saída, em HTML, as informações processadas em PHP, podendo ser texto, números ou variáveis. Essas informações, na maioria das vezes, são mescladas com os comandos HTML.

```
<html>
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=UTF-8">
        <title>Exemplo do echo</title>
    </head>
    <body>
        <?php
            echo "<H1>Página de teste do ECHO</H1>";
            echo "<P>Parágrafo em HTML</P>";
            echo "<FONT color=green>";
            echo "Mensagem solta em verde. <BR>";
            echo "</FONT>";
            echo "FIM.";
        ?>
    </body>
</html>
```

Figura 1.7: Exemplo de programa utilizando o comando echo
Fonte: Elaborada pelo autor

1.4 Como testar uma página em PHP

Existem vários editores de PHP. Dentre os gratuitos, recomenda-se o **PHPEditor**. Porém, vamos utilizar um ambiente de desenvolvimento mais moderno, que engloba várias linguagens e que está sempre em atualização, o **NetBeans**.

Para mais detalhes sobre o NetBeans e para fazer o *download* da sua instalação, acesse: http://netbeans.org

Para verificar se o seu NetBeans possui o módulo PHP instalado, basta acessar "**Ferramentas->Plug-ins**" e verificar, na aba **"Instalado"**, se o PHP se encontra ativado; senão, ative-o.

Se não estiver na aba "Instalado", procure-o na aba **"Plug-ins disponíveis"** e instale-o (é preciso estar conectado à internet).

Todos os arquivos devem ser salvos com a extensão **.php** no diretório **www** da instalação do servidor *web*. Crie subdiretórios dentro do **www** para melhor organizar seus arquivos.

Os subdiretórios dentro do diretório **www** devem ter permissão de gravação e escrita.

O primeiro passo é abrirmos o NetBeans e criarmos um novo projeto. Para isso, realize os seguintes passos:

1. Clique em **"Arquivo" > "Novo Projeto"**.

2. Dentre as opções **"Categorias"** da janela aberta, escolha "**PHP**". Dentro de **"Projetos"**, escolha "**Aplicativo PHP**". Clique em "**Próximo**".

3. Escolha um nome para o projeto (por exemplo, **MeuSitePHP**).

4. Na **"Pasta de códigos-fonte"**, escolha o diretório **www**, ou algum subdiretório dentro deste, por exemplo, **C:/Arquivos de Programas/EasyPHP/www/ProgWeb**. No Linux seria **/var/www/ProgWeb**. Clique em "**Finalizar**".

Após criarmos o projeto, ele passa a ser exibido na aba "**Projetos**", que fica no canto esquerdo do NetBeans. Para criar os arquivos em PHP, siga os seguintes passos:

1. Na aba de **“Projetos”**, clique com o botão direito do *mouse* no nome de nosso projeto.

2. Escolha as opções **“Novo” > “Página da *Web* do PHP”**.

3. Escolha um nome para a página com a extensão **.php** (por exemplo, **PrimeiraPagina.php**). Clique em “**Finalizar**”.

PRONTO!!! Você já pode programar em PHP. A Figura 1.8 apresenta um trecho da tela mostrada após a realização dos passos acima.

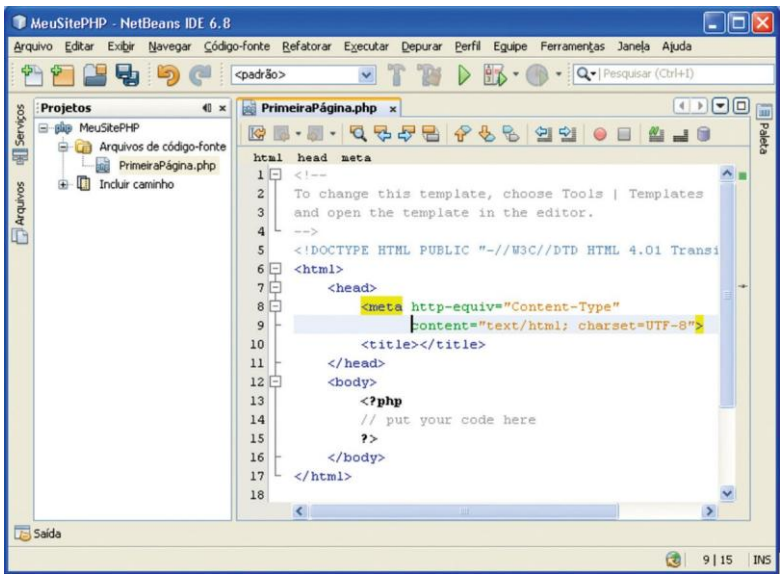


Figura 1.8: Criando uma página PHP no NetBeans
Fonte: NetBeans IDE 6.8 instalado no Microsoft Windows XP

Você também pode, e deve, criar subdiretórios dentro do seu projeto, para organizar melhor as suas páginas.

Uma arquitetura para desenvolvimento *web* que organiza a programação das páginas de um sistema, com o objetivo de aumentar a produtividade e facilitar a manutenção, será vista na Aula 6.

Para criar subdiretórios no NetBeans, siga os passos:

1. Clique com o botão direito do *mouse* no nome do projeto.

2. Escolha as opções **“Novo” > “Diretório”**.

3. Escolha um nome para o diretório. Clique em **“Finalizar”**.

Quando for criar novas páginas PHP, clique em cima do diretório no qual deseja colocar as páginas.

Depois de programar e salvar o arquivo, vamos testá-lo. Para isso, abra seu navegador favorito e digite: **http://localhost/nomedoarquivo.php**, em que:

localhost – corresponde ao seu computador local; e

nomedoarquivo.php – nome que você deu no seu arquivo PHP (no nosso exemplo, **PrimeiraPagina.php**).

Se o servidor *web* estiver em outro computador, então troque *localhost* pelo IP (ou *hostname*) desse servidor.

Caso tenha criado subdiretórios, então acrescente no endereço do navegador os subdiretórios criados. Exemplo: **http://localhost/ProgWeb/cadcliente.php**

Nesse exemplo foi criado o subdiretório **ProgWeb** e dentro dele foi colocada a página **cadcliente.php**.

1.5 Variáveis e tipos

As variáveis em PHP não precisam ser declaradas. Quando atribuímos algum valor para elas, o tipo é automaticamente reconhecido. Os tipos suportados são:

- Inteiros (*integer* ou *long*);
- Reais (*float* ou *double*);
- *Strings*;
- *Array* (vetores);
- Objetos*.

* Como se trata de um curso básico, a programação orientada a objetos em PHP não será vista.

Os nomes das variáveis devem ser criados com um '$' seguido de uma *string* que deve ser inicializada por uma letra ou '_'. Exemplos:

```
$x = 10.4;
$frase = "Exemplo de variável string";
$_cont = 0;
```

O PHP é *case sensitive*, ou seja, letras maiúsculas são diferentes de minúsculas. Portanto, para facilitar, criem as variáveis sempre em minúsculo.

Os comentários podem ser de uma linha, utilizando o símbolo **//**, ou de mais linhas, delimitado pelos símbolos **/*** e ***/**. Exemplos:

```
$cont = 0; // Exemplo de comentário de uma linha
/* Exemplo de comentário com mais de uma linha. Preste atenção nos
símbolos delimitadores. */
```

1.6 Operadores

Os principais operadores em PHP estão descritos nos quadros a seguir: os **operadores aritméticos** podem ser vistos no Quadro 1.4, os **operadores lógicos** estão descritos no Quadro 1.5, os **operadores de comparação** são mostrados no Quadro 1.6 e demais operadores importantes estão no Quadro 1.7 a seguir.

Quadro 1.4: Operadores aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão)

Fonte: Elaborado pelo autor

Quadro 1.5: Operadores lógicos

And	E
Or	Ou
Xor	Ou exclusivo
!	Negação
&&	E
||	Ou

Fonte: Elaborado pelo autor

Quadro 1.6: Operadores de comparação

==	Igual a
!=	Diferente de
<	Menor que
>	Maior que
<=	Menor ou igual a
>=	Maior ou igual a

Fonte: Elaborado pelo autor

Quadro 1.7: Outros operadores importantes

.	Concatenação de *strings*
=	Atribuição
++	Incremento (soma 1)
--	Decremento (subtrai 1)

Fonte: Elaborado pelo autor

1.7 Estruturas de controle

As estruturas de controle servem para controlar a ordem de execução das instruções de um programa. As principais são as de seleção e repetição.

1.7.1 Comandos de seleção

Os comandos de seleção servem para escolher um determinado bloco de comandos a partir da avaliação de uma expressão. Os comandos de seleção são: ***if*** (e suas variações) e o ***switch***.

Um bloco de comandos é delimitado pelos símbolos: '{' e '}'.

if – executa um bloco de comando caso a expressão seja verdadeira (se.. então). Veja um exemplo na Figura 1.9.

```
<?php
      $x = 10;
      if ( $x > 0 ) {
         echo "O valor é POSITIVO";
      }
?>
```

Figura 1.9: Exemplo do comando *if*
Fonte: Elaborada pelo autor

if...else – executa o primeiro bloco de comandos se a expressão for verdadeira e o bloco do ***else*** caso a expressão seja falsa. (se...então...senão). A Figura 1.10 mostra um exemplo.

```php
<?php
      $x = -20.4;
      if ( $x > 0 ) {
         echo "O valor é POSITIVO";
      }
      else {
         echo "O valor é NEGATIVO";
      }
?>
```

Figura 1.10: Exemplo do comando *if...else*
Fonte: Elaborada pelo autor

if...elseif...else – o ***elseif*** é utilizado quando várias condições precisam ser analisadas. Para cada ***elseif***, uma nova expressão deve ser analisada. Quando todas as expressões forem falsas, então o último bloco ***else*** será executado. Exemplo na Figura 1.11 a seguir.

```php
<?php
      $a = 5;  $b = 7;  $c = 3;
      if ( $a > $b ) {
         echo "a é MAIOR que b";
      }
      elseif ( $a > $c ) {
         echo "a é MAIOR que c";
      }
      else {
         echo "a é MENOR que b e c";
      }
?>
```

Figura 1.11: Exemplo do comando *if... elseif... else*
Fonte: Elaborada pelo autor

switch – funciona semelhante a vários *if* juntos. Uma expressão ou variável é analisada e, de acordo com o valor, um entre vários blocos de comandos é executado. Diferentemente do *if*, cuja expressão somente retorna verdadeiro ou falso, no *switch* o valor retornado pode ser diverso. A expressão é comparada com cada uma das cláusulas ***case*** até que uma coincida. Quando isso acontece, o bloco de comandos correspondente é executado até encontrar o comando ***break***, que interrompe a execução daquele bloco e finaliza o *switch*. Se nenhuma cláusula coincidir, então o bloco delimitado pelo comando ***default*** é executado. Na Figura 1.12 temos um exemplo do *switch*.

```php
<?php
    $n1 = 10;  $n2 = 5;
    $op = "*";   // Altere a opção para testar
    switch ( $op ) {
      case "+" : echo "Soma = " .($n1 + $n2);
                 break;
      case "-" : echo "Subtração = " .($n1 - $n2);
                 break;
      case "*" : echo "Multiplica = " .($n1 * $n2);
                 break;
      case "/" : echo "Divisão = " . ($n1 / $n2);
                 break;
      default : echo "Operação inválida! ";
    }
?>
```

Figura 1.12: Exemplo do comando *switch*
Fonte: Elaborada pelo autor

1.7.2 Comandos de repetição

Os comandos de repetição servem para executar repetidas vezes o mesmo bloco de comandos, até que uma condição de parada seja atingida. Os comandos são: ***while***,***do...while*** e ***for***. A diferença entre eles está na condição de parada das repetições e no contador de iteração.

while – a condição de parada é testada no início da iteração. Se for verdadeira, repete o bloco de comandos; se for falsa, interrompe as repetições. Exemplo na Figura 1.13.

```
<?php
      $i = 0;  $f = 99;
      while ( $i < $f ) {
         $i++;
         $f--;
         echo "I = $i    F = $f <BR>";
      }
?>
```

Figura 1.13: Exemplo do *while*
Fonte: Elaborada pelo autor

do...while – funciona de maneira semelhante ao *while*; a diferença é que a condição é testada depois do bloco de comandos. Isso garante que pelo menos uma vez o bloco de comandos será executado. Veja um exemplo na Figura 1.14 a seguir.

```
<?php
      $i = 0;  $f = 99;
      do {
         $i++;
         $f--;
         echo "I = $i       F = $f <BR>";
      } while ( $i < $f );
?>
```

Figura 1.14: Exemplo do *do...while*
Fonte: Elaborada pelo autor

for – é utilizado quando se conhece a quantidade total de iterações ou quando se pretende contar essas iterações. Sua sintaxe é:

for (inicialização; condição; incremento) { . . }

Em que:

- **Inicialização** – é uma instrução de atribuição executada apenas uma vez, no início do laço. Geralmente utilizada para inicializar a variável que irá controlar o número de repetições do laço.

- **Condição** – é a expressão que controla a parada das repetições. Se for verdadeira, o bloco de comandos é executado novamente; se for falsa, termina.

- **Incremento** – define a maneira como a variável de controle do laço será alterada a cada vez que o laço for repetido. Ela é executada ao final da execução de cada repetição do corpo do laço.

Veja exemplo na Figura 1.15 a seguir.

```
<?php
      // Exemplo que gera a sequencia de Fibonacci
      // Um número é a soma de seus 2 antecessores
      $nc = 0;   // Número corrente
      $a1 = 1;   // Número anterior 1
      $a2 = 1;   // Número anterior 2
      echo "1 <BR>1 <BR>";
      for ($i=0; $i < 50; $i++) {
         $nc = $a1 + $a2;
         $a2 = $a1;
         $a1 = $nc;
         echo "$nc <BR>";
      };
?>
```

Figura 1.15: Exemplo do *for*
Fonte: Elaborada pelo autor

Atividades de aprendizagem 1.1: Crie um código em PHP que exiba uma sequência de números de 1 a 100. Os números pares devem ser formatados em negrito e os números ímpares em itálico.

1.8 Array

Os *arrays* são estruturas para armazenar valores que precisam ser indexados. Diferentemente do C, em que os índices são apenas números inteiros e consecutivos, em PHP os índices podem ser de vários tipos. Mesmo se forem inteiros não precisam ser consecutivos. Os valores armazenados não precisam ser do mesmo tipo. Veja um exemplo na Figura 1.16, onde **print_r** mostra todos os elementos do *array*.

```
<?php
      $estados[0] = "Parado";
      $estados[1] = "Executando";
      $estados[2] = "Finalizado";
      $estados["ES"] = "Espírito Santo";
      $estados["RJ"] = "Rio de Janeiro";

      echo "<PRE>";
      print_r($estados);  // Mostra todo o array
      echo "</PRE>";
      echo $estados[2] . "<BR>";
      echo $estados["RJ"];
?>
```

Figura 1.16: Exemplo do *array*
Fonte: Elaborada pelo autor

Perceba que podemos trabalhar com índices diferentes em um mesmo *array*. Outra forma semelhante para inicializar o *array* pode ser vista na Figura 1.17, que é idêntica a da Figura 1.16 a seguir.

```php
<?php
      $estados =  array (
            0 => "Parado",
            1 => "Executando",
            2 => "Finalizado",
            "ES" => "Espírito Santo",
            "RJ" => "Rio de Janeiro"
      );

      echo "<PRE>";
      print_r($estados);  // Mostra todo o array
      echo "</PRE>";
      echo $estados[2] . "<BR>";
      echo $estados["RJ"];
?>
```

Figura 1.17: Outra forma de inicializar o *array*
Fonte: Elaborada pelo autor

Para mais detalhes sobre funções para manipulação de *arrays*, consulte:
SOARES, Walace. **PHP 5**: conceitos, programação e integração com banco de dados. 3ª ed. São Paulo: Érica, 2007.

Crie um *array* e preencha com alguns estados, indexados pela sigla, como no programa na Figura 1.16. Faça uma função que receba esse *array* e uma sigla como parâmetro e mostre o nome do estado.

1.9 Funções

As funções em PHP seguem o mesmo princípio das de outras linguagens. A diferença é que como não precisamos declarar os tipos, a lista de parâmetros possui apenas o nome das variáveis. As funções que retornam valor também não precisam informar o tipo de retorno. A sintaxe é:

```
function nome_função (lista de parâmetros) { ... }
```

Para organizar melhor o código, as declarações das funções ficam dentro do bloco <HEAD>, em uma página HTML. As chamadas das funções ficam no <BODY>. Veja exemplos de funções na Figura 1.18 a seguir.

```php
<html>
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=UTF-8">
        <title>Exemplo de funções</title>
<?php
    function cabecalho() {  // Sem parâmetro
        echo "PROGRAMA DE TESTE DE FUNÇÕES<BR>";
    }

    function quadrado ( $n ) {
        return ($n * $n);   // Retorna o resultado
    }

    function soma ( $n1, $n2) {
        return ($n1 + $n2);
    }
?>
    </head>
    <body>
        <?php
           $x = 4;
           $resQ = quadrado($x);
           $resS = soma($x,$x);
           cabecalho();
           echo "O quadrado de $x é $resQ   <BR>";
           echo "A soma de $x e $x é $resS  <BR>";
        ?>
    </body>
</html>
```

Figura 1.18: Exemplos de funções
Fonte: Elaborada pelo autor

Por definição, a passagem de parâmetros é por valor. Caso se queira passar os parâmetros por referência, para alterar uma variável dentro da função, utiliza-se o símbolo '&' antes do parâmetro. Veja a Figura 1.19 a seguir.

```php
<?php
    function troca ( &$n1, &$n2 ) {
        // Troca os valores
        $aux = $n1;
        $n1 = $n2;
        $n2 = $aux;
    }

    $x = 4;
    $y = 5;
    echo "ANTES. X = $x e Y = $y <BR>";
    troca($x, $y);
    echo "DEPOIS. X = $x e Y = $y <BR>";
?>
```

Figura 1.19: Exemplo de função com passagem de parâmetro por referência
Fonte: Elaborada pelo autor

Com os conceitos e comandos aprendidos nesta aula é possível criar páginas *web* básicas em PHP. O conhecimento desses comandos é de extrema importância para os demais recursos que serão abordados nas próximas aulas. Portanto, somente passe para a próxima aula se todos esses conceitos e comandos foram entendidos e bem praticados.

Resumo

Nesta primeira aula vimos como instalar e configurar o servidor *web* Apache e o módulo PHP. Aprendemos a sintaxe do PHP e com isso criamos nossa primeira página *web*, ainda sem muitos recursos e sem interatividade. Na próxima aula entenderemos como os dados de um formulário são manipulados em um servidor com PHP, aprenderemos a enviar um arquivo para o servidor a partir de uma página *web* e conheceremos a forma de gerenciar as informações de uma conexão com o servidor.

Leituras complementares:
SOARES, Walace. **PHP 5:** conceitos, programação e integração com banco de dados. 3ª ed. São Paulo: Érica, 2007.
GUTMANS, Andi; BAKKEN, Stig Saether; RETHANS, Derick. **PHP 5:** programação ponderosa. Rio de Janeiro: Alta Books, 2005.

***Links* interessantes**:
http://www.php.net
http://www.apache.org
http://pt.wikipedia.org/wiki/PHP

Atividade de aprendizagem

1. Pesquise nos livros os comandos para pegar a data do sistema. Faça uma função de saudação da sua página. Essa função deverá pegar a hora corrente e mostrar na tela a mensagem abaixo, com sua respectiva formatação.

0 <= Hora < 12 à "BOM DIA" (em vermelho)
12 <= Hora < 18 à "BOA TARDE" (em verde)
18 <= Hora < 24 à "BOA NOITE" (em azul)

Aula 2 – Recebendo dados do formulário

Objetivos

Entender como os dados de um formulário são manipulados em um servidor com PHP.

Aprender e exercitar o envio de um arquivo para o servidor a partir de uma página *web*.

Conhecer a forma de gerenciar as informações de uma conexão com o servidor.

Após o conhecimento da sintaxe dos comandos em PHP, vamos agora criar as páginas dinâmicas tratando as informações obtidas de outras páginas, como por exemplo, de formulários.

A partir de um formulário em HTML em um navegador, o usuário envia dados para uma página no servidor. Aprenderemos os comandos em PHP para capturar esses dados e fazer sua validação.

É extremamente importante que vocês revisem o conteúdo de formulários HTML vistos na disciplina **Fundamentos do Desenvolvimento *Web***.

2.1 Métodos GET e POST

Na definição de um formulário em HTML, três atributos serão importantes para a criação de páginas em PHP no servidor: **action, method** e **enctype**. Vejamos um exemplo na Figura 2.1 a seguir.

```
<FORM name="cadcliente"
   action="http://localhost/cadastro.php"
   method="POST"
   enctype="application/x-www-form-urlencoded">
   . . .
</FORM>
```

Figura 2.1: Exemplo de definição de um formulário em HTML
Fonte: Elaborada pelo autor

- **name**– nome do formulário. Útil para referência em funções *javascript*.

- **action** – arquivo no servidor que será chamado para tratar os dados do formulário. A página em HTML com o formulário irá passar os dados para o programa PHP especificado por este campo.

- **method** – método de envio de dados para o servidor. Dois tipos são permitidos GET e POST. Mais detalhes a seguir.

- **enctype** – define o formato de como os dados serão enviados para o servidor. Necessário para o envio de arquivos, que será tratado mais adiante. Por enquanto, nem precisamos especificar nada. O exemplo acima mostra o valor padrão, que pode ser omitido.

A diferença entre GET e POST está na forma como os dados são enviados para o servidor. No método POST, os dados são enviados ocultos. É o método recomendado quando se utiliza formulário.

Já no método GET, os dados são enviados de forma aberta, na URL, na forma do par **'campo=valor'**. Para isso, utiliza-se o símbolo **'?'** depois do nome do arquivo. Os pares 'campo=valor' são separados pelo símbolo **'&'**. Essa forma de envio é utilizada quando temos poucas informações a serem passadas, e elas podem ser especificadas direto na URL. Utiliza-se principalmente quando queremos passar dados através de um *link*, sem os campos de um formulário. Exemplo:

```
http://www.xxx.com/cons.php?nome=João&cidade=Colatina
```

Nesse exemplo foi utilizado o método GET, que chama o programa *cons.php*, passando o campo **nome** igual a 'João' e o campo **cidade** igual a 'Colatina'.

Um exemplo mais real pode ser visto no *site* do **Google**. Depois que você fizer uma pesquisa qualquer, verifique o endereço que aparece. Perceba que existem vários pares **'campo=valor'** que são passados para o servidor, mas que não têm formulário para digitar.

2.2 Obtendo e validando os dados

Para enviar dados para o servidor, vamos criar um formulário como o da Figura 2.2 a seguir. Preste atenção no nome de cada campo. Esse nome será usado no programa PHP. Veja também o atributo *action* do <FORM>, ele informa o nome do arquivo PHP.

```
<html>
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Cadastro de Clientes</title>
</head>
<body>
<form method="post" name="formCadastro"
      action="http://localhost/cadcliente.php" >
  <H1>Cadastro de clientes</H1>
  <table width="100%">
    <tr>
      <th width="18%"> Nome</th>
      <td width="82%"><input type="text" name="txtNome">
      </td>
    </tr>
    <tr>
    <th>CPF</th>
    <td><input name="txtCPF" type="text" maxlength="14">
    </td>
    </tr>
    <tr>
      <th>Endereço</th>
    <td><textarea name="txtEndereco" cols="30" rows="4">
        </textarea>    </td>
    </tr>
    <tr>
  <th>Estado</th>
     <select name="listEstados" >
        <option value="BA">Bahia</option>
        <option value="ES">Espírito Santo</option>
        <option value="MG">Minas Gerais</option>
        <option value="RJ">Rio de Janeiro</option>
        <option value="SP">São Paulo</option>
     </select>        </td>
</tr>
<tr>
   <th>Data Nasc.</th>
   <td><input name="txtData" type="text"></td>
</tr>
<tr>
  <th>Sexo</th>
  <td>
  <input type="radio" name="sexo" value="M" checked>
      Masculino     <BR>
  <input type="radio" name="sexo" value="F" />
      Feminino      </td>
</tr>
<tr>
  <th>Áreas de Interesse</th>
   <td>
      <input name="checkCinema" type="checkbox"
             value="true" /> Cinema    <BR>
      <input name="checkMusica" type="checkbox"
             value="true" /> Música    <BR>
      <input name="checkInfo" type="checkbox"
             value="true" /> Informática          </td>
 </tr>
 <tr>
    <th>Login</th>
    <td><input name="txtLogin" type="text"></td>
 </tr>
 <tr>
    <th>Senha</th>
    <td><input name="txtSenha1" type="password"></td>
 </tr>
 <tr>
    <th>Confirmação Senha</th>
    <td><input name="txtSenha2" type="password"></td>
 </tr>
 <tr>
   <td>
<input type="submit" name="btnEnviar" value="Enviar">
   </td>
   <td>
<input type="reset" name="btnLimpar" value="Limpar">
   </td>
      </td>
    </tr>
  </table>
</form>
</body>
</html>
```

Figura 2.2: Exemplo de formulário para envio dos dados
Fonte: Elaborada pelo autor

Seja passando os dados via POST ou GET, o programa no servidor para capturar e tratar os dados será o mesmo. Os dados enviados para o programa PHP serão transformados em um *array*, **$_POST**, se o método de envio for o POST, e **$_GET**, se o método de envio foi o GET. Esses vetores são indexados pelos nomes dos campos.

Se o método de envio for o POST, os nomes dos campos serão aqueles informados no atributo **name** dos comandos HTML. Se o envio for por GET, então o nome dos campos serão aqueles especificados no endereço pelo par **'campo=valor'**.

Vamos agora criar um arquivo PHP com o mesmo nome dado no *action* do formulário (no exemplo **cadcliente.php**). Nessa página, vamos capturar os dados enviados e mostrar uma página de resposta com esses dados. Para pegar cada campo, devemos informar qual o *array* que contém os dados ($_POST ou $_GET) e entre colchetes [ ] o nome do campo, definido no formulário HTML.

Depois de pegar os dados, podemos verificar se foram preenchidos e se estão corretos. Por exemplo: verificar se o campo foi preenchido, verificar datas, validar CPF, verificar se o *e-mail* é válido, etc.

A Figura 2.3 mostra o código que recebe os dados e verifica se alguns campos estão preenchidos; caso não estejam, mostra uma mensagem de erro e, por último, exibe na tela os dados informados. Mais adiante aprenderemos a inserir esses dados no banco de dados.

```
<html>
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Cadastro de Clientes</title>
</head>
<body>
    <H1>Os dados informados são:</H1>
    <?php
        // Recebe cada campo do formulário
        // e coloca em uma variável.
        $nome = $_POST["txtNome"];
        $ender = $_POST["txtEndereco"];
        $cpf = $_POST["txtCPF"];
        $estado = $_POST["listEstados"];
        $dtNasc = $_POST["txtData"];
        $sexo = $_POST["sexo"];
        $cinema = $_POST["checkCinema"];
        $musica = $_POST["checkMusica"];
        $info = $_POST["checkInfo"];
        $login = $_POST["txtLogin"];
        $senha1 = $_POST["txtSenha1"];
        $senha2 = $_POST["txtSenha2"];
```

```
    // Verificar campos
      $camposOK = true;  // Determina se ocorreu erro
      if ( $nome == "" ) {
          echo "Informe o NOME. <BR>";
          $camposOK = false;
      }
      if ( $ender == "" ) {
          echo "Informe o ENDEREÇO. <BR>";
          $camposOK = false;
      }
      // Verificar se as SENHAS conferem
      if ( $senha1 != $senha2 ) {
          echo "As SENHAS não conferem!. <BR>";
          $camposOK = false;
      }
      // *** Acrescentar as validações de CPF e DATA

      // Mostrando os valores em forma de tabela
      // Cada campo é uma linha <TR> da tabela
      if ( $camposOK ) {
         echo "<TABLE border='0' cellpadding='5'>";
         echo "<TR><TD>NOME:</TD><TD><B> $nome </B></TD></TR>";
            echo "<TR><TD>CPF:</TD><TD><B> $cpf </B></TD></TR>";
            echo "<TR><TD>ENDEREÇO:</TD><TD><B> $ender </B></TD></TR>";
            echo "<TR><TD>ESTADO:</TD><TD><B> $estado </B></TD></TR>";
            echo "<TR><TD>DATA NASC.:</TD><TD><B> $dtNasc
</B></TD></TR>";
            echo "<TR><TD>SEXO:</TD><TD><B> $sexo </B></TD></TR>";
            echo "<TR><TD>LOGIN:</TD><TD><B> $login </B></TD></TR>";
            echo "<TR><TD>SENHA:</TD><TD><B> $senha1 </B></TD></TR>";

     // Campos do tipo CheckBox retornam
     // Verdadeiro (true) se foi marcado
         echo "<TR><TD>ÁREAS DE INTERESSE:</TD><TD><B>";
            if ( $cinema == true ) {
                echo "Cinema <BR>";
            }
            if ( $musica ) {
                echo "Música <BR>";
            }
            if ( $info ) {
                echo "Informática ";
            }
            echo "</B></TD></TR></TABLE>";
        } // Fim IF camposOK
    ?>
</body>
</html>
```

Figura 2.3: Página PHP que recebe os dados
Fonte: Elaborada pelo autor

A Figura 2.4 mostra a página exibida pelo programa acima, com os valores digitados no formulário.

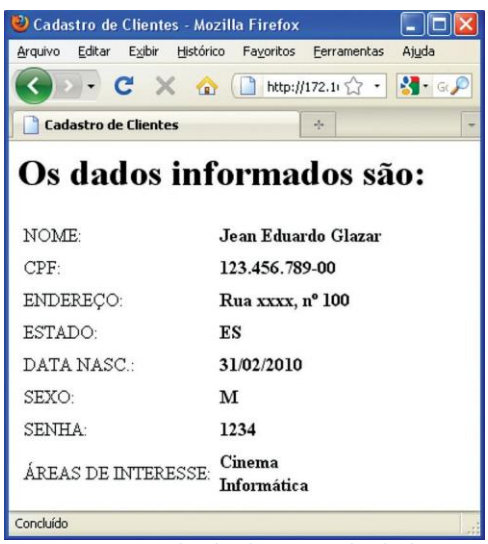


Figura 2.4: Resultado do envio de dados
Fonte: Página exibida pelo Mozilla Firefox 3.5

Funções para validar datas, CPF, *e-mail*, etc. são facilmente encontradas na internet. Veja alguns *links*:
http://codigofonte.uol.com.br/codigos/php/validacao

http://www.revistaphp.com.br

http://www.criarweb.com/php

http://phpbrasil.com

http://www.codigofonte.net/scripts/php.

Note que os valores dos campos do tipo ***ComboBox***e ***ListBox*** são os valores definidos no atributo ***value*** da opção escolhida (<option>) e não o nome que aparece no formulário. No exemplo acima, o estado é 'ES'.

O valor do campo do tipo ***RadioButton*** também é o valor do atributo ***value***. No exemplo acima, o sexo é 'M' ou 'F'.

Campos do tipo ***CheckBox*** retornam ***true***(verdadeiro) se foram marcados e ***false*** (falso), caso contrário.

Pesquise na internet as funções para validar data e CPF. Acrescente essas funções ao programa da Figura 2.3 para validar nosso campo CPF e data de nascimento. Informe as mensagens de erro em vermelho.

2.3 *Upload* de arquivos

Para enviar um arquivo para o servidor, como por exemplo uma foto, precisamos inserir um campo apropriado no formulário. Esse campo exibe um botão "Procurar" que abre a janela para escolher um arquivo. O comando é:

```
<INPUT type="file" name="nome_do_campo">
```

Para testar, vamos acrescentar um campo ao formulário da Figura 2.2 para fazer o *upload* de uma foto. Precisamos mudar a tag <FORM>, adicionando o atributo **enctype** e colocar mais uma linha, depois da confirmação da senha, para fazer o *upload* da foto. Os comandos alterados e acrescentados estão na Figura 2.5 a seguir.

Para o *upload* funcionar é necessário acrescentar o atributo **enctype** dentro da tag <FORM>, assim: **enctype="multipart/form-data"**.

```
. . .

<form method="post" name="formCadastro"
     action="http://localhost/cadcliente.php"
     enctype="multipart/form-data" >

. . .

   <tr>
      <th>Foto</th>
     <td><input type="file" name="txtFoto" ></td>
   </tr>

. . .
```

Figura 2.5: Formulário com campo para *upload* de arquivo
Fonte: Elaborada pelo autor

Quando o arquivo é enviado para o servidor, várias informações são armazenadas em um vetor chamado **$_FILES**, como o nome, o tipo, o tamanho, etc. Para obter esses dados, devemos informar o nome do campo entre colchetes, assim:

```
$arquivo = $_FILES['nome_do_campo'];
```

A variável **$arquivo** foi utilizada para receber o arquivo. A partir desse comando, sempre que pretendermos trabalhar com o arquivo, utilizaremos a variável **$arquivo**. Diversas informações podem ser obtidas a partir dessa variável, como: códigos de erros, se ocorrerem, tamanho, tipo e o nome temporário do arquivo.

Se ocorrer algum erro no *upload*, o código de erro fica armazenado no atributo **'error'**. O código zero significa que não ocorreu nenhum erro. Para obter esse código, utilize o seguinte comando:

```
$arquivo['error']
```

As verificações que podemos fazer são em relação ao tamanho, ao tipo do arquivo, etc. No nosso exemplo, vamos verificar se o arquivo é uma imagem (gif, jpg, png, bmp) e se o tamanho é menor que 100.000 *bytes*. Os comandos para pegar o tipo e o tamanho do arquivo, respectivamente, são:

```
$arquivo['type']    e    $arquivo['size']
```

O arquivo, propriamente dito, quando chega ao servidor, é colocado em um diretório temporário, configurado no arquivo **php.ini**. Para mover para o diretório definitivo, usa-se o seguinte comando:

```
move_uploaded_file($arquivo['tmp_name'], destino)
```

em que:

$arquivo['tmp_name'] – pega o nome do arquivo temporário; e
destino – diretório de destino do arquivo.

O diretório de destino deve existir e com permissão para escrita.

Se a cópia for realizada com sucesso, esse comando retorna *true*. Caso ocorra algum erro, como por exemplo, o diretório de destino não existe ou não tem permissão de gravação, essa função retornará *false*.

A última etapa é mostrar a foto que foi enviada. Para isso basta colocar a tag <IMG> com o diretório destino para onde o arquivo foi copiado. A Figura 2.6 mostra o trecho de código para lidar com o *upload* do arquivo. Esse código deve ser colocado no programa da Figura 2.3, antes da parte que mostra os dados na tela. A Figura 2.7 mostra o resultado do envio dos dados com a imagem.

```
. . .
// Obtendo a foto
$arquivo = $_FILES["txtFoto"];

// Verificando erro no upload
if ( $arquivo['error'] != 0 ) {
   echo "Erro no UPLOAD do arquivo. <BR>";
   $camposOK = false;
}

// Verificando o tamanho
if ( $arquivo['size'] == 0 ) {
   echo "Erro no arquivo. Tamanho igual a ZERO. <BR>";
   $camposOK = false;
}
if ( $arquivo['size'] > 100000 ) {
   echo "Tamanho maior que o permitido (100 kbytes). <BR>";
   $camposOK = false;
}

// Verificando o tipo do arquivo
if ( ( $arquivo['type'] != "image/gif" ) &&
     ( $arquivo['type'] != "image/jpeg" ) &&
    ( $arquivo['type'] != "image/pjpeg" ) &&
     ( $arquivo['type'] != "image/x-png" ) &&
    ( $arquivo['type'] != "image/png" ) &&
     ( $arquivo['type'] != "image/bmp" ) )  {
   echo "Erro no arquivo. TIPO não permitido. <BR>";
   $camposOK = false;
}
// Movendo o arquivo para algum diretório válido
// dentro do www
$destino = "/var/www/ProgWeb/imagens/";
$destino = $destino . $arquivo['name'];
$res=move_uploaded_file($arquivo['tmp_name'],$destino);
if ( $res == false ) {
   echo "Erro ao copiar o arquivo para o destino.<BR>";
   $camposOK = false;
}
// Mostrando a foto. Coloque o caminho onde está a foto
echo "<IMG heigth=120 width=120
          src='imagens/" . $arquivo['name'] . "'>";
. . .
```

Figura 2.6: Página PHP que trata um arquivo recebido
Fonte: Elaborada pelo autor

Figura 2.7: Resultado do envio de dados com foto
Fonte: Página exibida pelo Mozilla Firefox 3.5

Teste o programa da Figura 2.6.

2.4 Headers

Os ***headers*** servem para o gerenciamento da conexão entre o PHP e o navegador do usuário, podendo trocar informações contidas no cabeçalho HTTP de uma página.

Os ***headers*** são informações trocadas entre o navegador e o servidor de maneira transparente ao usuário, e podem conter dados sobre o tipo e a versão do navegador, a página de onde partiu a requisição (*link*), os tipos de arquivos aceitos como resposta, e uma série de outras informações.

A sintaxe da função ***header***:

header(cabeçalho)

Em que: **cabeçalho** é um comando já definido para indicar o tipo de cabeçalho HTTP a ser usado.

Por exemplo, o cabeçalho “HTTP/” indica que um código de retorno é enviado para o navegador do cliente. O exemplo na Figura 2.8 mostra o envio de uma mensagem de “Página não encontrada”:

```
<?php
   header("HTTP/1.0 404 Página não encontrada");
   exit;
?>
```

Figura 2.8: Exemplo do envio de uma mensagem usando o *header*
Fonte: Elaborada pelo autor

Uma segunda forma de usar o *header* é para redirecionar para outra página. Esse comando é muito útil para, ao final de uma página puramente em PHP, redirecionar para outra página padronizada de resposta ou de erro. Usa-se o comando “***Location***” e logo em seguida o nome da página que se pretende redirecionar. Veja um exemplo na Figura 2.9:

```
<?php
   header("Location:principal.php");
   exit;
?>
```

Figura 2.9: Exemplo de redirecionamento de página usando o *header*
Fonte: Elaborada pelo autor

O comando ***header*** deve ser usado antes de qualquer comando de exibição (*echo*, *tags* HTML, *include*).

Usaremos o ***header*** mais adiante como forma de redirecionamento de página.

Com os conhecimentos desta aula você tem a capacidade de criar páginas *web* dinâmicas, nas quais, a partir de um formulário em HTML em um navegador, o usuário possa enviar dados para um programa no servidor e receber uma mensagem. Esse fluxo de informação é a base de todas as páginas *web* dinâmicas.

Resumo

Nesta aula programamos nossa primeira página dinâmica com recebimento e validação de dados. Também aprendemos a enviar um arquivo junto com os dados. Porém, nossas páginas ainda não estão completas. Falta salvar os dados para que eles não se percam. Para isso é preciso acessar banco de dados a partir no PHP, que será nossa próxima Aula.

Atividade de aprendizagem

Acrescente no formulário da Figura 2.2 mais um campo para fazer o *upload* de um arquivo do tipo PDF, como se fosse o envio do currículo da pessoa. Altere o programa da Figura 2.3 para receber esse arquivo, fazer as suas validações (tipo e tamanho) e coloque um *link* para esse arquivo, com o nome da pessoa, para quando clicar no *link*, abrir o arquivo PDF.

Leituras complementares:

SOARES, Walace. **PHP 5:** conceitos, programação e integração com banco de dados. 3ª ed. São Paulo: Érica, 2007.

GUTMANS, Andi; BAKKEN, Stig Saether; RETHANS, Derick. **HP 5:** programação ponderosa. Rio de Janeiro: Alta Books, 2005.

MELO, Alexandre Altair de; NASCIMENTO, Maurício G. F. **PHP profissional**. 2ª ed. São Paulo: Novatec, 2007.

Aula 3 – Acesso, inserção e listagem no banco de dados MySQL

Objetivos

Conhecer como o PHP interage com o banco de dados.

Construir páginas *web* para inserir dados em um banco de dados.

Construir páginas *web* para recuperar informações do banco de dados.

Até agora já aprendemos a criar páginas dinâmicas que recebem e validam os dados, mas essas informações estão se perdendo. Precisamos armazená-las em algum lugar. Esse local é o banco de dados.

Como pré-requisito, é fundamental que você revise a linguagem SQL, estudada na disciplina de **Banco de Dados**, por ser essa a linguagem universal dos bancos de dados. É por meio dela que o PHP irá "conversar" com o banco de dados.

A maioria dos *sites* dinâmicos acessa algum banco de dados. Em alguns casos, somente para tarefas simples, como cadastrar usuários e senhas. Em outros casos, o banco de dados é vital para o funcionamento do sistema *web*, como uma loja virtual.

Com o PHP podemos acessar diversos banco de dados, como o MySQL, PostgreSQL, Oracle, SQL Server, Firebird, Sysbase, Informix, SQLite e outros mais. Para os bancos de dados que o PHP não tem um módulo específico, podemos utilizar os *drivers* ODBC.

Um dos bancos de dados mais utilizados com o PHP é o MySQL. O PHP possui um módulo específico para esse banco. Utilizaremos o MySQL em nossos exemplos por ser um banco simples de operar e utilizar pouco processamento e memória, em comparação com os outros.

3.1 Criando o banco de dados

Antes de construir o *site*, devemos modelar os dados a serem manipulados pelas páginas, porque assim teremos uma melhor visão das informações a serem acessadas em cada página. Veja a seção sobre “Modelagem de Dados” da disciplina **Banco de Dados**.

Para usarmos em nossos exemplos, vamos criar um banco de dados no MySQL com o nome **ProgWebBD** e construir as tabelas descritas na Figura 3.1 a seguir.

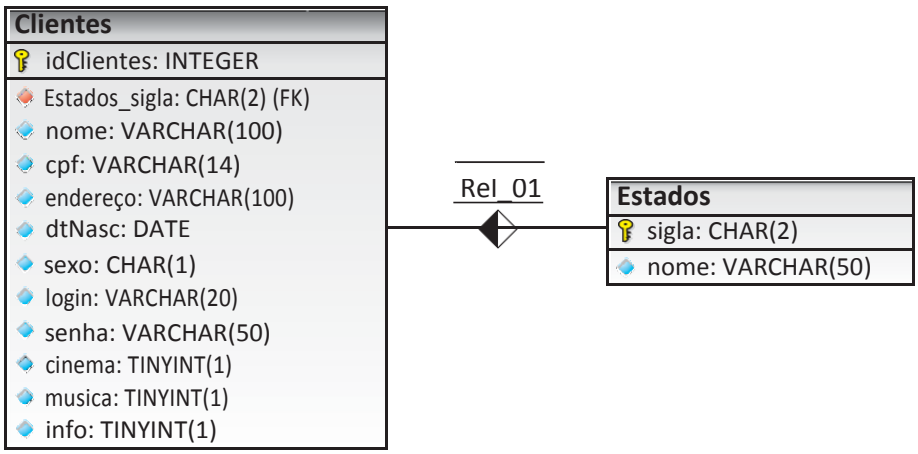


Figura 3.1: Tabelas a serem utilizadas nos exemplos
Fonte: Equipe produção CEAD/IFES (2011)

O campo **idClientes** é do tipo “autoincremento”.

Os campos **cinema, musica** e **info** armazenam 1 se aquela opção foi marcada, e 0 caso contrário.

3.2 Conectando ao banco de dados

Em uma página PHP, o primeiro passo é conectar com o banco de dados. Utilizaremos o comando **mysql_connect** para criar essa conexão.

A conexão é a “estrada” por onde trafegarão os dados da sua página até o banco de dados, e vice-versa.

mysql_connect – abre a conexão de uma página em PHP com o banco de dados. Sua sintaxe é:

mysql_connect(servidor, usuário, senha)

onde:

- **servidor – IP** (ou *hostname*) é a porta do servidor onde está o banco de dados, no formato **servidor: porta**. Se o banco de dados estiver no mesmo computador, pode usar *localhost*. Se a porta não for informada, será utilizada a porta padrão, que no MySQL é a 3306.

- **Usuário** e **senha** cadastrados no banco de dados.

Podem ocorrer alguns erros ao se tentar abrir a conexão. Os casos mais comuns são: não encontrar o servidor ou o usuário e a senha não terem permissão de acesso ao banco. Caso ocorra algum erro, o ideal seria mostrar uma mensagem de erro e interromper a execução da página, já que sem a conexão não poderemos acessar o banco de dados. Para isso usaremos a função **die**.

die – função que exibe uma mensagem e interrompe a execução da página. Sua sintaxe é:

die("mensagem")

A mensagem pode ser concatenada com a função **mysql_error**, que informa a mensagem original do erro. Exemplo:

```
die("Erro ao conectar. " . mysql_error() );
```

O ponto '**.**' serve para juntar as duas mensagens, ou seja, ele é o operador que concatena *strings*.

O **die** pode ser usado junto com outra função. Se der erro nessa função, o **die** é chamado automaticamente. Utiliza-se o operador **or** para associar o **die** a alguma função. O exemplo abaixo chama a função **mysql_connect;** se der erro, automaticamente o **die** é executado para mostrar a mensagem e interromper a execução da página.

```
mysql_connect(localhost,"root","root") or die("Erro ao conectar. " .
mysql_error() );
```

Caso a abertura da conexão ocorra normalmente, o segundo passo é escolher o nome do banco de dados que será utilizado. Em um servidor de banco de dados podem existir vários bancos. Usaremos para isso a função **mysql_select_db**.

mysql_select_db – seleciona o banco a ser utilizado no servidor conectado anteriormente. Sua sintaxe é:

mysql_select_db("nome_banco")

Exemplo: Selecionar o banco de dados criado anteriormente para os exemplos.

mysql_select_db("ProgWebBD");

O programa da Figura 3.2 mostra o exemplo de conexão com o nosso banco de dados de exemplo. O servidor de banco de dados MySQL está no servidor com IP 172.16.43.10, usuário "root", senha "root" e o nome do banco criado foi "ProgWebBD".

```
<?php

   // PASSO 1 - CONECTAR AO BANCO DE DADOS
   mysql_connect("172.16.43.10", "root", "root" )  or die("Não
foi possível conectar ao BANCO DE DADOS. " . mysql_error() );

   // PASSO 2 - SELECIONAR O BANCO DE DADOS
   mysql_select_db("ProgWebBD") or die("Não foi possível
encontrar o banco. " . mysql_error() );

?>
```

Figura 3.2: Exemplo de conexão com o banco de dados
Fonte: Elaborada pelo autor

Cada página que necessitar acessar o banco de dados deverá ter esses dois comandos no início.

Como um sistema *web* geralmente possui várias páginas, replicar esses comandos não será uma boa solução. Caso tenha que mudar algum parâmetro, como por exemplo o IP do servidor, todas as página sofrerão modificações. Um trabalho e tanto!

Para evitar esse trabalho de manutenção, colocam-se os comandos de conexão com o banco em um único arquivo e todas as páginas fazem acesso a esse arquivo utilizando o comando:

include ou include_once

include – insere pedaços de códigos PHP de um determinado arquivo na página atual.

include_once – a diferença é que este comando verifica se o arquivo já foi inserido anteriormente, ou seja, insere somente uma única vez.

Sua sintaxe é:

include("nome_arquivo.php") ou
include_once("nome_arquivo.php")

Então, o código da Figura 3.2 ficará em um arquivo, por exemplo, **"conexaobd.php"**, e todas as outras páginas incluirão esse código da seguinte forma:

```
include_once("conexaobd.php")
```

PRONTO!!! Uma vez conectado com o banco de dados, podemos realizar todas as operações para manipulação dos dados: inserir, pesquisar, alterar e excluir. O que precisaremos saber para realizar essas operações é sobre a linguagem SQL.

Uma conexão estabelecida com o comando **mysql_connect** é encerrada, automaticamente, ao final da execução da página. Caso queira encerrá-la antes disso, deve ser utilizado o comando **mysql_close**.

mysql_close – fecha a conexão com o banco de dados. Sua sintaxe é:

mysql_close(identificador)

em que: **identificador** é a variável que indica a conexão criada. Se o identificador não for fornecido, a última conexão estabelecida será encerrada.

3.3 Inserindo dados

A linguagem padrão de comunicação com os bancos de dados é a linguagem SQL. Para fazer com que o PHP execute os comandos SQL no banco de dados MySQL, utiliza-se a função **mysql_query**.

mysql_query – função que executa um comando SQL no banco de dados MySQL. Retorna verdadeiro (*true*) em caso de sucesso e falso (*false*) caso contrário. Sua sintaxe é:

mysql_query(comando, conexao)

em que:

- **comando** – é o comando na linguagem SQL, como: INSERT, SELECT, UPDATE, DELETE, etc.
- **conexão** – parâmetro opcional que indica a conexão com o banco de dados. Se não for informada, utiliza a última conexão aberta.

Portanto, para inserir os dados no banco, o comando em SQL que é utilizado é o INSERT. Para testarmos, vamos criar duas páginas: uma em HTML com o formulário para digitar os dados e a outra em PHP, que irá receber os dados, verificar se estão corretos e depois inserir no banco.

O formulário conterá os mesmos campos que a tabela **Clientes** do nosso banco de dados de teste da Figura 3.1. Esse formulário será o mesmo utilizado na Figura 2.2, da Aula 2, com a inclusão de mais um campo para o *login*, chamado "txtLogin". Esse formulário enviará os dados para a página em PHP no servidor chamada **"cadclientebd.php"** (atributo *action* da *tag* <FORM>).

O programa em PHP no servidor será semelhante ao programa da Figura 2.3, da Aula 2. A diferença ficará por conta da inclusão dos dados no banco, ao invés de mostrar os dados na página. Para isso, vamos criar uma variável ($sql) que receberá o comando INSERT com os dados do formulário. Logo após, essa variável será passada para o **mysql_query**.

Na sintaxe do INSERT, a parte do VALUES é em que passamos os valores para o banco de dados. É nesse ponto que usaremos as variáveis com os valores obtidos dos formulários. A Figura 3.3 apresenta este comando:

```
$sql = "INSERT INTO Clientes ( nome, cpf, endereco,
estado, dtNasc, sexo, login, senha, cinema, musica,
info ) VALUES ( '$nome' , '$cpf', '$ender', '$estado',
'$dtNasc', '$sexo', '$login', '$senha1', $cinema,
$musica, $info ) ";
```

Figura 3.3: Exemplo de montagem do comando SQL
Fonte: Elaborada pelo autor

Lembre-se que na linguagem SQL, os campos do tipo *string* e data devem estar entre aspas simples (‘ ’). No exemplo acima, os campos **$cinema, $musica** e **$info** não estão entre aspas porque são do tipo **inteiro**.

Logo em seguida chamaremos o comando em PHP para executar esse SQL no banco de dados. Esse comando é o **mysql_query** e pode ser visto na Figura 3.4.

```
mysql_query( $sql ) or die ("ERRO ao inserir dados de cliente. $sql");
```

Figura 3.4: Exemplo de chamada para executar o comando SQL no MySQL
Fonte: Elaborada pelo autor

Os campos no banco de dados do tipo “autonumeração” (ou “autoincremento”) não devem ser passados para o comando INSERT.

Verifique que em nossa tabela de exemplo, o campo **idClientes** é um campo autoincremento e ele não foi passado para o INSERT.

Caso queira obter o último número inserido de um campo “autoincremento”, utiliza-se a função **mysql_insert_id**.

mysql_insert_id – retorna o último número inserido de um campo do tipo “autoincremento”. Sua sintaxe é:

mysql_insert_id(conexao)

em que: **conexão** é um parâmetro opcional que indica a conexão com o banco de dados. Se não for informada, utiliza a última conexão aberta.

Em nosso exemplo, poderíamos criar a variável $cod para receber o código do cliente inserido, conforme exemplo na Figura 3.5 a seguir.

```
$cod = mysql_insert_id();
```

Figura 3.5: Obtendo o último código “autoincremento” inserido
Fonte: Elaborada pelo autor

Os campos do formulário do tipo **checkbox** enviam para a página PHP o valor ***true*** se estiverem marcados e “ ” (vazio) se não estiverem marcados. Portanto, antes de passar essa variável para o comando INSERT, devemos substituir o “ ” (vazio) por zero, já que em nosso banco esses campos armazenam 0 ou 1.

O valor ***true*** corresponde ao 1; portanto, não teria problema ao passar para o SQL.

Os comandos para substituir o “ ” (vazio) por zero pode ser visto na Figura 3.6:

```
if ( $cinema == "" ) $cinema = 0;
if ( $musica == "" ) $musica = 0;
if ( $info == "" ) $info = 0;
```

Figura 3.6: Substituindo o “ ” (vazio) por zero
Fonte: Elaborada pelo autor

3.3.1 Inserindo datas no banco

O MySQL trabalha com o tipo data no formato americano, ou seja, “ano-mês--dia”. Portanto é preciso converter o nosso formato de data (“dia-mês-ano”) para o formato americano, antes de passar a data para o comando INSERT.

Admitindo que a data recebida esteja no formato **“dd/mm/aaaa”**, separam-se os campos dia, mês e ano utilizando a função **substr**.

substr – retorna uma parte de uma *string*. Sua sintaxe é:

substr(texto, pos_inicial, tamanho

onde:

- **texto** – *string* original.
- **pos_inicial** – posição onde inicia o pedaço da *string* que se deseja. A primeira letra está na posição zero.
- **tamanho** – da *substring* a partir da posição inicial.

Logo, o dia é o pedaço da data que inicia na posição 0 e tem tamanho 2. O mês é o pedaço da data que inicia na posição 3 e tem tamanho 2. O ano é o pedaço da data que inicia na posição 6 e tem tamanho 4. O exemplo da Figura 3.7 mostra o código em PHP, em que $dtNasc é a variável com a data a ser modificada.

```
$dia = substr($dtNasc,0,2);
$mes = substr($dtNasc,3,2);
$ano = substr($dtNasc,6,4);
```

Figura 3.7: Exemplo de separação dos campos de uma data
Fonte: Elaborada pelo autor

Em seguida, concatenam-se os campos no formato desejado, no nosso caso “ano-mês-dia”, como mostrada na Figura 3.8 a seguir.

```
$dtNasc = "$ano-$mes-$dia";
```

Figura 3.8: Concatenando a data no padrão do MySQL: “ano-mês-dia”
Fonte: Elaborada pelo autor

Podemos também verificar se essa é uma data válida. Para isso, usa-se o comando **checkdate**.

checkdate – retorna verdadeiro (*true*) se a data é válida e falso (*false*) caso contrário. Sua sintaxe é:

checkdate(mes, dia, ano)

Para testar nossa data, o exemplo ficaria conforme Figura 3.9 a seguir.

```
if ( ! checkdate($mes, $dia, $ano) ) {
    echo "DATA inválida. <BR>";
    $camposOK = false;
}
```

Figura 3.9: Função que verifica se uma data é válida
Fonte: Elaborada pelo autor

O programa da Figura 3.10 mostra todo o código da página **cadclientebd.php**, que recebe os dados do formulário, verifica se os campos estão corretos, converte e verifica a data e depois insere no banco de dados.

```
<html>
<head>
    <meta http-equiv="Content-Type"
        content="text/html; charset=UTF-8">
    <title>Cadastro de Clientes</title>
</head>
<body>
    <?php
        // Recebe cada campo do formulário
        // e coloca em uma variável.
        $nome = $_POST["txtNome"];
        $ender = $_POST["txtEndereco"];
        $cpf = $_POST["txtCPF"];
        $estado = $_POST["listEstados"];
        $dtNasc = $_POST["txtData"];
        $sexo = $_POST["sexo"];
        $cinema = $_POST["checkCinema"];
        $musica = $_POST["checkMusica"];
        $info = $_POST["checkInfo"];
        $login = $_POST["txtLogin"];
        $senha1 = $_POST["txtSenha1"];
        $senha2 = $_POST["txtSenha2"];

        // Verificar campos
        $camposOK = true;  // Determina se ocorreu erro
        if ( $nome == "" ) {
            echo "Informe o NOME. <BR>";
            $camposOK = false;
        }
        if ( $ender == "" ) {
            echo "Informe o ENDEREÇO. <BR>";
            $camposOK = false;
        }
        if ( $login == "" ) {
            echo "Informe o LOGIN. <BR>";
            $camposOK = false;
        }
        // Verificar se as SENHAS conferem
        if ( $senha1 != $senha2 ) {
            echo "As SENHAS não conferem!. <BR>";
            $camposOK = false;
        }

        // *** Acrescentar as validações de CPF

        // #### Codigo para converter DATA de
        // "dd/mm/aaaa" para "aaaa-mm-dd"
        if ( strlen($dtNasc) == 10 ) {
            $dia = substr($dtNasc,0,2);
            $mes = substr($dtNasc,3,2);
            $ano = substr($dtNasc,6,4);
            $dtNasc = "$ano-$mes-$dia";
            if ( ! checkdate($mes, $dia, $ano) ) {
                echo "DATA inválida. <BR>";
                $camposOK = false;
            }
        } else {
            echo "DATA inválida. <BR>";
            $camposOK = false;
        }
```

```php
// Verificar se as SENHAS conferem
if ( $senha1 != $senha2 ) {
    echo "As SENHAS não conferem!. <BR>";
    $camposOK = false;
}

// *** Acrescentar as validações de CPF

// #### Codigo para converter DATA de
// "dd/mm/aaaa" para "aaaa-mm-dd"
if ( strlen($dtNasc) == 10 ) {
    $dia = substr($dtNasc,0,2);
    $mes = substr($dtNasc,3,2);
    $ano = substr($dtNasc,6,4);
    $dtNasc = "$ano-$mes-$dia";
    if ( ! checkdate($mes, $dia, $ano) ) {
        echo "DATA inválida. <BR>";
        $camposOK = false;
    }
} else {
    echo "DATA inválida. <BR>";
    $camposOK = false;
}
```

Figura 3.10: Página em PHP que insere os dados do cliente no banco
Fonte: Elaborada pelo autor

Crie a tabela **Produtos** no banco de dados MySQL com os seguintes campos:

idProduto – inteiro autoincremento (**chave primária**)

descrição – varchar(100)

preço – float

qtdeEstoque – inteiro

dataValidade – date

Crie o formulário para entrar com esses dados e a página em PHP para receber, validar e inserir os dados no banco de dados.

3.4 Listando os dados

Após inserir os dados no banco, temos a possibilidade de recuperá-los e mostrá-los para o usuário. O comando em SQL que faz isso é o SELECT.

O comando em PHP para recuperar os dados é o mesmo usado no inserir, o **mysql_query**. O que muda é o comando SQL passado para o banco, que agora é o SELECT.

O retorno de um SELECT no banco é um conjunto de registros. Precisamos percorrer todos esses registros, pegando o primeiro, passando para o próximo, e assim por diante até ao último. O comando em PHP que faz isso é o **mysql_fetch_assoc**.

mysql_fetch_assoc – retorna um registro de uma consulta e aponta para o próximo registro. Retorna *false* quando não existir mais registros, ou seja, quando chegar ao último. Retorna o registro em forma de *array*, em que os índices são os nomes das colunas da tabela no banco de dados. Sua sintaxe é:

mysql_fetch_assoc(identificador)

em que: **identificador** é o resultado do SELECT, ou seja, o conjunto de registros.

Como o **mysql_fetch_assoc** retorna um registro por vez, e para quando chegar ao último (*false*), podemos colocá-lo como condição de parada em um comando ***while***, para percorrer todos os registros.

O registro retornado a cada laço do ***while*** é guardado em uma variável do tipo ***array***. Cada posição do ***array*** é um campo da tabela. Para pegar o valor, usa-se como índice do ***array*** o nome do campo definido na tabela do banco, desta forma:

$variavel_registro[“nome_campo”]

O nome do campo deve ser o mesmo usado no banco de dados. Letras maiúsculas são diferentes de minúsculas.

Para saber o total de registros retornados por uma consulta ao banco, usamos o comando **mysql_num_rows**.

mysql_num_rows – retorna a quantidade de registros da última consulta ao banco. Sua sintaxe é:

mysql_num_rows(identificador);

em que: **identificador** é o resultado do SELECT, ou seja, o conjunto de registros.

O programa da Figura 3.11 mostra o exemplo de uma página que consulta todos os dados da tabela **Clientes,** do nosso banco de dados de teste. Para cada registro obtido, uma linha de uma tabela em HTML é criada para mostrar os resultados. Perceba que dentro do ***while*** ficam apenas os comandos para criar uma linha da tabela (<TR>). Os comandos para abrir e fechar a tabela ficam antes e depois do ***while***, respectivamente (<TABLE> e </TABLE>). A primeira linha da tabela, com os nomes dos campos, também fica fora do ***while***, porque não pode ser repetida.

```
<html>
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Pesquisar</title>
</head>
<body>
<?php

    include_once("conexaobd.php");
    $sql = "SELECT * FROM Clientes";

    $res = mysql_query( $sql ) or die ("ERRO ao pesquisar dados do
cliente. " . mysql_error() );

    $total = mysql_num_rows($res);
    echo "<H3>O total de clientes é: $total </H3>";

    echo "<TABLE border=1>";
    echo "<TR><TD>CÓDIGO</TD><TD>NOME</TD><TD>CPF</TD>
          <TD>ENDEREÇO</TD><TD>ESTADO</TD>
          <TD>DATA NASC</TD><TD>SEXO</TD><TD>LOGIN</TD>
          <TD>CINEMA</TD><TD>MÚSICA</TD>
          <TD>INFORMÁTICA</TD></TR>";

    while (  $registro = mysql_fetch_assoc($res)  ) {
       $id     = $registro["idClientes"];
       $nome   = $registro["nome"];
       $ender  = $registro["endereco"];
       $cpf    = $registro["cpf"];
       $estado = $registro["Estados_sigla"];
       $login  = $registro["login"];
       $dtNasc = $registro["dtNasc"];
       $sexo   = $registro["sexo"];
       $cinema = $registro["cinema"];
       $musica = $registro["musica"];
       $info   = $registro["info"];

       echo "<TR><TD>$id</TD><TD>$nome</TD><TD>$cpf</TD>
         <TD>$ender</TD><TD>$estado</TD><TD>$dtNasc</TD>
         <TD>$sexo</TD><TD>$login</TD><TD>$cinema</TD>
         <TD>$musica</TD><TD>$info</TD></TR>";
    }
    echo "</TABLE>";
?>
</body>
</html>
```

Figura 3.11: Página em PHP que exibe todos os clientes
Fonte: Elaborada pelo autor

3.4.1 Formulário dinâmico

O procedimento de listar todos os registros pode ser útil quando queremos preencher algum objeto da página com os dados da tabela, como por exemplo, preencher as opções de um combobox a partir do banco de dados. Nesse caso, o formulário é dinâmico, porque qualquer alteração no banco de dados, automaticamente, é refletida no formulário. Situações nas quais isso pode ocorrer são inúmeras, como: preencher um combobox com todos os cursos cadastrados no banco; ou exibir as opções cadastradas no banco de dados para que um cliente possa escolher a cor de um produto; ou selecionar uma das cidades cadastradas para realizar a prova de um concurso; e assim por diante.

Todos esses exemplos podem ser obtidos com o procedimento do programa da Figura 3.11. Dentro do *while* é que ficam os comandos HTML para cada opção desejada.

Como exemplo, vamos mostrar como preencher o combobox do campo **listEstado** do nosso formulário da Figura 2.2, da Aula 2. Naquele formulário, as opções foram definidas no próprio código HTML. Vamos agora ler os estados a partir do nosso banco de dados.

Utilizaremos o programa da Figura 3.11 dentro do formulário da Figura 2.2, da Aula 2, na região onde é montado o combobox **listEstados**. Nosso SELECT será na tabela **Estados** (veja Figura 3.1). Dentro do ***while*** vamos colocar os comandos para criar as opções do combobox, que é o <OPTION>. Para cada registro da tabela será criado um comando <OPTION>, mostrando o nome do estado. Para o atributo ***value***, será usada a sigla do estado. Quando o usuário escolhe uma opção de um combobox, apenas o atributo ***value*** é enviado para o servidor; em nosso caso, somente a sigla do estado será enviada. A Figura 3.12 mostra o trecho de programa alterado do formulário para cadastro.

```
. . .
<tr>
  <th>Estado</th>
  <td>
    <select name="listEstados" >
    <?php
       include_once("conexaobd.php");
       $sql = "SELECT * FROM Estados";
       $res = mysql_query( $sql ) or die ("ERRO ao pesquisar ESTADOS.
" . mysql_error() );

       while (  $registro = mysql_fetch_assoc($res)  ) {
          $sigla = $registro["sigla"];
          $nome  = $registro["nome"];
          echo "<OPTION value='$sigla'>$nome</OPTION>";
       }
    ?>
    </select>
  </td>
</tr>
. . .
```

Figura 3.12: Montando um formulário dinâmico
Fonte: Elaborada pelo autor

Toda página que contiver códigos em PHP deve ter a extensão **.PHP**. Portanto, o formulário de cadastro de clientes que antes era apenas HTML, deve ser renomeado para a extensão PHP.

Antes de testar, você deve preencher a tabela Estados com algumas informações.

Com o conhecimento adquirido até agora, você é capaz de criar *sites* que armazenam dados em um banco de dados. Você também tem a capacidade de recuperar as informações do banco de dados e mostrá-las para o usuário. A grande maioria dos *sites* utiliza algum banco de dados para manipular as informações. Vimos um exemplo com o banco de dados MySQL, mas você pode trabalhar com qualquer banco de dados.

Essa é a base de toda a interação de páginas *web* com o banco de dados. Portanto, pratique! Teste os exemplos apresentados e faça os exercícios propostos. A próxima aula dependerá muito destes conceitos. Somente passe para a próxima aula quando não existir mais nenhuma dúvida.

Resumo

Nesta aula aprendemos a conectar com o banco de dados e a realizar as operações para inserir e listar dados. Para realizar essas operações, vimos exemplos de páginas *web* dinâmicas com os comandos PHP. Na próxima aula continuaremos trabalhando com bancos de dados. Aprenderemos a alterar e excluir dados do banco.

Atividades de aprendizagem

1. Crie uma página em PHP para listar todos os produtos da tabela criada na terceira Atividade de aprendizagem (Atividade 3.1) desta aula.

2. Crie a tabela **Vendas** no banco de dados MySQL com os seguintes campos:

   idVenda – inteiro autoincremento (**chave primária**)

   idCliente – chave estrangeira da tabela Clientes

   idProduto – chave estrangeira da tabela Produtos

   qtdeVendida – inteiro

   preçoTotal – *float*

   dataVenda – *date*

   formaPagto – inteiro (pode ser chave estrangeira de outra tabela com as formas de pagamento)

   Crie o formulário para entrar com os dados de uma venda. Preencha um combobox com todos os clientes cadastrados e outro com os produtos. Lembre-se de colocar no atributo **value** o **id** do cliente e do produto. Os demais campos serão do tipo texto.

   Se a forma de pagamento estiver em uma tabela separada, coloque um combobox para ela também.

Depois crie uma página em PHP para receber os dados do formulário e inserir na tabela **Vendas**

Leituras complementares

DAVIS, Michele E.; PHILLIPS, Jon A. **Aprendendo PHP e MySQL**. Rio de Janeiro: Alta Books, 2008.

MILANI, André. **Construindo aplicações *web* com PHP e MySQL**. São Paulo: Novatec, 2010.

NIEDERAUER, Juliano. **Desenvolvendo websites com PHP**. 2ª ed. São Paulo: Novatec, 2004.

WELLING, Luke; THOMSON, Laura. **PHP e MySQL**: desenvolvimento *web*. 3ª ed. Rio de Janeiro: Elsevier, 2005.

Aula 4 – Consulta, exclusão e alteração no banco de dados MySQL

Objetivos

Aprender como realizar uma consulta filtrando por determinado campo.

Aprender a excluir e a alterar um registro no banco de dados a partir de páginas *web* em PHP.

Vamos continuar interagindo com o banco de dados. Como pré-requisito, é fundamental que vocês tenham assimilado o conteúdo da aula anterior; sem ele não será possível se conectar com o banco de dados.

4.1 Consultando no banco de dados MySQL

Em muitas situações, listar todas as informações de uma tabela do banco de dados não é uma boa opção. Tabelas que possuem grande quantidade de dados aumentam o tráfego da internet para trazer todas essas informações para a página.

Vamos aprender agora como filtrar os dados de um consulta. Podemos realizar uma consulta no banco de dados escolhendo qualquer um dos campos para pesquisar, ou uma combinação deles. Por exemplo, podemos listar todos os clientes que nasceram em um determinado mês, ou listar todos os funcionários com salário maior que um determinado valor, ou ainda, listar todas as vendas do último mês que ainda não foram pagas.

Para selecionar determinadas informações do banco de dados, usaremos a cláusula WHERE dentro do SELECT, como vocês já viram na disciplina de **Banco de Dados**. O que veremos aqui é como pegar os campos de um formulário para fornecer como limite para a consulta e exibir a resposta na mesma página.

Então, a nossa página terá o formulário em HTML para a consulta, e outra parte com o código em PHP para exibir o resultado, que pode ser em forma de tabela.

Como exemplo, nosso formulário terá um combobox para escolher o tipo de consulta, chamado **comboTipo**. As opções serão por nome ou por mês de aniversário. Terá também um campo para digitarmos o valor que queremos pesquisar, chamado **txtPesquisa**. Ao enviar os dados, esse formulário chamará a mesma página, para que o PHP possa exibir os dados da pesquisa. Portanto o atributo **action** do <FORM> apontará para o mesmo arquivo. O programa da Figura 4.1 mostra essa página, que chamaremos de **"pesqclientes.php"**.

```
<html>
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Pesquisar clientes</title>
</head>
<body>
 <H2>Pesquisar clientes por NOME ou MÊS</H2>
 <FORM action="pesqclientes.php" method="POST">
     <select name="comboTipo">
         <option value="0">Nome</option>
         <option value="1">Mês</option>
     </select>
     <input type="text" name="txtPesquisa">
     <input type="submit" name="btnPesq"
            value="Pesquisar">
 </FORM>
<?php
   // PHP para pesquisar no banco e
   // montar a tabela de resposta
   // quando o usuário clicar o botão
   // "Pesquisar" do formulário
?>
</body>
</html>
```

Figura 4.1: Programa de um formulário para pesquisar os clientes (pesqclientes.php)
Fonte: Elaborada pelo autor

Quando o usuário clicar no botão **Enviar**, os dados para pesquisar serão enviados para a mesma página, que agora executará o PHP na parte de baixo e mostrará o mesmo formulário, mais a tabela de resposta.

Porém, na primeira vez que chamarmos essa página, o PHP não pode ser executado. Ele somente será executado da segunda vez em diante, ou seja, quando ele receber os dados para a pesquisa. Para distinguir quando uma página está recebendo uma informação ou não, usaremos o comando **isset**.

isset – retorna verdadeiro (*true*) se uma certa variável já foi declarada, e falso (*false*), em caso contrário. Sua sintaxe é:

isset(variavel)

Na primeira vez que a página é chamada, não temos nenhuma variável. Na segunda vez em diante, a página recebe as variáveis **comboTipo** e **txtPesquisa** do formulário (via método POST ou GET). Podemos utilizar qualquer uma dessas duas variáveis para testar quando o PHP será executado.

Dentro do PHP, obtém-se o tipo de pesquisa que será feita, por nome ou por mês (**comboTipo**), e monta-se o SQL correspondente com o valor digitado em **txtPesquisa**. O resultado é mostrado em forma de tabela. O programa da Figura 4.2 mostra o código completo da página de pesquisa **pesqclientes.php.**

```
<html>
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Pesquisar clientes</title>
</head>
<body>
 <H2>Pesquisar clientes por NOME ou MÊS</H2>
 <FORM action="pesqclientes.php" method="POST">
     <select name="comboTipo">
         <option value="0">Nome</option>
         <option value="1">Mês</option>
     </select>
     <input type="text" name="txtPesquisa">
     <input type="submit" name="btnPesq"
            value="Pesquisar">
 </FORM>

<?php
  // PHP para pesquisar no banco
  if (  isset( $_POST["comboTipo"] )   ) {

     // Pegar os campos do formulário acima
     $tipo = $_POST["comboTipo"];
     $pesq = $_POST["txtPesquisa"];

     // Montar o SQL para pesquisar
     include_once("conexaobd.php");
     $sql = "SELECT * FROM Clientes ";
     if ( $tipo == 0 ){
        $sql = $sql . "WHERE nome LIKE '$pesq%'  ";
     } else if ( $tipo == 1 ){
        $sql = $sql . "WHERE MONTH(dtNasc) = '$pesq'";
     }

     echo "<H2>Resultado da pesquisa:</H2>";
     echo "<TABLE border=1>";
     echo "<TR><TD>CÓDIGO</TD><TD>NOME</TD><TD>CPF</TD>
           <TD>ENDEREÇO</TD><TD>ESTADO</TD>
           <TD>DATA NASC</TD><TD>SEXO</TD><TD>LOGIN</TD>
           <TD>CINEMA</TD><TD>MÚSICA</TD>
           <TD>INFORMÁTICA</TD></TR>";

     $res = mysql_query( $sql ) or die ("ERRO ao pesquisar dados do
cliente. " . mysql_error() );
     while (  $registro = mysql_fetch_assoc($res)  ) {
       $id     = $registro["idClientes"];
       $nome   = $registro["nome"];
       $ender  = $registro["endereco"];
       $cpf    = $registro["cpf"];
       $estado = $registro["Estados_sigla"];
       $login  = $registro["login"];
       $dtNasc = $registro["dtNasc"];
       $sexo   = $registro["sexo"];
       $cinema = $registro["cinema"];
       $musica = $registro["musica"];
       $info   = $registro["info"];

       echo "<TR><TD>$id</TD><TD>$nome</TD><TD>$cpf</TD>
         <TD>$ender</TD><TD>$estado</TD><TD>$dtNasc</TD>
         <TD>$sexo</TD><TD>$login</TD><TD>$cinema</TD>
         <TD>$musica</TD><TD>$info</TD></TR>";
     }
     echo "</TABLE>";
  } //fim-if
?>
</body>
</html>
```

Figura 4.2: Programa de uma página para pesquisar os clientes (pesqclientes.php)
Fonte: Elaborada pelo autor

Quando a pesquisa é com um campo do tipo *string*, como o nome do exemplo anterior, usa-se a palavra LIKE no lugar da igualdade, dentro da cláusula WHERE do SELECT. O símbolo **'%'** dentro do valor a ser pesquisado serve como curinga. Significa que pode vir qualquer caracter naquela posição. O exemplo anterior busca no banco de dados qualquer cliente que começa com o valor digitado pelo usuário.

O que muda de uma pesquisa para outra são os campos e as tabelas do banco de dados, como, por exemplo, pesquisar produtos, vendas, locações, reservas, etc. Portanto, as modificações no código para realizar outro tipo de pesquisa seriam alterar o SELECT e a forma de exibir os dados.

Crie uma página em PHP para **pesquisar os produtos** da tabela criada na Atividade de aprendizagem da aula 3. Você deve permitir pesquisar pela descrição, código e data de validade.

4.2 Excluindo no banco de dados MySQL

Na grande maioria dos sistemas, não se usa muito excluir informações do banco de dados, porque isso faz com que a empresa perca o histórico dos dados. O que se faz é colocar um campo na tabela que informa o estado da informação como, por exemplo: “ativo”, “inativo”, “em andamento”, “finalizado”, “aguardando resposta”, etc.

Em vez de excluir o cliente do banco de dados, a empresa pode apenas mudar seu estado para inativo, para que no futuro possa realizar alguma campanha para recuperar os clientes perdidos.

Mesmo não sendo muito utilizado, em algumas situações pode ser necessário excluir informações do banco. Então vamos aprender a fazer isso: para a operação de excluir do banco, mais uma vez o que muda é o comando SQL. Os comandos em PHP são os mesmos: abrir a conexão, selecionar o banco e executar o comando SQL.

Na maioria dos sistemas, ao excluir, é necessário conhecer pelo menos um dado da tabela. Dificilmente iremos excluir vários dados de uma vez. O ideal seria obter a informação que corresponde à chave primária da tabela, por ser uma informação única. Uma das formas de fazer isso é usar algum mecanismo de pesquisa para escolher a informação a ser pesquisada.

Em nosso exemplo, vamos utilizar a página já criada de pesquisa. Na tabela que exibe as informações, podemos colocar em cada linha uma palavra, uma imagem ou algum outro objeto que, quando clicado, chama a página em PHP para excluir. A Figura 4.3 mostra a página resultante da pesquisa com uma imagem que corresponde à operação de excluir.

Para facilitar a visualização da imagem, alguns campos da tabela de resultados foram omitidos. Para o seu teste, use todos os campos da tabela.

Essa imagem possui um *link* (tag <A> do HTML) que aponta para a página do excluir, chamada em nosso exemplo de **“excluircliente.php”**. Em cada imagem devemos passar o código da informação que será excluída. Usaremos o método GET para envio de informações para a outra página (vejam a seção 2.1). O dado que passaremos será o código do cliente. Essa variável será chamada de **“codigo”**. O programa da Figura 4.4 mostra a página **“pesqclientes.php”** alterada para adicionar na tabela de resultado, mais uma coluna, que corresponde a imagem para a operação excluir. Percebam que em cada linha da tabela, o *link* passará via GET, a variável **codigo**, que receberá o **id** de cada cliente no banco de dados.

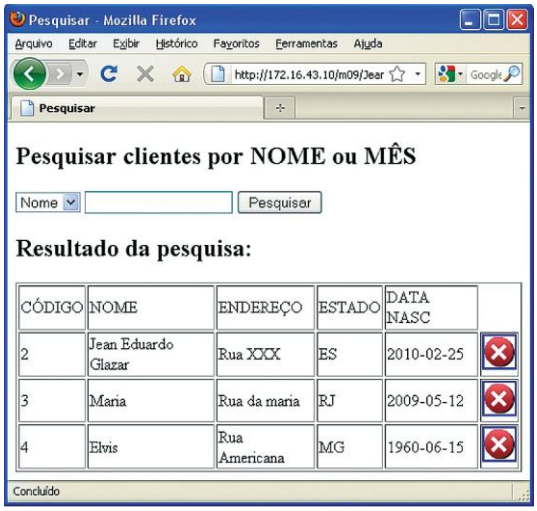


Figura 4.3: Exemplo de pesquisa com uma imagem para o excluir
Fonte: Página exibida pelo Mozilla Firefox 3.5

```
. . .
while (  $registro = mysql_fetch_assoc($res)  ) {
   $id     = $registro["idClientes"];
   $nome   = $registro["nome"];
   $ender  = $registro["endereco"];
   $cpf    = $registro["cpf"];
   $estado = $registro["Estados_sigla"];
   $login  = $registro["login"];
   $dtNasc = $registro["dtNasc"];
   $sexo   = $registro["sexo"];
   $cinema = $registro["cinema"];
   $musica = $registro["musica"];
   $info   = $registro["info"];

   echo "<TR><TD>$id</TD><TD>$nome</TD><TD>$cpf</TD>
         <TD>$ender</TD><TD>$estado</TD><TD>$dtNasc</TD>
         <TD>$sexo</TD><TD>$login</TD><TD>$cinema</TD>
         <TD>$musica</TD><TD>$info</TD>
         <TD><A href='excluircliente.php?codigo=$id'>
              <IMG src='excluir.png'></A></TD></TR>";
}
. . .
```

Figura 4.4: Resultado da pesquisa com *link* para o "excluircliente.php"
Fonte: Elaborada pelo autor

Altere e execute a página **"pesqclientes.php"**, para colocar o *link* para o excluir cliente, como mostrado acima. Ao passar o *mouse* em cima da imagem do excluir, mas sem clicar, olhe na barra de *status*, na parte de baixo do navegador, o *link* de cada imagem. Perceba que o que muda de um para o outro é o código do cliente.

A página **"excluircliente.php"** será a responsável por executar o comando SQL no banco de dados. Essa página recebe o código passado via GET, pelo *link* da imagem excluir, e executa o comando DELETE no banco, com a cláusula WHERE informando o **id** do cliente que será excluído. Ao final, a página é redirecionada novamente para a página de pesquisa (você poderia redirecionar para qualquer outra página do seu sistema). O programa da Figura 4.5 mostra o código PHP completo dessa página.

```
<?php
 if (  isset( $_GET["codigo"] )   ) {
    // Pega o codigo passado pelo link via GET
    $cod = $_GET["codigo"];

    // Montar o SQL para excluir
    include_once("conexaobd.php");
    $sql = "DELETE FROM Clientes WHERE idClientes=$cod";
    $res = mysql_query( $sql ) or die ("ERRO ao excluir dados do
cliente. " . mysql_error() );
 }
 header("Location:pesqclientes.php");
?>
```

Figura 4.5: Programa em PHP para excluir um cliente
Fonte: Elaborada pelo autor

Implemente a página **“excluircliente.php”**, como mostrada acima, e teste suas páginas.

4.3 Alterando no banco de dados MySQL

A operação de alterar dados no banco segue o mesmo princípio que as demais informações. Novamente o que muda é o comando SQL passado para o banco.

Porém, o que torna uma página *web* um pouco mais trabalhosa para alterar dados, mas nem tanto, é que devemos ter as informações que serão alteradas. Essas informações podem estar dispostas de diversas formas em uma página. O mecanismo mais fácil e apropriado para isso é o formulário em HTML. Então, ao escolher uma determinada informação para alterar, vamos chamar uma página com o formulário já preenchido com as informações anteriores. O usuário altera no formulário os dados que deseja e depois os envia para outra página em PHP que atualizará o banco de dados.

Antes de chamar o formulário com os dados, na maioria das vezes, é interessante usar algum mecanismo de busca, como uma página de pesquisa. Em nosso exemplo, vamos utilizar três passos para alterar os dados de um cliente:

Passo 1 – usar a página **“pesqclientes.php”** para pesquisar o cliente de quem queremos modificar os dados. Colocar um *link* em cada cliente para que, ao clicar, a página **“formaltcli.php”** seja chamada, enviando o código do cliente via GET;

Passo 2 – a página **“formaltcli.php”** exibirá um formulário já preenchido com os dados do cliente escolhido. Após alterar os dados clicaremos em um botão para enviá-los para a página **“alterarclibd.php”**;

Passo 3 – a página **“alterarclibd.php”** atualizará o banco de dados com as informações atuais oriundas do formulário. Ao final, será redirecionado para a página com o formulário, que exibirá a mensagem de sucesso ou erro.

A Figura 4.6 mostra o diagrama de navegação desse mecanismo.

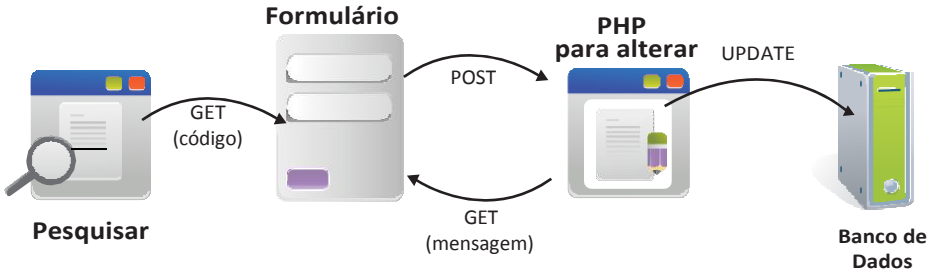


Figura 4.6: Diagrama de navegação das páginas para alterar dados
Fonte: Equipe produção CEAD/IFES (2011)

4.3.1 Passo 1 – *link* no pesquisar

Assim, como colocamos um *link* para a página que exclui os dados, vamos fazer de forma semelhante para a operação de alterar. Colocaremos um *link* no nome de cada cliente, que chamará a página com o formulário, passando o código do cliente via GET.

Poderíamos também usar uma imagem ou qualquer outro mecanismo para chamar o formulário, desde que passe o código do cliente que se deseja alterar.

A linha na página **"pesqclientes.php"** que será alterada é o comando *echo* que monta cada linha da tabela, dentro do *while*. Perceba que a coluna referente ao nome passa a ter um *link* para o formulário do alterar e a coluna referente à imagem do excluir continua com o *link* para a página de excluir. Ambos os *links* passam o código do cliente via método GET. A nova linha é mostrada na Figura 4.7 a seguir.

```
echo "<TR><TD>$id</TD>
  <TD><A href='formaltcli.php?codigo=$id'>$nome</A></TD>
  <TD>$cpf</TD><TD>$ender</TD><TD>$estado</TD>
  <TD>$dtNasc</TD><TD>$sexo</TD><TD>$login</TD>
  <TD>$cinema</TD><TD>$musica</TD><TD>$info</TD>
  <TD><A href='excluircliente.php?codigo=$id'>
      <IMG src='excluir.png'></A></TD></TR>";
```

Figura 4.7: Comando com o *link* no campo nome para chamar o alterar
Fonte: Elaborada pelo autor

4.3.2 Passo 2 – exibir o formulário

Na página "formaltcli.php", antes de exibir o formulário, é necessário obter os demais dados do banco, uma vez que temos somente o código do cliente passado como GET. Então, executaremos o SELECT para isso. Após, colocaremos os dados do banco em variáveis PHP que serão utilizadas para preencher o formulário.

O campo do tipo **data**, obtido no banco de dados, está no formato **"Y-m-d"** (padrão do MySQL). Então temos que convertê-lo para o nosso formato **"d/m/Y"**, para exibir no formulário. Usaremos para isso duas funções, a **strtotime** e a **date**.

strtotime – converte uma data no formato *string* para o formato *timestamp* (usado pelo PHP). Sua sintaxe é:

strtotime(string_data)

date – converte uma data no formato *timestamp* para algum formato em *string*. Sua sintaxe é:

date(formato, timestamp)

Portanto, para converter de **"Y-m-d"** para **"d/m/Y"** usaremos os seguintes comandos:

```
$varData = date("d/m/Y", strtotime($varData));
```

Em que: $varData é a data no formato **"Y-m-d"**.

Em cada campo do formulário colocaremos a variável PHP, correspondente ao dado obtido do banco. Como temos campos de tipos diferentes, cada um terá uma forma diferente de preencher os dados. Em todas elas, alternaremos entre códigos HTML e PHP.

a) Campos do tipo "text"

Coloca o atributo *value* e, entre aspas duplas, abre o PHP e pega a variável correspondente com o *echo*. Veja a Figura 4.8 desta forma:

```
<input type="text" name="txtNome"
       value="<?php echo $nome; ?>" >
```

Figura 4.8: Atribuindo valor para campos do tipo "text"
Fonte: Elaborada pelo autor

A expressão **<?php** echo $nome; **?>** abre o código em PHP, pega a variável **$nome** com o dado obtido do banco e coloca no atributo *value* do campo do formulário.

Todo o código PHP deve ficar dentro das aspas duplas do atributo *value*.

b) Campos do tipo "textarea"

O conteúdo desse campo fica entre as *tags* <TEXTAREA> e </TEXTAREA>. Então, colocaremos o código PHP com o valor a ser preenchido entre essas *tags*. Segue o exemplo na Figura 4.9 para o campo **endereço**.

```
<textarea name="txtEndereco" cols="30" rows="4">
   <?php echo $ender; ?>
</textarea>
```

Figura 4.9: Atribuindo valor para campos do tipo “textarea”
Fonte: Elaborada pelo autor

c) Campos do tipo “radio” e “checkbox”

Coloca-se a palavra ***checked*** na opção que ficará marcada. Portanto, é necessário testar com o comando *if* qual será essa opção. O exemplo na Figura 4.10 com o campo **sexo** ficaria assim:

```
<?php
   if ($sexo == 'M' ) {
      echo "<input type='radio' name='sexo'
            value='M' checked >Masculino<BR>";
      echo "<input type='radio' name='sexo'
            value='F'>Feminino<BR>";
   } else {
      echo "<input type='radio' name='sexo'
            value='M'>Masculino<BR>";
      echo "<input type='radio' name='sexo'
            value='F' checked >Feminino<BR>";
   }
?>
```

Figura 4.10: Atribuindo valor para campos do tipo “radio” e “checkbox”
Fonte: Elaborada pelo autor

d) Campos do tipo “combobox” (<SELECT>)

Nesses campos, a opção marcada recebe o atributo ***selected***. Então, uma das soluções seria semelhante a dos campos do tipo **“radio”**. Testa com o *if* e coloca o atributo ***selected*** na opção correspondente. Porém, muitas vezes, as opções de uma caixa de seleção são obtidas do banco de dados. Nesse caso, no PHP que obtém as opções do banco, coloca-se o *if* para testar qual opção será a selecionada. Se for a que queremos que fique marcada, então o comando <OPTION> fica com o atributo ***selected***, como o exemplo na Figura 4.11, com o campo que lista os estados.

```
<select name="listEstados" >
<?php
   include_once("conexaobd.php");
   $sql = "SELECT * FROM Estados";

   $res = mysql_query( $sql ) or die ("ERRO ao pesquisar ESTADOS. " .
mysql_error() );

   while (  $registro = mysql_fetch_assoc($res)  ) {
      $sigla = $registro["sigla"];
      $nome  = $registro["nome"];

      if ($sigla == $estado )
         echo "<OPTION value='$sigla' selected >
               $nome</OPTION>";
      else
         echo "<OPTION value='$sigla'>$nome</OPTION>";
   }
?>
</select>
```

Figura 4.11: Atribuindo valor para campos do tipo “combobox”
Fonte: Elaborada pelo autor

A variável **$estado** é obtida da tabela de **Clientes**, ou seja, é o estado que deverá ficar marcado. A variável **$sigla** é obtida de cada registro da tabela com todos os estados.

A partir do formulário preenchido, o usuário poderá alterar os dados que lhe convêm. Ao clicar no botão para enviar os dados, o formulário enviará as novas informações para a página **“alterarclibd.php”**, que realizará o UPDATE

no banco. Portanto, a declaração do formulário (<FORM>) ficará como no exemplo na Figura 4.12 a seguir.

```
<form method="post" name="formCadastro"
      action="alterarclibd.php" >
  . . .
</form>
```

Figura 4.12: Definição do action para a página "alterarclibd.php"
Fonte: Elaborada pelo autor

O valor da chave primária de uma tabela não pode ser alterado, porque pode causar inconsistência nos dados. Porém, ela é fundamental para o processo de alterar, porque será por meio dela que o sistema identificará o registro a ser alterado.

Em nosso exemplo, a chave primária é o código do cliente. Então colocaremos um campo no formulário para esse código. Para que o usuário não possa alterar esse valor, esse campo ficará oculto. Para criar um campo oculto, basta usar o atributo ***type*** com o valor ***hidden***, como no exemplo na Figura 4.13 a seguir.

```
<input type="hidden" name="txtCodigo"
       value="<?php echo $id; ?>" >
```

Figura 4.13: Campo oculto (*hidden*) com o código do cliente
Fonte: Elaborada pelo autor

e) Mensagem de erro

Os dados serão enviados para a página PHP que atualizará o banco de dados. Para saber se ocorreu erro ou não, redirecionaremos para a página com o formulário novamente, informando a mensagem de erro ou sucesso. Essa mensagem será enviada via o método GET, com o nome **mensagem**. Então, em nossa página com o formulário, colocaremos o código PHP abaixo para pegar a mensagem, caso tenha, e exibir junto com os dados. A Figura 4.14 mostra esta programação.

```
. . .
<?php
    // Exibir a mensagem de erro ou sucesso
    if ( isset($_GET["mensagem"]) ) {
      $msg = $_GET["mensagem"];
      echo "<FONT color='red'>$msg</FONT>";
    }
?>
. . .
```

Figura 4.14: Exibindo a mensagem de resposta
Fonte: Elaborada pelo autor

O programa da Figura 4.15 mostra o código completo da página **"formaltcli.php"**.

```
<html>
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Alterar dados do Clientes</title>
</head>
<body>

<?php
    // Consultar no banco pelo código do cliente
    // Preencher o formulário com os valores
    // Pega o código do cliente passado como GET
    $cod = $_GET["codigo"];

    include_once("conexaobd.php");
    $sql=" SELECT * FROM Clientes WHERE idClientes=$cod";
    $res = mysql_query( $sql ) or die ("ERRO ao pesquisar dados do
cliente. " . mysql_error() );

    if (  $registro = mysql_fetch_assoc($res)  ) {
       $id     = $registro["idClientes"];
       $nome   = $registro["nome"];
       $ender  = $registro["endereco"];
       $cpf    = $registro["cpf"];
       $estado = $registro["Estados_sigla"];
       $login  = $registro["login"];
       $dtNasc = $registro["dtNasc"];
       $sexo   = $registro["sexo"];
       $cinema = $registro["cinema"];
       $musica = $registro["musica"];
       $info   = $registro["info"];

       // #### Codigo para converter DATA
       // de "aaaa-mm-dd" para "dd/mm/aaaa"
       $dtNasc = date("d/m/Y", strtotime($dtNasc));
    }
?>

<form method="post" name="formCadastro"
      action="alterarclibd.php" >
  <input name="txtCodigo" type="hidden"
         value="<?php echo $id; ?>">

  <H1>Cadastro de clientes</H1>

  <?php
     // Exibir a mensagem de erro ou sucesso
     if ( isset($_GET["mensagem"]) ) {
       $msg = $_GET["mensagem"];
       echo "<FONT color='red'>$msg</FONT>";
     }
  ?>


<form method="post" name="formCadastro"
      action="alterarclibd.php" >
  <input name="txtCodigo" type="hidden"
         value="<?php echo $id; ?>">

  <H1>Cadastro de clientes</H1>

  <?php
     // Exibir a mensagem de erro ou sucesso
     if ( isset($_GET["mensagem"]) ) {
       $msg = $_GET["mensagem"];
       echo "<FONT color='red'>$msg</FONT>";
     }
  ?>

  <table width="100%">
    <tr>
      <th width="18%"> Nome</th>
      <td width="82%"><input type="text" name="txtNome"
                     value="<?php echo $nome; ?>" >
      </td>
    </tr>
    <tr>
    <th>CPF</th>
    <td><input name="txtCPF" type="text" maxlength="14"
               value="<?php echo $cpf; ?>">
    </td>
    </tr>
    <tr>
      <th>Endereço</th>
    <td><textarea name="txtEndereco" cols="30" rows="4">
            <?php echo $ender; ?>
        </textarea>    </td>
    </tr>
    <tr>
    <th>Estado</th>
    <td><select name="listEstados" >
    <?php
      $sql = "SELECT * FROM Estados";

      $res = mysql_query( $sql ) or die ("ERRO ao pesquisar ESTADOS.
" . mysql_error() );

      while ( $registro = mysql_fetch_assoc($res) ) {
        $sigla = $registro["sigla"];
        $nome  = $registro["nome"];

        if ($sigla == $estado )
          echo "<OPTION value='$sigla' selected>
                $nome</OPTION>";
        else
         echo "<OPTION value='$sigla'>$nome</OPTION>";
      }
    ?>
    </select>
    </td>
  </tr>
  <tr>
     <th>Data Nasc.</th>
    <td><input name="txtData" type="text"
        value="<?php echo $dtNasc; ?>" >
    </td>
  </tr>
  <tr>
    <th>Sexo</th>
    <td>
```

```
        <?php
          if ($sexo == 'M' ) {
             echo "<input type='radio' name='sexo'
                    value='M' checked >Masculino<BR>";
             echo "<input type='radio' name='sexo'
                    value='F'>Feminino<BR>";
          } else {
             echo "<input type='radio' name='sexo'
                    value='M'>Masculino<BR>";
             echo "<input type='radio' name='sexo'
                    value='F' checked >Feminino<BR>";
          }
        ?>
        </select>
        </td>
      </tr>
      <tr>
         <th>Data Nasc.</th>
         <td><input name="txtData" type="text"
             value="<?php echo $dtNasc; ?>" >
         </td>
      </tr>
      <tr>
        <th>Sexo</th>
        <td>

          <?php
            if ($sexo == 'M' ) {
               echo "<input type='radio' name='sexo'
                      value='M' checked >Masculino<BR>";
               echo "<input type='radio' name='sexo'
                      value='F'>Feminino<BR>";
            } else {
               echo "<input type='radio' name='sexo'
                      value='M'>Masculino<BR>";
               echo "<input type='radio' name='sexo'
                      value='F' checked >Feminino<BR>";
            }
          ?>
        </td>
      </tr>
      <tr>
        <th>Áreas de Interesse</th>
        <td>
          <?php
            if ( $cinema == 1 )
               echo "<input name='checkCinema' type='checkbox'
value='true' checked/>Cinema<BR>";
            else
               echo "<input name='checkCinema' type='checkbox'
value='true'/>Cinema<BR>";
            if ( $musica == 1 )
               echo "<input name='checkMusica' type='checkbox'
value='true' checked/>Musica<BR>";
            else
               echo "<input name='checkMusica' type='checkbox'
value='true'/>Musica<BR>";
            if ( $info == 1 )
               echo "<input name='checkInfo' type='checkbox'
value='true' checked/>Info<BR>";
            else
               echo "<input name='checkInfo' type='checkbox'
value='true'/>Info<BR>";
          ?>
        </td>
      <tr>
         <th>Login</th>
         <td><input name="txtLogin" type="text"
                value="<?php echo $login; ?> ">
         </td>
      </tr>
      </tr>
      <tr>
         <th>Senha</th>
         <td><input name="txtSenha1" type="password"></td>
      </tr>
      <tr>
         <th>Confirmação Senha</th>
         <td><input name="txtSenha2" type="password"></td>
      </tr>
      <tr>
        <td>
     <input type="submit" name="btnEnviar" value="Enviar">
        </td>
        <td>
     <input type="reset" name="btnLimpar" value="Limpar">
        </td>
      </tr>
    </table>
</form>
</body>
</html>
```

Figura 4.15: Página com o formulário preenchido (“formaltcli.php”)
Fonte: Elaborada pelo autor

4.3.3 Passo 3 – atualizar o banco

A página **“alterarclibd.php”** receberá os novos dados do formulário do programa da Figura 4.15, via POST, e atualizará o banco de dados com o comando UPDATE. O programa da Figura 4.16 mostra essa página.

A página para atualizar o banco é semelhante à de inserir. A diferença é que é utilizado o comando UPDATE, ao invés de INSERT. Também foram tirados os comandos *echo* e colocadas as mensagens em uma única variável, chamada **$mensagem**, que será enviada para a página anterior mostrá-la.

```php
<?php
   // Recebe cada campo do formulário
   // e coloca em uma variável.
   $cod = $_POST["txtCodigo"];
   $nome = $_POST["txtNome"];
   $ender = $_POST["txtEndereco"];
   $cpf = $_POST["txtCPF"];
   $estado = $_POST["listEstados"];
   $dtNasc = $_POST["txtData"];
   $sexo = $_POST["sexo"];
   $cinema = $_POST["checkCinema"];
   $musica = $_POST["checkMusica"];
   $info = $_POST["checkInfo"];
   $login = $_POST["txtLogin"];
   $senha1 = $_POST["txtSenha1"];
   $senha2 = $_POST["txtSenha2"];

   // Verificar campos
   $camposOK = true;  // Determina se ocorreu erro
   $mensagem = "";    // Armazena a mensagem de erro
   if ( $nome == "" ) {
      $mensagem = $mensagem . "Informe o NOME. <BR>";
      $camposOK = false;
   }
   if ( $ender == "" ) {
      $mensagem = $mensagem . "Informe o ENDEREÇO.<BR>";
      $camposOK = false;
   }
   if ( $login == "" ) {
      $mensagem = $mensagem . "Informe o LOGIN. <BR>";
      $camposOK = false;
   }
   // Verificar se as SENHAS conferem
   if ( $senha1 == "" ) {
     $mensagem = $mensagem . "SENHAS não conferem!<BR>";
     $camposOK = false;
   }
```

Figura 4.16: Página que atualiza o banco de dados (“alterarclibd.php”)
Fonte: Elaborada pelo autor

Após realizar uma alteração no banco de dados, como por exemplo, com o comando UPDATE, é possível obter quantos registros foram alterados. Para isso basta utilizar a função **mysql_affected_rows**.

mysql_affected_rows – retorna a quantidade de registros que foram afetados por um comando SQL, como por exemplo o UPDATE. Sua sintaxe é:

mysql_affected_rows(conexao)

em que: **conexão** é um parâmetro opcional que indica a conexão com o banco de dados. Se não for informado, utiliza a última conexão aberta.

Com isso, concluímos o assunto sobre banco de dados. Vimos que o processo de acesso e manipulação de dados independe da operação a ser feita. O que define a operação que faremos com o banco de dados é o comando em SQL, como o INSERT, SELECT, UPDATE, DELETE e outros mais. Pronto! Agora você é capaz de criar um *site* completo, armazenando e recuperando dados do banco de dados.

Resumo

Nesta aula aprendemos a realizar uma consulta filtrando por determinado campo, a excluir e a alterar um registro no banco de dados a partir de páginas *web* em PHP. Na próxima aula aprenderemos a trabalhar com sessões, o que nos permitirá, por exemplo, implementar a validação de usuário no sistema, bem como trocar dados entre as páginas de uma sessão.

Encontra-se a seguir a Tabela 4.1 com o resumo dos comandos aprendidos até agora para manipulação do PHP com o banco de dados MySQL.

Tabela 4.1: Resumo dos principais comandos

FUNÇÃO	DESCRIÇÃO	PARÂMETROS
mysql_connect	Abre a conexão com o banco de dados.	IP do servidor onde está o banco de dados. Usuário e senha com permissão de acesso ao banco de dados.
mysql_select_db	Seleciona o banco a ser utilizado.	Nome do banco.
mysql_close	Fecha a conexão com o banco de dados.	Conexão criada anteriormente.
mysql_query	Função que executa um comando SQL no banco de dados MySQL. Retorna verdadeiro (*true*) em caso de sucesso e falso (*false*) caso contrário.	Comando na sintaxe SQL.
mysql_insert_id	Retorna o último número inserido de um campo do tipo "autoincremento"	Parâmetro opcional que indica a conexão com o banco de dados.
mysql_fetch_assoc	Retorna um registro de uma consulta, em forma de *array*, e aponta para o próximo registro. Retorna *false* quando chegar ao último.	Conjunto de registros resultante de uma consulta.
mysql_num_rows	Retorna a quantidade de registros da última consulta ao banco.	Conjunto de registros resultante de uma consulta.
mysql_affected_rows	Retorna a quantidade de registros que foram afetados por um comando SQL.	Parâmetro opcional que indica a conexão.
mysql_error	Retorna uma mensagem caso ocorra algum erro no acesso ao banco de dados	
isset	Retorna verdadeiro (*true*) se uma certa variável já foi declarada, e falso (*false*), em caso contrário	Variável a ser verificada.
include e include_once	Insere pedaços de códigos PHP de um determinado arquivo na página atual	Nome do arquivo PHP.
die	Função que exibe uma mensagem e interrompe a execução da página.	Mensagem a ser exibida.
substr	Retorna uma parte de uma *string*	*String* original. Posição inicial. Tamanho da *substring*.

continua

checkdate	Retorna verdadeiro (*true*) se a data é válida, e falso (*false*), em caso contrário	Mês, dia e ano.
strtotime	Converte uma data no formato *string* para o formato *timestamp*.	Data no formato *string* (padrão americano: Y-m-d).
date	Converte uma data no formato *timestamp* para algum formato em *string*.	Formato final da data. Data original do tipo *timestamp*.

conclusão

Fonte: Adaptado de Soares (2007)

Atividades de aprendizagem

1. Crie uma página em PHP para **pesquisar as vendas** da tabela criada na segunda atividade dentre as Atividades de aprendizagem propostas ao final da Aula 3. Você deve permitir pesquisar todas as vendas de um determinado cliente e todas as vendas de um determinado produto.

Obs.: Como na tabela **Vendas** tem apenas o código do cliente e do produto, será necessário pesquisar pelo nome usando as tabelas **Clientes** e **Produtos**, ou seja, no SELECT será preciso juntar (JOIN) a tabela **Vendas** com as tabelas **Clientes** e **Produtos**, de acordo com o tipo da pesquisa. Veja a apostila da disciplina de **Bancos de Dados** sobre como realizar a pesquisa unindo mais de uma tabela.

2. Implemente uma página para excluir os produtos da tabela criada na terceira Atividade de aprendizagem da Aula 3. Altere a página do pesquisar produtos, criada na primeira Atividade de aprendizagem desta Aula, e implemente o PHP para excluir o produto do banco.

3. Implemente as páginas para alterar os dados do cliente, como mostrada acima e teste.

4. Faça o mesmo procedimento para alterar os dados dos produtos. Altere a página do pesquisar produtos, criada na primeira Atividade de aprendizagem desta Aula, para colocar o *link* para a página de formulário com os dados do produto preenchidos. Implemente o PHP para atualizar o banco.

Leituras complementares:
DAVIS, Michele E.; PHILLIPS, Jon A. **Aprendendo PHP e MySQL**. Rio de Janeiro: Alta Books, 2008.
MILANI, André. **Construindo aplicações *web* com PHP e MySQL**. São Paulo: Novatec, 2010.
NIEDERAUER, Juliano. **Desenvolvendo *websites* com PHP**. 2ª ed. São Paulo: Novatec, 2004.
WELLING, Luke; THOMSON, Laura. **PHP e MySQL:** desenvolvimento *web*. 3 ª ed. Rio de Janeiro: Elsevier, 2005.

Aula 5 – Gerenciando sessões

Objetivos

Aprender para que serve uma sessão em um sistema *web* e como programá-la.

Aprender a como utilizar a sessão para autenticar usuários.

Construir páginas *web* para autenticar os usuários.

Esta Aula é voltada para o gerenciamento de sessões em sistemas *web*. O uso de sessões permitirá que informações sejam trocadas entre páginas *web* de uma mesma **sessão**. Com isso, será possível fazer a validação do usuário em uma página e verificar em todas as outras se o usuário foi autenticado ou não. E ainda mais, de posse do tipo do usuário (administrador, gerente, caixa, operador, etc.), é possível fazer controle de acesso, programando em cada página os tipos de usuários que podem acessá-las.

A cada página visitada por um usuário, uma nova conexão é criada pelo HTTP do navegador ao servidor. As informações da conexão anterior não são mantidas. Entretanto, em diversas situações é desejável que certas informações sejam armazenadas temporariamente, entre uma página e outra.

Como exemplo, pode-se citar a autenticação de *login*, que é feita em uma página e em todas as outras é necessário verificar se o usuário está logado, além de permitir obter os dados desse usuário em todas as páginas.

Para testar o uso de sessões, vamos utilizar o ***login*** e **senha** cadastrados na tabela de **Clientes**, como o modelo de dados da Figura 3.1, da Aula 3.

Outro exemplo é o carrinho de compras. Enquanto o usuário visita várias páginas, o carrinho de compras armazena os produtos comprados.

A-Z

Sessão
Uma sessão é basicamente um meio de mantermos dados durante a navegação por várias páginas de um *site*. Quando uma sessão é aberta, ela recebe um identificador único, o que permite ao PHP recuperar os dados vinculados àquela sessão.

5.1 Criando uma sessão

Para criar uma sessão, usa-se o comando:

```
session_start();
```

Após esse comando, um *array*, chamado $_**SESSION**, é criado para manipularmos as informações armazenadas. Com esse *array* é possível incluir uma nova variável, alterar as existentes e excluir uma ou todas as variáveis.

As páginas em que for preciso acessar informações do *array* $_SESSION, deve-se usar também o comando **session_start(),** para acessar a sessão criada.

5.2 Manipulando as variáveis de uma sessão

Os dados armazenados em uma sessão são armazenados no *array* **$_SESSION**. O índice desse *array*, entre colchetes, é o nome da variável de sessão.

Para criar uma nova variável na sessão, basta colocar seu nome como índice do *array* $_SESSION. O exemplo da Figura 5.1 mostra a criação da sessão e das variáveis ***login***, **tipo** e **nome**, obtidas do banco de dados, depois de validar o ***login*** 3 e a **senha** do usuário.

```
<?php
   // Validar login e senha no banco e
   // obter demais informações do cliente
   . . .
   session_start();
   $_SESSION["login"] = $login_banco;
   $_SESSION["nome"] = $nome_banco;
   $_SESSION["tipo"] = $tipo_banco;
   . . .
?>
```

Figura 5.1: Criação da sessão e suas variáveis
Fonte: Elaborada pelo autor

O procedimento para acessar uma variável já presente no <IT>array<IT> $_SESSION é o mesmo. Em nosso exemplo acima, poderíamos obter o nome e o *login* do usuário em qualquer página. Para eliminar uma variável de uma sessão, utiliza-se o comando:

```
unset($_SESSION[“nome_da_variável”])
```

5.3 Excluindo a sessão

Uma sessão é automaticamente eliminada quando o navegador do usuário é fechado. Caso queira destruir a sessão antes disso, como por exemplo, ao clicar em um botão do tipo "sair" ou "*logout*", devem ser realizadas três etapas. A primeira é acessar a sessão com o comando **session_start()**. A segunda é a liberação de todas as variáveis da sessão com o comando **session_unset()**. A última é a destruição da sessão com o comando:

```
session_destroy();
```

O programa da Figura 5.2 mostra o exemplo de um código PHP chamado a partir de um botão "sair" ou "*logout*", para finalizar a sessão. Após encerrar a sessão, a página é redirecionada para o formulário inicial de *login*.

```
<?php
    session_start();
    session_unset();
    session_destroy();
    header("Location:formlogin.php");
?>
```

Figura 5.2: Página que finaliza uma sessão ("logout.php")
Fonte: Elaborada pelo autor

5.4 Caso de uso: autenticando usuários

Para testarmos o uso de sessões, vamos criar as páginas para fazer a autenticação do usuário. No nosso banco de dados de exemplo, na tabela **Clientes**, já temos o ***login*** e a **senha** dos usuários. Vamos utilizar essa tabela para validar o usuário.

5.4.1 Formulário de *login*

Vamos criar um formulário, chamado **"formlogin.php"**, com os campos de ***login*** e **senha**. Esse formulário enviará os dados para o PHP **"login.php"**, via POST, que consultará o banco de dados verificando se o *login* e a senha estão corretos. Se estiverem corretos, então cria a sessão, coloca as variáveis ***login*** e **nome**,e redireciona para a página inicial do sistema. Se o *login* e a senha não conferem, então redireciona para o formulário de *login*, passando uma mensagem de erro via GET, isso para que no próprio formulário de *login* o usuário possa ver a mensagem de erro e digitar novamente o *login* e a senha. A Figura 5.3 mostra o diagrama de navegação desse processo.

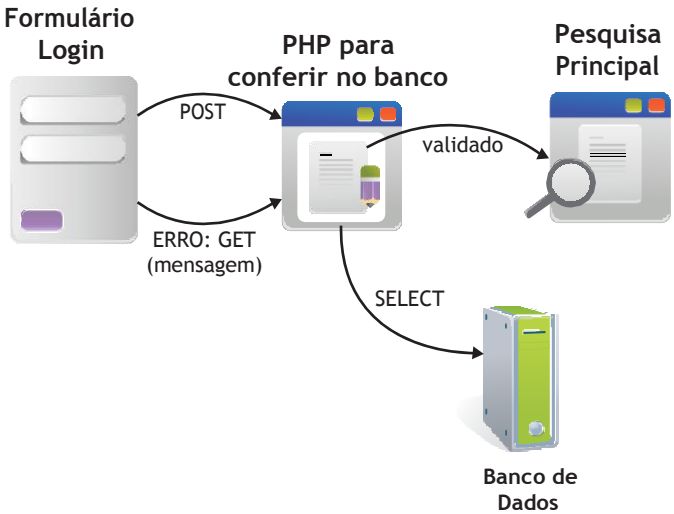


Figura 5.3: Diagrama de navegação para realizar o *login*
Fonte: Equipe produção CEAD/IFES (2011)

No formulário de *login*, vamos colocar o código PHP para exibir a mensagem de erro, caso ocorra. A mensagem ficará acima dos campos. Você pode colocar a mensagem em outro local, caso deseje. O programa da Figura 5.4 mostra o código do formulário **"formlogin.php"** e o programa da Figura 5.5 mostra o PHP **"login.php"**.

```
<html>
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8">
    <title>Formulário de Login</title>
</head>
<body>
<form method="post" name="formLogin" action="login.php">
  <H1 align="center">Acesso aos usuários</H1>

  <?php
     // Exibir mensagem de erro caso ocorra
     if ( isset($_GET["erro"] ) ) {
        $erro = $_GET["erro"];
        echo "<CENTER><FONT color='red'>
              $erro</FONT></CENTER>";
     }
  ?>

  <table align="center">
    <tr>
      <th>Login</th>
      <td><input type="text" name="txtLogin"></td>
    </tr>
    <tr>
      <th>Senha</th>
      <td><input type="password" name="txtSenha"></td>
    </tr>
    <tr>
      <td><input type="submit" value="Logar"></td>
      <td><input type="reset" value="Limpar"></td>
    </tr>
  </table>
</form>
</body>
</html>
```

Figura 5.4: Formulário de *login* ("formlogin.php")
Fonte: Elaborada pelo autor

```php
<?php
   // Pegar os campos do formulário acima
   $login = $_POST["txtLogin"];
   $senha = $_POST["txtSenha"];

   // Montar o SQL para pesquisar
   include_once("conexaobd.php");
   $sql = "SELECT * FROM Clientes WHERE login = '$login' AND senha =
'$senha' ";
   $res = mysql_query( $sql ) or die ("ERRO ao pesquisar login. " .
mysql_error() );

   if (  $registro = mysql_fetch_assoc($res)  ) {
      // Criar a sessão. Login e senha conferem
      $nome = $registro["nome"];
      session_start();
      $_SESSION["login"] = $login;
      $_SESSION["nome"] = $nome;
      header("Location:principal.php");
   } else {
  // Login e senha NÃO conferem
  header("Location:formlogin.php?erro=Login inválido.");
   }
?>
```

Figura 5.5: PHP para conferir *login* e senha no banco ("login.php")
Fonte: Elaborada pelo autor

A página principal é o ponto de entrada do seu sistema. Crie essa página com *link*s para todas as operações que implementamos, como cadastrar e pesquisar cliente, cadastrar e pesquisar produtos, cadastrar e pesquisar vendas, e assim por diante.

5.4.2 Verificar a sessão

Uma vez autenticado o usuário e criada a sessão, todas as outras páginas, no seu início, irão validar se a sessão foi criada, para evitar que uma página seja acessada sem passar pelo processo de *login*. Uma página somente é acessada se existe uma sessão. Uma sessão existe somente se o usuário foi autenticado.

Para verificar se o usuário foi autenticado, basta verificar se a variável ***login*** está registrada na sessão, pois ela somente é adicionada quando o *login* e a senha são validados. Se não estiver, é porque o usuário não está logado; nesse caso, redirecione para o formulário de *login*.

Como todas as páginas devem verificar se o usuário está logado, vamos colocar esse código em um arquivo separado, e todas as páginas incluem-no com o comando **include_once**. O programa da Figura 5.6 mostra esse código, chamado **"validar.php"**.

```php
<?php
   session_start();
   if ( ! isset ( $_SESSION["login"] ) ) {
     // Se a variável de sessão "login" NÃO existe
     // é porque a sessão NÃO foi criada, ou seja,
     // o usuário NÃO foi validado
     // Redireciona para o formulário de login
     header("Location:formlogin.php?erro=Usuário não logado.");
    }
?>
```

Figura 5.6: Verificar se o usuário está logado ("validar.php")
Fonte: Elaborada pelo autor

Em todas as páginas do seu sistema, basta incluir o **"validar.php"** no início, antes de qualquer coisa, e da seguinte forma, conforme Figura 5.7.

```
<?php
    include_once("validar.php");
?>
```

Figura 5.7: Incluir o programa de validação de sessão em cada página
Fonte: Elaborada pelo autor

5.4.3 Fazer *logout*

Para sair do seu sistema, basta colocar um *link* "sair" ou "*logout*" na página principal, que aponte para o PHP do programa da Figura 5.2.

Com essas páginas, completamos nosso *site* com a autenticação de usuários, muito utilizada nos *sites* de hoje em dia. Mas a sessão não serve apenas para isso. Ela pode ser usada para a troca de informação entre as páginas. Uma página adiciona uma informação na sessão e outra página a recupera. Isso pode ser feito nas páginas de alteração de dados, na Aula 4, para transmitir o código do cliente e seus dados para as páginas que exibirão e alterarão os dados do cliente. Tente você fazer essa mudança.

Resumo

Nesta aula, você viu tudo o que precisava para criar sistemas *web* com acesso a banco de dados e autenticação de usuários. Porém a programação que fizemos misturou várias linguagens em um único arquivo. Na próxima aula, veremos uma metodologia para organizar melhor os arquivos, separando a programação da parte visual (HTML e CSS), regras do negócio (PHP) e manipulação do banco de dados (SQL). A seguir, você encontra na Tabela 5.1 os principais comandos para trabalhar com sessões.

Tabela 5.1: Principais comandos para trabalhar com sessões

FUNÇÃO	DESCRIÇÃO	PARÂMETROS
session_start	Cria uma nova sessão ou acessa uma já criada.	
session_destroy	Destrói a sessão.	
$_SESSION	*Array* com os dados compartilhados pela sessão.	Nome da variável entre colchetes.
unset	Elimina UMA variável da sessão.	Nome da variável da sessão.
session_unset	Eliminam TODAS as variáveis da sessão.	

Fonte: Adaptado Soares (2007)

Atividades de aprendizagem

Implemente o processo de autenticação do usuário para todas as páginas do seu sistema, como mostrado acima. Crie a página principal com os *link*s para todas as suas operações e para a operação sair (*logout*). Tente acessar diretamente uma página, sem se logar, para ver o que acontece.

Aula 6 – Caso de uso: aplicação utilizando o padrão MVC

Objetivos

Conhecer o padrão de desenvolvimento de sistemas *web* MVC.

Construir um sistema web no padrão MVC.

Chegamos à nossa última aula de nossa disciplina. Esta Aula é voltada, principalmente, para aqueles que irão trabalhar com desenvolvimento de sistemas, seja *web* ou não.

Vamos aprender a organizar melhor o código para que o processo de desenvolvimento seja produtivo, com melhor qualidade e com menos erros. E caso tenha erros, que sejam de fácil correção. Para isso, vamos aprender uma metodologia que separa o sistema em camadas, o MVC.

6.1 O que é MVC?

O padrão MVC (do inglês *Model-View-Controller*) separa os dados (*Model*) do *layout* (*View*). Dessa forma, alterações feitas no *layout* não afetam a manipulação de dados, que por sua vez poderão ser reorganizados sem alterar o *layout*. O problema é resolvido introduzindo-se um componente entre a manipulação dos dados e a apresentação: o fluxo da aplicação (***Controller***).

A-Z

Controller
O *controller* (controlador) é responsável por transformar eventos gerados pela interface em ação de negócios.

O MVC é usado em padrões de projeto de *software*, mas abrange mais a arquitetura de uma aplicação do que é típico para um padrão de projeto.

6.1.1 Vantagens

Como o modelo MVC gerencia múltiplos visualizadores usando o mesmo modelo, é fácil manter, testar e atualizar sistemas múltiplos. Como a parte visual é separada do modelo de negócio, é possível alterar a parte visual sem alterar o sistema todo.

É muito simples adicionar novas funcionalidades apenas incluindo seus visualizadores e controles sem alterar o que já foi feito, tornando a aplicação

escalável. É possível ter desenvolvimento em paralelo para o modelo, visualizador e controle, pois são independentes, ganhando em produtividade. Dentre as principais vantagens podemos destacar:

- reaproveitamento de código;
- facilidade de manutenção;
- integração de equipes e/ou divisão de tarefas;
- camada de persistência independente;
- implementação de segurança;
- facilidade na alteração da interface da aplicação;
- aplicação escalável.

6.1.2 Desvantagens

As vantagens superam em muito as desvantagens, mas mesmo assim podemos citar alguns pontos desfavoráveis:

- requer uma quantidade maior de tempo para analisar e modelar o sistema;
- requer pessoal especializado;
- não é aconselhável para pequenas aplicações.

6.2 Estrutura do MVC

O MVC possui três componentes: modelo, visão e controlador (veja Figura 6.1).

a) **Modelo** – representa os dados da aplicação e as regras de negócio que governam o acesso e a modificação dos dados. O modelo fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo. Pode ainda ser subdividido em "regras do negócio" e "persistência dos dados".

b) **Visão** – renderiza o conteúdo de uma parte particular do modelo e encaminha para o controlador as ações do usuário; acessa também os dados do modelo via controlador e define como esses dados devem ser apresentados. Apresenta os dados para o usuário sem se preocupar com a origem deles.

c) **Controlador** – define o comportamento da aplicação. É ele que interpreta as ações do usuário e as mapeia para chamadas do modelo. Com base na ação do usuário e no resultado do processamento do modelo,

o controlador seleciona uma visualização a ser exibida como parte da resposta à solicitação do usuário. Há, normalmente, um controlador para cada conjunto de funcionalidades relacionadas.

Figura 6.1: Estrutura do MVC
Fonte: Equipe produção CEAD/IFES (2011)

6.3 Alterando nosso sistema para o MVC

As páginas do nosso sistema estão todas misturadas. Temos páginas com a interface visual, regras de negócio e manipulação de dados em um único local. Isso dificulta a produtividade do processo de desenvolvimento e, caso tenha erro, fica mais difícil encontrá-lo.

Como vocês mesmo podem ter observado, algumas páginas possuem várias linguagens misturadas, como: o HTML para a parte visual; o PHP para a validação e regras de negócio; e o SQL para a persistência de dados.

Nossos exemplos foram desenvolvidos dessa forma de propósito. Primeiro, para mostrar como seria o raciocínio lógico do processo de páginas *web*. E depois para que você possa comparar com uma arquitetura comprovadamente melhor para desenvolvimento de sistemas.

Portanto, vamos agora separar nossos códigos. Tudo que for para a interface visual, seja para entrada de dados ou para a saída de dados (por exemplo, mensagens de erro), será separado na camada da **visão**. Criaremos um arquivo, chamado **"controlador.php"** que irá receber todas as requisições da interface gráfica, fazer a validação dos dados e chamar as funções correspondentes à camada do **modelo**, que por sua vez, ficarão em arquivos separados. Na camada do **modelo** ficarão as funções em PHP referentes ao que o sistema faz, ou seja, as suas operações.

Para que o sistema fique independente do banco de dados a ser usado, criaremos uma camada, chamada **persistência**, que fará o mapeamento das funções do modelo para o banco de dados MySQL. Se o banco de dados mudar, o que pode acontecer com qualquer sistema, basta alterarmos apenas as funções da camada de persistência, sem ter que alterar todas as páginas do sistema.

6.3.1 Persistência dos dados

Vamos começar separando as funções referente ao acesso ao banco de dados MySQL. Essas funções são todas aquelas que começam com "**mysql_**". Vamos mapear as funções da camada do modelo nas funções do MySQL. Poderíamos fazer uma camada de persistência para cada tipo de banco de dados. O programa da Figura 6.2 mostra como seria essa persistência.

```php
<?php

// Este arquivo implementa a camada de persistência
//   para o banco de dados MySQL.

/* Faz a conexão com o SGBD */
function conectarSGBD($host, $login, $senha){
    $a = mysql_connect($host, $login, $senha) or die ("Falha na
conexão com o SGBD. Host = $host");
    return $a;
}

/* Desfaz a conexão com o SGBD */
function disconectarSGBD($link){
    mysql_close($link);
}

/* Seleciona o Banco de Dados */
function selecionarBD($bancoDados){
    mysql_select_db($bancoDados) or die ("Falha na seleção do banco:
$bancoDados");
}

function inserirBD($sql){
    mysql_query($sql) or die("Falha na execução da consulta: $sql");
}

function alterarBD($sql){
    mysql_query($sql) or die("Falha na execução da consulta: $sql");
}

function excluirBD($sql){
    mysql_query($sql) or die("Falha na execução da consulta: $sql");
}

/* Retorna a linha na qual o marcador do resultado da consulta está
apontando */
function obterLinha($result_set){
    $a = mysql_fetch_assoc($result_set);
    return $a;
}

/* Retorna o número de linhas do resultado da consulta */
function obterNumLinhas($result_set){
    $a = mysql_num_rows($result_set);
    return $a;
}

?>
```

Figura 6.2: Camada de persistência dos dados ("persistencia.php")
Fonte: Elaborada pelo autor

Se for usar outro banco de dados, basta implementar outra camada, alterando o código dentro de cada função. Por exemplo, se o banco fosse o PostgreSQL, as funções seriam: pq_connect, pg_select_bd, pg_query, etc.

6.3.2 Modelo de negócio

Nesta camada ficam as funções referentes aos casos de uso do sistema, ou seja, as funcionalidades que o sistema terá. Por exemplo, as operações que fazemos com o cliente são: incluir, alterar, excluir e pesquisar. Então, cada uma dessas operações será transformada em uma função. Podemos separar as funções de acordo com os elementos do sistema. Por exemplo, todas as operações referentes ao cliente ficarão em um arquivo chamado de **"clientes.php"**. As operações sobre os produtos ficarão em outro arquivo, e assim por diante.

A camada do **modelo de negócio** não gera nenhum comando em HTML. O retorno, sucesso ou fracasso é retornado para o controlador que será responsável por montar a página de resposta.

O programa da Figura 6.3 mostra como ficarão as operações com os clientes. É o mesmo código já visto, porém em nova arquitetura. Perceba que quando for necessário acessar o banco de dados, as funções chamadas serão as da camada de persistência do programa da Figura 6.2.

```php
<?php

// Este arquivo implementa o caso de uso do Cliente

// Inclusão dos arquivos da camada de persistência
include_once("persistencia.php");

// Constantes utilizadas
define("COD_INVALIDO", "0");
define("SUCESSO_CLIENTE", "1");


//######### Valida os campos dos clientes ############
function validarCampos($nome, $ender, $cpf, $estado, &$dtNasc, $sexo,
&$cinema, &$musica, &$info, $login, $senha1, $senha2){
   // Verificar campos
   $mensagem = "";    // Armazena a mensagem de erro
   if ( $nome == "" ) {
      $mensagem = $mensagem . "Informe o NOME. <BR>";
   }
   if ( $ender == "" ) {
      $mensagem = $mensagem . "Informe o ENDEREÇO.<BR>";
   }
   if ( $login == "" ) {
      $mensagem = $mensagem . "Informe o LOGIN. <BR>";
   }
   // Verificar se as SENHAS conferem
   if ( $senha1 == "" ) {
     $mensagem = $mensagem . "SENHAS não conferem!<BR>";
   }
   // Verificar se as SENHAS conferem
   if ( $senha1 != $senha2 ) {
     $mensagem = $mensagem . "SENHAS não conferem!<BR>";
   }
   // *** Acrescentar as validações de CPF

   // #### Codigo para converter DATA de
   // "dd/mm/aaaa" para "aaaa-mm-dd"
   if ( strlen($dtNasc) == 10 ) {
      $dia = substr($dtNasc,0,2);
      $mes = substr($dtNasc,3,2);
      $ano = substr($dtNasc,6,4);
      $dtNasc = "$ano-$mes-$dia";
      if ( ! checkdate($mes, $dia, $ano) ) {
         $mensagem = $mensagem . "DATA inválida. <BR>";
      }
   } else {
      $mensagem = $mensagem . "DATA inválida. <BR>";
   }
   // Converte de vazio para ZERO (falso)
   if ( $cinema == "" ) $cinema = 0;
   if ( $musica == "" ) $musica = 0;
   if ( $info == "" ) $info = 0;

   return $mensagem;
}
//######### Inserir o cliente no banco ############
function incluirCliente($nome, $ender, $cpf, $estado, $dtNasc, $sexo,
$cinema, $musica, $info, $login, $senha1, $senha2){
   // Verificar campos
   $mensagem = validarCampos($nome, $ender, $cpf, $estado, $dtNasc,
$sexo, $cinema, $musica, $info, $login, $senha1, $senha2);

   if ( $mensagem == "" ) {  // Não ocorreu erro
      // Conexão com o servidor de banco de dados
      $link = conectarSGBD("127.0.0.1", "root", "");
      selecionarBD("etec");
      // Criação do SQL para inserção do registro do cliente
      $sql = "INSERT INTO Clientes ( nome, cpf, endereco,
Estados_sigla, dtNasc, sexo, login, senha, cinema, musica, info )
      VALUES ( '$nome' , '$cpf' ,  '$ender'  , '$estado', '$dtNasc',
'$sexo', '$login', '$senha1', $cinema, $musica, $info ) ";
      // Execução da consulta
      $resultado = inserirBD($sql);
      return SUCESSO_CLIENTE;
   } else {
      return $mensagem;
   }
}


//######### Alterar os dados do cliente ############
function alterarCliente($cod,$nome, $ender, $cpf, $estado, $dtNasc,
$sexo, $cinema, $musica, $info, $login, $senha1, $senha2){

   // Verificar campos
   $mensagem = validarCampos($nome, $ender, $cpf, $estado, $dtNasc,
$sexo, $cinema, $musica, $info, $login, $senha1, $senha2);

   if ( $mensagem == "" ) {  // Não ocorreu erro
      // Conexão com o servidor de banco de dados
      $link = conectarSGBD("127.0.0.1", "root", "");
      selecionarBD("etec");

      $sql = "UPDATE Clientes SET nome = '$nome',
              cpf = '$cpf', endereco = '$ender',
              Estados_sigla = '$estado',
              dtNasc = '$dtNasc', sexo = '$sexo',
              login = '$login', senha = '$senha1',
              cinema = $cinema, musica = $musica,
              info = $info WHERE idClientes = $cod";

       $resultado = inserirBD($sql);
       return SUCESSO_CLIENTE;
   } else {
      return $mensagem;
   }
}
```

```
//######## Excluir um cliente do banco ############
function excluirCliente($cod){

   if ($cod == "")
      return COD_INVALIDO;
   else{
      // Conexão com o servidor de banco de dados
      $link = conectarSGBD("127.0.0.1", "root", "");
      selecionarBD("etec");
      $sql = "DELETE FROM Clientes WHERE idClientes=$cod";
      $resultado = excluirBD($sql);
      return SUCESSO_CLIENTE;
   }
}

?>
```

Figura 6.3: Operações com os Clientes (“clientes.php”)
Fonte: Elaborada pelo autor

A função **validarCampos** foi separada porque é repetida no **incluir** e **alterar**, que ficaram mais simples e muito parecidas. Todas as funções seguem o mesmo padrão: validar os campos, criar a conexão com o banco de dados e chamar a função da camada de persistência.

6.3.3 Controlador

O controlador será apenas um arquivo PHP. Ele receberá da interface visual o código da operação e os dados, e chamará a função correspondente da camada do modelo, retornando a mensagem de erro ou sucesso para a página que originou a requisição.

Todas as páginas de interface visual deverão ter um campo que informa qual operação o controlador chamará.

Em nossos exemplos, esse campo será chamado de **operação**. As possíveis operações serão todas aquelas da camada do modelo, por exemplo, as funções do arquivo **clientes.php** do programa da Figura 6.3: incluirCliente, alterarCliente, excluirCliente, etc.

O **inserir** e **alterar** – enviam os dados para o servidor com o método POST e o **excluir** envia com o método GET. Para padronizar o controlador, vamos adotar o método GET. Portanto, os formulários deverão ser alterados. O programa da Figura 6.4 mostra um exemplo da estrutura do controlador.

```php
<?php

// Inclusão dos arquivos da camada do modelo
include_once("clientes.php");
//include_once("produtos.php");

// Obtem a operacao passada pela camada de visão
$operacao = $_GET["operacao"];

//Verifica qual operacao tratar

// ############## INCLUIR CLIENTE ####################
if ($operacao == "incluirCliente"){
   // Recebe cada campo do formulário
   // e coloca em uma variável.
   $nome = $_GET["txtNome"];
   $ender = $_GET["txtEndereco"];
   $cpf = $_GET["txtCPF"];
   $estado = $_GET["listEstados"];
   $dtNasc = $_GET["txtData"];
   $sexo = $_GET["sexo"];
   $cinema = $_GET["checkCinema"];
   $musica = $_GET["checkMusica"];
   $info = $_GET["checkInfo"];
   $login = $_GET["txtLogin"];
   $senha1 = $_GET["txtSenha1"];
   $senha2 = $_GET["txtSenha2"];

   // Chama o método de inclusão de cliente
   //     implementado em clientes.php
   $retorno = incluirCliente($nome, $ender, $cpf,
       $estado, $dtNasc, $sexo, $cinema, $musica,
       $info, $login, $senha1, $senha2);

   // Com base no retorno, devolve a mensagem
   // ou redireciona para outra página
   if ($retorno == SUCESSO_CLIENTE)
header("Location:formclientes.php?mensagem=Cliente inserido com
sucesso.");
   else     header("Location:formclientes.php?mensagem=$retorno");

// ############## ALTERAR CLIENTE ####################
} else if ($operacao == "alterarCliente") {
   // Recebe cada campo do formulário
   // e coloca em uma variável.
   $cod = $_GET["txtCodigo"];
   $nome = $_GET["txtNome"];
   $ender = $_GET["txtEndereco"];
   $cpf = $_GET["txtCPF"];
   $estado = $_GET["listEstados"];
   $dtNasc = $_GET["txtData"];
   $sexo = $_GET["sexo"];
   $cinema = $_GET["checkCinema"];
   $musica = $_GET["checkMusica"];
   $info = $_GET["checkInfo"];
   $login = $_GET["txtLogin"];
   $senha1 = $_GET["txtSenha1"];
   $senha2 = $_GET["txtSenha2"];

   // Chama o método de alteração de cliente
   //     implementado em clientes.php
   $retorno = alterarCliente($cod, $nome, $ender, $cpf,
            $estado, $dtNasc, $sexo, $cinema, $musica,
            $info, $login, $senha1, $senha2);

   // Com base no retorno, devolve a mensagem
   // ou redireciona para outra página
   if ($retorno == SUCESSO_CLIENTE)
header("Location:formalterarcli.php?codigo=$cod&mensagem=Cliente
alterado com sucesso.");
   else
header("Location:formalterarcli.php?codigo=$cod&mensagem=$retorno");


// ############## EXCLUIR CLIENTE ####################
} else if ($operacao == "excluirCliente") {

   $cod = $_GET["codigo"];

   // Chama o método de excluir cliente
   //      implementado em clientes.php
   $retorno = excluirCliente($cod);

   // Com base no retorno, devolve a mensagem
   // ou redireciona para outra página
   if ($retorno == SUCESSO_CLIENTE)
header("Location:pesqclientes.php?mensagem=Cliente excluido com
sucesso.");
   else     header("Location:pesqclientes.php?mensagem=$retorno");


// ############## OUTRAS OPERAÇÕES ##################

} else if ($operacao == "incluirServico") {

  /* TRATAMENTO DA OPERACAO DE INCLUIR SERVICO */

} else if ($operacao == "alterarServico") {

  /* TRATAMENTO DA OPERACAO DE ALTERAR SERVICO */

} else if ($operacao == "excluirServico") {

  /* TRATAMENTO DA OPERACAO DE EXCLUIR SERVICO */
}

/* E ASSIM POR DIANTE */
?>
```

Figura 6.4: Estrutura do controlador (“controlador.php”)
Fonte: Elaborada pelo autor

6.3.4 Camada da visão

Páginas COM formulário – para as páginas com formulário (em nosso exemplo: **formclientes.php, formaltcli.php** e **login.php**) as mudanças serão as seguintes: a primeira é passar a enviar os dados para o controlador. Para padronizar o controlador, os formulários também passarão os dados via GET. A segunda é colocar um campo escondido para definir qual a operação que o controlador fará. A terceira é colocar a mensagem de resposta fornecida de volta pelo controlador. Para enviar os dados para o controlador com o método GET, basta alterar os atributos **action** e **method**, da seguinte forma, conforme a Figura 6.5.

```
<FORM name="cadastro" action=" controlador.php"
      method="GET" >
```

Figura 6.5: Alterando o método de envio do formulário para GET
Fonte: Elaborada pelo autor

A segunda mudança, conforme já mencionado, é colocar um campo escondido para indicar ao controlador qual a operação ele fará. Como exemplo, chamaremos esse campo de **operação**. O valor a ser passado (atributo **value**) deverá ser o mesmo valor com o qual o controlador fará a comparação. Cada operação deverá ter um valor diferente. Por exemplo: “incluirCliente”, “alterarCliente”, “excluirCliente”, etc., conforme a Figura 6.6.

```
<input type="hidden" name="operacao"
       value="incluirCliente" >
```

Figura 6.6: Campo operação oculto (*hidden*)
Fonte: Elaborada pelo autor

A terceira mudança, conforme já mencionado, é colocar em algum lugar da sua página a mensagem de resposta, caso tenha, de sucesso ou falha da operação. Essa mensagem será enviada do controlador para a página, usando o método GET. Em nosso exemplo, veja o controlador, o nome do campo foi chamado de **mensagem**. O código em PHP a ser inserido será como o apresentado na Figura 6.7 a seguir.

```
. . .
<?php
    // Exibir a mensagem de erro ou sucesso
    if ( isset($_GET["mensagem"]) ) {
      $msg = $_GET["mensagem"];
      echo "<FONT color='red'>$msg</FONT>";
    }
?>
. . .
```

Figura 6.7: Exibindo a mensagem de resposta
Fonte: Elaborada pelo autor

Páginas SEM formulário –para as páginas que não possuem formulário, apenas *link*s para outras páginas, enviando os campos via método GET, as mudanças serão apenas alterar o *link* para o controlador e adicionar o campo que corresponde à operação. Em nosso exemplo, a única mudança será no *link* do **excluir** da página de pesquisa **"pesqclientes.php"** (veja a Figura 6.8).

```
. . .
   echo "<TR><TD>$id</TD>
  <TD><A href='formaltcli.php?codigo=$id'>$nome</A></TD>
  <TD>$cpf</TD><TD>$ender</TD><TD>$estado</TD>
  <TD>$dtNasc</TD><TD>$sexo</TD><TD>$login</TD>
  <TD>$cinema</TD><TD>$musica</TD><TD>$info</TD>
  <TD><A href=' controlador.php?operacao=excluirCliente&codigo=$id'>
      <IMG src='excluir.png'></A></TD></TR>";
. . .
```

Figura 6.8: *Link* do excluir apontando para o controlador
Fonte: Elaborada pelo autor

As operações de consulta ao banco também podem ser colocadas no controlador, porém o processo é mais complicado, requer um estudo mais aprofundado do MVC. Por enquanto deixe da forma que se encontra, dentro de páginas da camada da visão. Para quem for trabalhar com desenvolvimento de sistemas *web*, será muito importante apronfudar-se mais nessa metodologia.

Perceba que antes, a cada página de formulário, tínhamos uma página em PHP para realizar aquela operação. Em um sistema maior, a quantidade de arquivos para organizar seria muito grande.

Com o MVC, continuamos com as páginas da interface gráfica, porém as páginas em PHP no servidor ficaram bastante reduzidas. Temos o controlador, o arquivo de persitência e arquivos nos quais podemos agrupar as funções de cada elemento do sistema. Por exemplo, todas as operações referentes a "produto" ficariam em um arquivo, as operações referentes a "vendas" em outro arquivo, e assim por diante.

Resumo

Nesta Aula, conhecemos o padrão de desenvolvimento de sistemas *web* MVC. Vimos como organizar melhor o código para que o processo de desenvolvimento seja mais produtivo, com melhor qualidade e com menos erros. Aprendemos a separar a programação da parte visual (HTML e CSS), regras do negócio (PHP) e manipulação do banco de dados (SQL). No fim, alteramos uma parte do nosso sistema *web* de exemplo para o padrão MVC.

Caro estudante, chegamos ao final desta disciplina. Espero que você tenha gostado e que seja capaz de desenvolver sistemas *web*. Diverti-me criando

Aprender mais sobre MVC é muito importante caso você pretenda trabalhar com programação *web*. Seguem alguns *links*:

http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm

http://codeigniterbrasil.com/passos-iniciais/mvc-model-view-controller/

http://www.tiagolemos.com.br/2009/07/10/o-que-e-o-mvc-model-view-controller/

http://msdn.microsoft.com/en-us/library/ms978748.aspx.

essas páginas e também aprendi um pouquinho mais sobre PHP. Mas isso não é tudo! Temos muito mais recursos em PHP que não daria tempo de estudar nesta disciplina. Caso você se interesse, veja as referências no final de cada aula e na seção “Referências”.

Atividades de aprendizagem

1. Faça as alterações do seu sistema para o MVC. Comece com as operações sobre os clientes, como descrito acima.

2. Acrescente no MVC as operações com os “produtos”. Comece criando o arquivo com as funções referentes a “produtos”. Depois altere o controlador para fazer a ligação das páginas visuais com essas funções.

3. Acrescente no MVC as operações de *login*. No controlador teremos mais um *else if* para a validação do *login* e senha e criação da sessão. Nas demais páginas, coloque o PHP para verificar se a sessão foi criada, como na Aula 5.

Referências

DAVIS, Michele E.; PHILLIPS, Jon A. **Aprendendo PHP e MySQL**. Rio de Janeiro: Alta Books, 2008.

SOARES, Walace. **PHP 5**: conceitos, programação e integração com banco de dados. 3ª ed. São Paulo: Érica, 2007.

GUTMANS, Andi; BAKKEN, Stig Saether; RETHANS, Derick. **PHP 5:** programação ponderosa. Rio de Janeiro: Alta Books, 2005.

MELO, Alexandre Altair de; NASCIMENTO, Maurício G. F. **PHP profissional**. 2ª ed. São Paulo: Novatec, 2007.

MILANI, André. **Construindo aplicações *web* com PHP e MySQL**. São Paulo: Novatec, 2010.

NIEDERAUER, Juliano. **Desenvolvendo *websites* com PHP**. 2ª ed. São Paulo: Novatec, 2004.

WELLING, Luke; THOMSON, Laura. **PHP e MySQL**: **desenvolvimento web**. 3. ed. Rio de Janeiro: Elsevier, 2005.

Currículo do professor-autor

Graduado em Ciência da Computação pela Universidade Federal de Viçosa-MG (1997) e mestre em Engenharia de Produção/Pesquisa Operacional pela UFRJ/ COPPE (2000). Trabalhou três anos como Analista de Sistemas e líder de projeto pela DBA Engenharia de Sistemas prestando serviços de desenvolvimento de sistemas para a Caixa Econômica Federal, no Rio de Janeiro. Foi professor do Curso de Ciência da Computação da Faculdade de Aracruz (FACHA, hoje FAA-CZ) de 2002 a 2007. É professor efetivo do Instituto Federal do Espírito Santo (IFES) desde 2006, lecionando disciplinas para os cursos: Técnico em Informática, Superior de Tecnologia em Redes de Computadores e Sistemas de Informação, onde atua como coordenador. Na EaD, trabalha desde 2008 como professor-autor-formador e tutor a distância nos cursos do IFES: Tecnólogo em Análise e Desenvolvimento de Sistemas e Técnico em Informática. Para mais informações, veja o currículo completo na Plataforma Lattes:

http://lattes.cnpq.br/4033155608776526